convertigo
MASHUP YOUR BUSINESS

Reference Manual

Object properties and APIs

CEMS 7.2.0

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introducing the Reference Manual

This chapter presents the purpose of the *Reference Manual*, as well as key information about its structure.

- Introduction
- Opening a sample project from the Studio

1.1 Introduction

This *Reference Manual* is the reference material concerning Convertigo detailed technical information.

It contains the following chapters:

- *"Introducing the Reference Manual"* - briefly describes the Reference Manual.
- *"Convertigo Objects"* - lists and describes all Convertigo objects ordered by groups and, for each group, by categories, with their properties.
- *"JavaScript Objects APIs"* - proposes APIs of JavaScript objects that are available in Convertigo transactions and sequences.
- *"Interfaces to Convertigo"* - offers technical information about how to access to Convertigo.
- *"Convertigo Templating Framework"* - presents in details the Templating framework that allows to easily connect UIs to the APIs present in Convertigo server (transactions/ sequences) and to template data in the UIs.
- *"Internationalization framework"* - presents in details the Internationalization framework that allows to easily develop a multi-language UI.

In the *Convertigo Objects* chapter, object icons as displayed in the Convertigo studio appear next to object names. In this chapter, objects are described in three sections:

1. The *Object Description* section contains a textual description of the object, its purpose.
2. The *Object Properties* section contains a table listing object properties and how to configure them:

Table 1 - 1: Object Properties table

Column	Description
Property	Property name
Type	Property type (boolean, string, javascript expression, etc.)
Description	Property description, how to paramter it

3 The *Example* section contains one or several examples of the described object with screenshots.

Example use-cases are based upon Convertigo projects that the user can open in the Convertigo Studio to test live the objects. To open an example project, follow the procedure described in next section *"Opening a sample project from the Studio"* on page 1 - 2.

After the last chapter, you can find *"Appendixes"* in relation with information about certain properties of objects.

1.2 Opening a sample project from the Studio

The full example projects are stored in Convertigo installation folder. The following procedure describes how to access them from the Studio using the **New Project** wizard.

To open an example project from the Studio

1. If not already opened, run the **Convertigo Studio**;
2. In the **Studio** menu bar, select ***File > New > Project*** or click on  in the toolbar then select ***Project***.

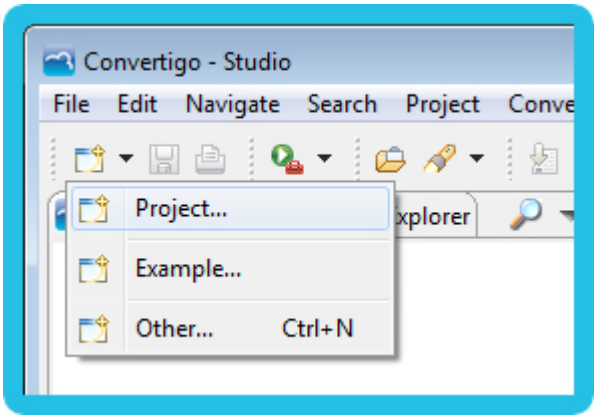


Figure 1 - 1: Launching the New Project wizard

A **New Project** wizard opens.

3. Expand **Convertigo Samples and Demos** category:

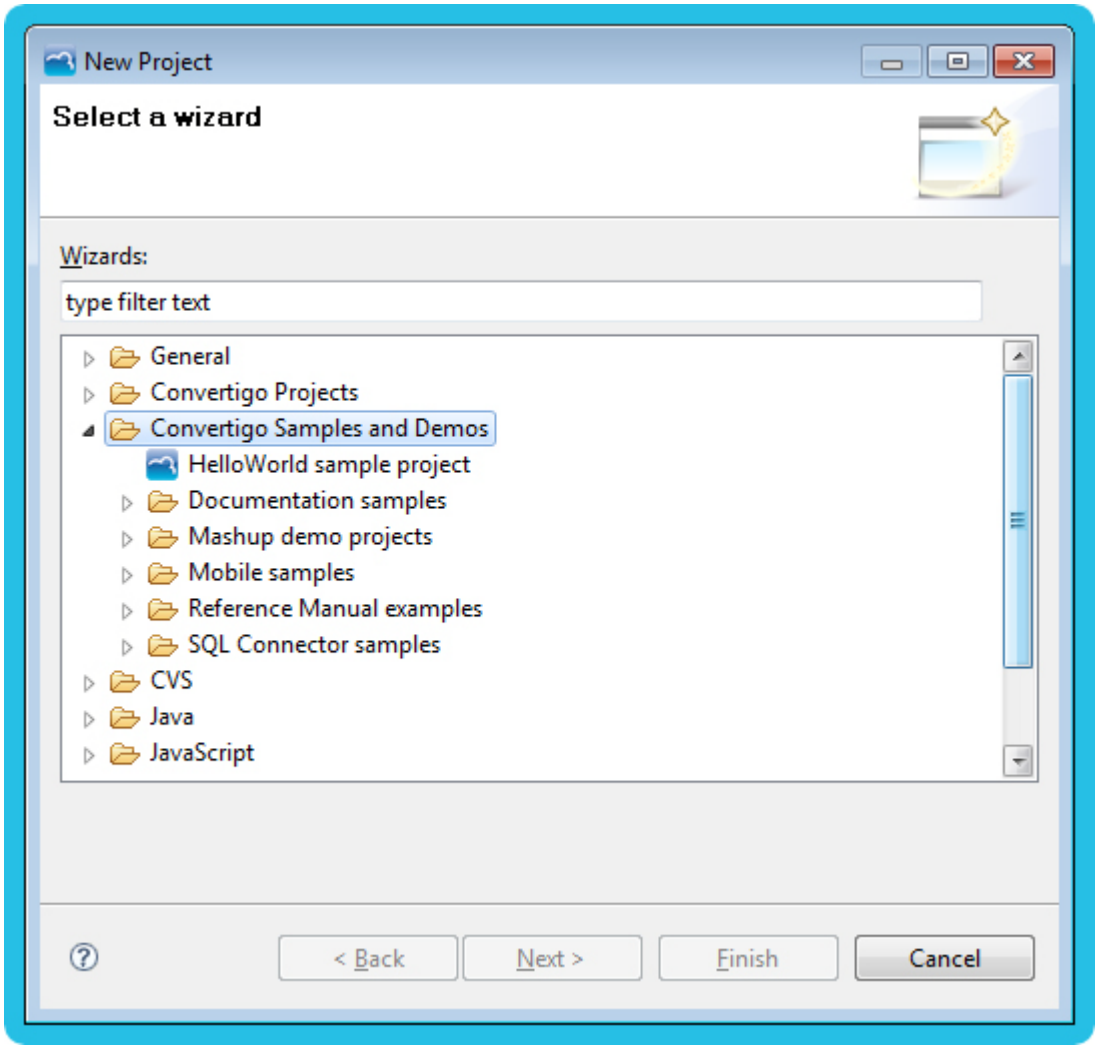


Figure 1 - 2: Opening a sample project in New project wizard

4 Expand **Reference Manual examples** and select the sample project to open depending on the object example you are consulting (some example projects are not yet available in Studio, those showing legacy extraction rules for examples):

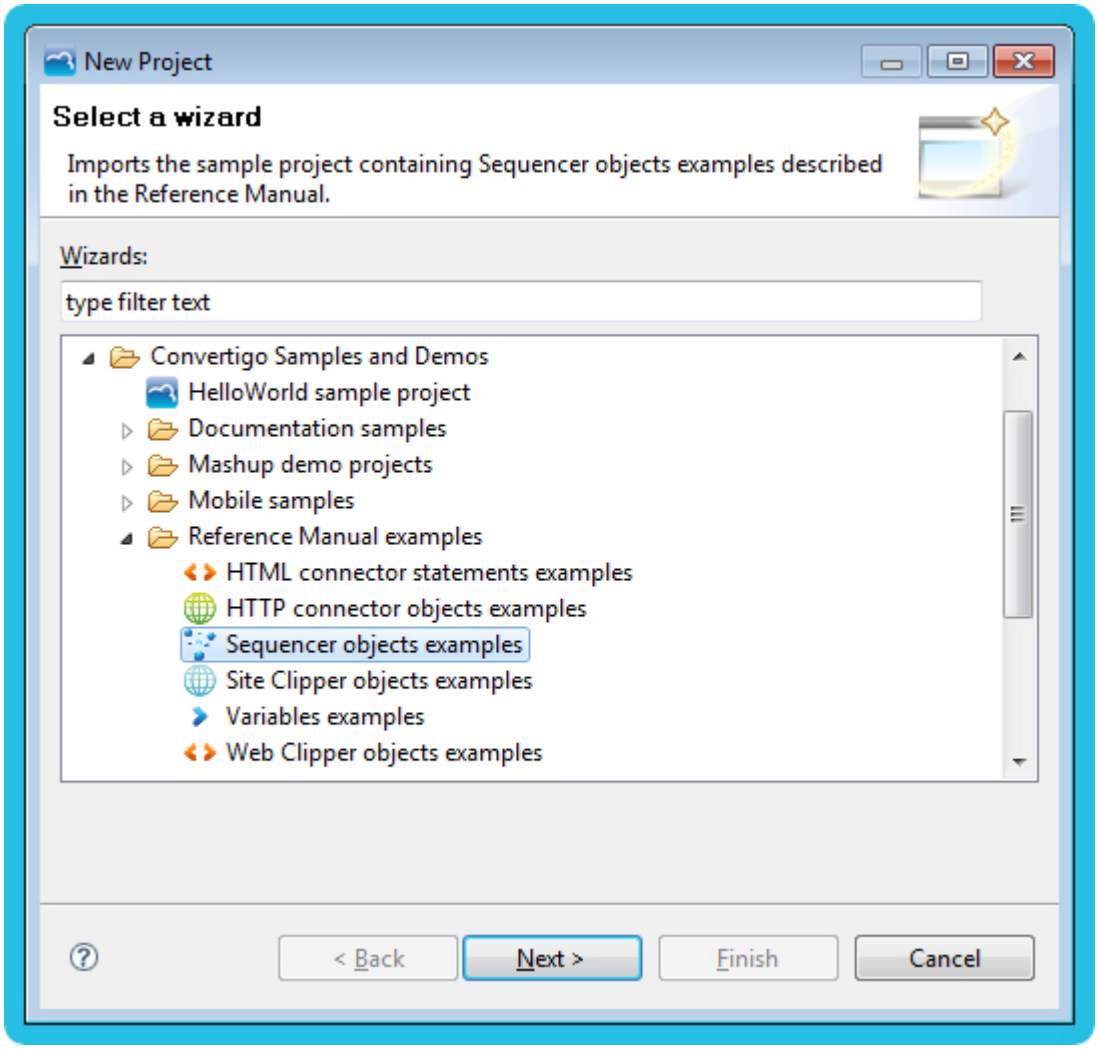


Figure 1 - 3: Selecting appropriate sample project

5 Click on **Next**, then **Finish**.

In the case of Sequencer project referencing transactions from another project, a pop-up indicates that a related project is missing and should be opened before running the project (for example when opening Steps examples project, the Web Integrator documentation sample project should also be opened):

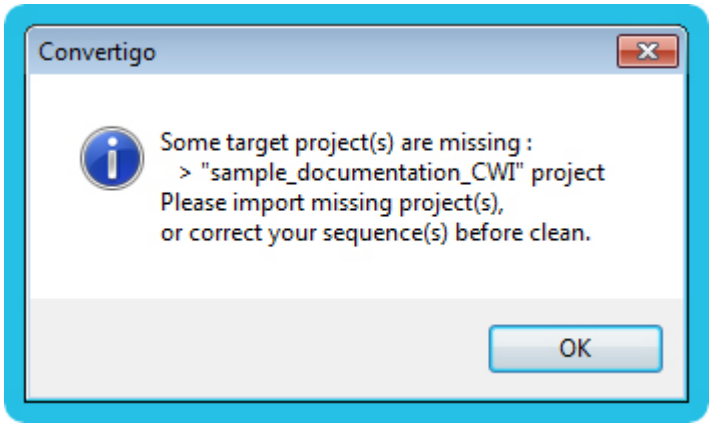


Figure 1 - 4: Pop-up indicating a related project is missing

6 In this case, open the missing project(s) following the same procedure. Beware that some projects can be in **Convertigo Samples and Demos** > **Documentation samples** category in the wizard page:

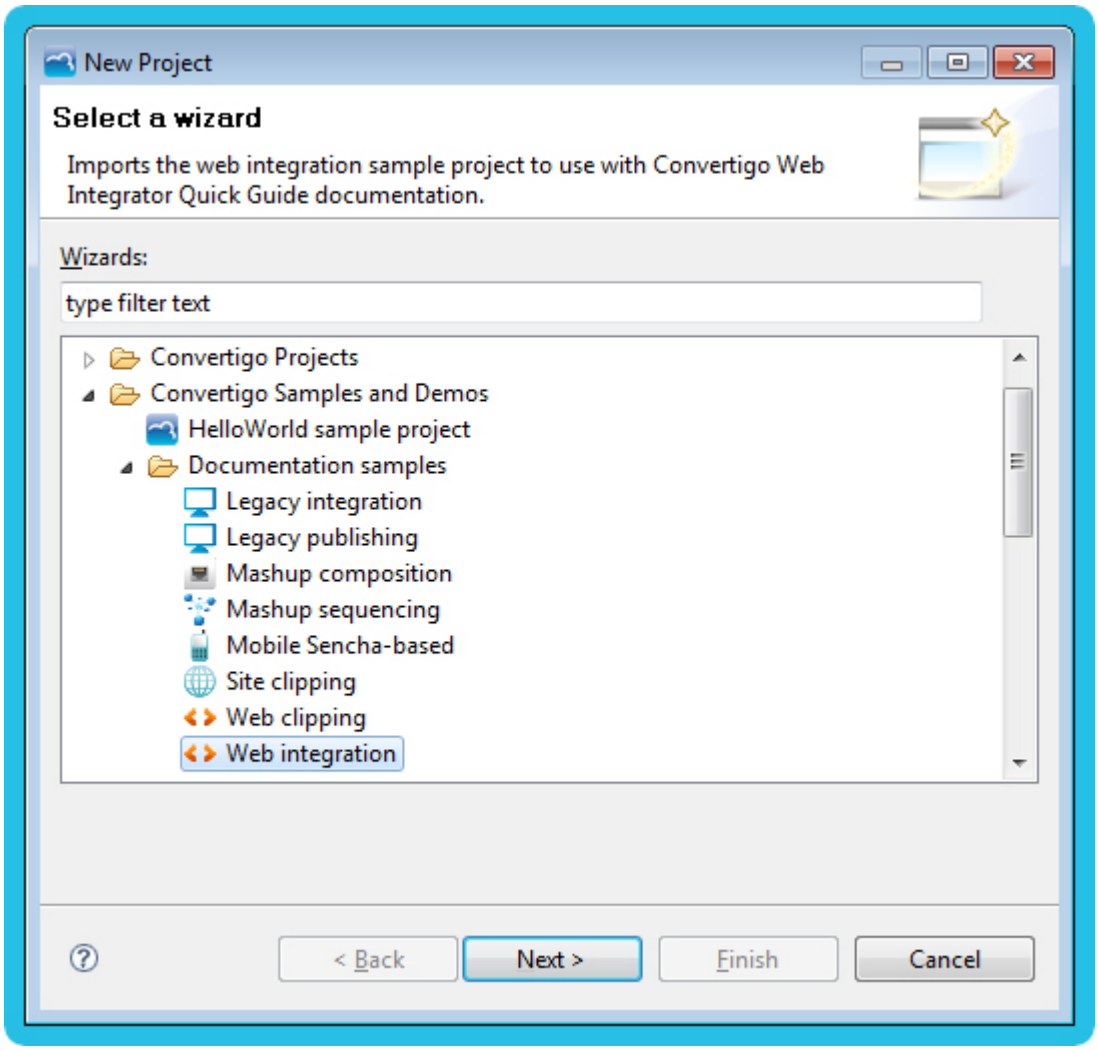


Figure 1 - 5: Opening missing CWI sample project in wizard

The following table lists the names of all projects that could be referenced by other projects and their location in the New Project wizard for you to know which project is missing:

Table 1 - 2: Sample projects names and their location in the New Project wizard

Sample project name	Location in the New Project wizard
sample_documentation_CLI	Convertigo Samples and Demos > Documentation samples > Legacy integration
sample_documentation_CLP	Convertigo Samples and Demos > Documentation samples > Legacy publishing
sample_documentation_CMC	Convertigo Samples and Demos > Documentation samples > Mashup composition
sample_documentation_CMS	Convertigo Samples and Demos > Documentation samples > Mashup sequencing
sample_documentation_CWC	Convertigo Samples and Demos > Documentation samples > Web clipping
sample_documentation_CWI	Convertigo Samples and Demos > Documentation samples > Web integration

Table 1 - 2: Sample projects names and their location in the New Project wizard

Sample project name	Location in the New Project wizard
sample_refManual_http	Convertigo Samples and Demos > Reference Manual Examples > HTTP connector objects examples
sample_refManual_siteClipper	Convertigo Samples and Demos > Reference Manual Examples > Site Clipper objects examples
sample_refManual_statements	Convertigo Samples and Demos > Reference Manual Examples > HTML connector statements examples
sample_refManual_steps	Convertigo Samples and Demos > Reference Manual Examples > Sequencer steps examples
sample_refManual_variables	Convertigo Samples and Demos > Reference Manual Examples > Variables examples
sample_refManual_webClipper	Convertigo Samples and Demos > Reference Manual Examples > Web Clipper objects examples

In both cases, after opening a missing related project or not, the Reference Manual example project is then opened in the Studio. You can see its objects by deploying them in the **Projects** view:

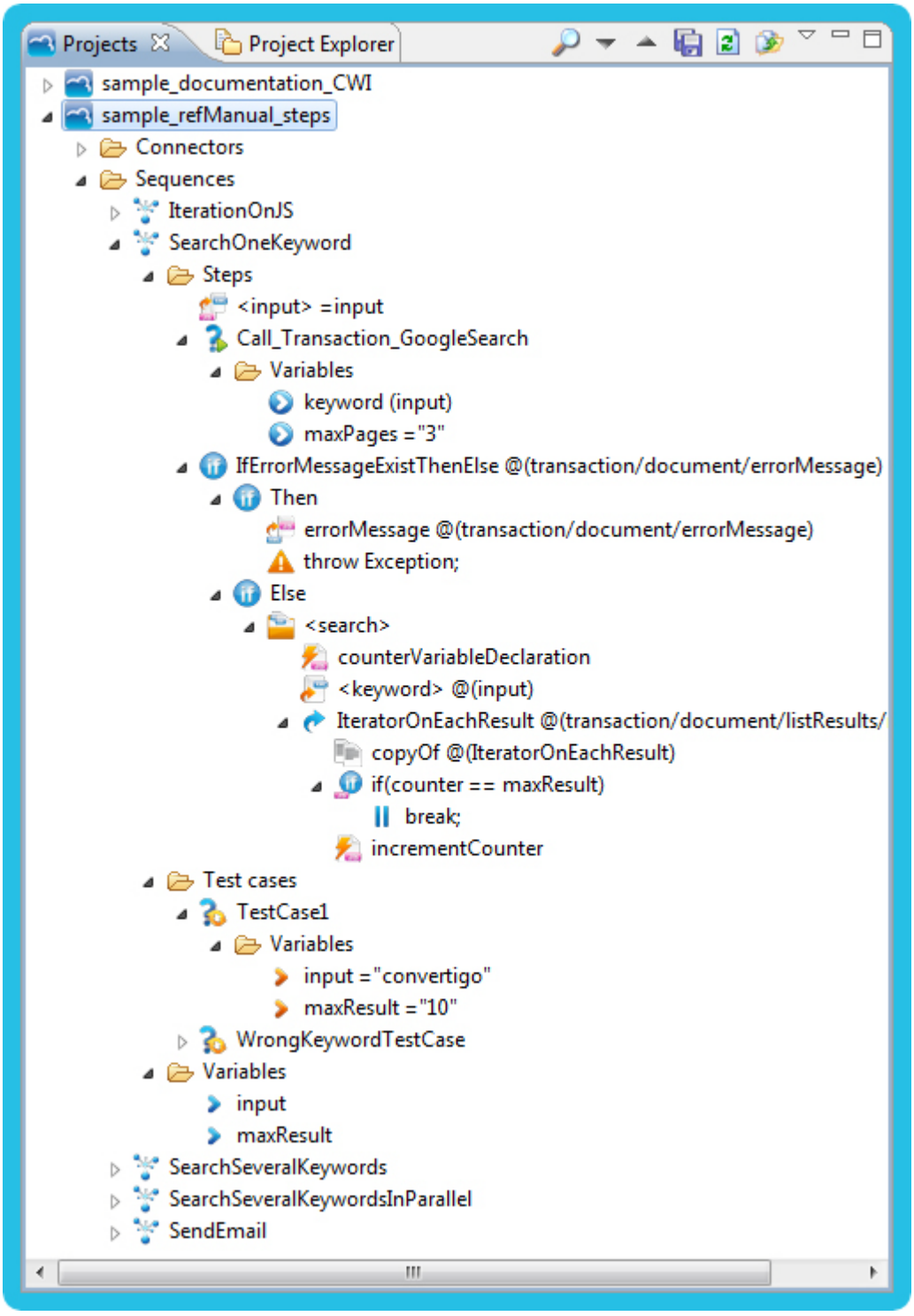


Figure 1 - 6: Reference Manual steps example project in Projects view

2 Convertigo Objects

This chapter lists and describes all Convertigo objects ordered by groups and, for each group, by categories:

- Common
- Mobile Application
- Sequencer
- SAP
- SQL
- CICS
- Web services
- Web
- Legacy
- SiteClipper

2.1 Common

2.1.1 Main objects

PROJECT 

OBJECT DESCRIPTION

Defines a Convertigo project.

The *Project* is the basic entity of a Convertigo project. It contains all Convertigo objects needed for a Convertigo project to run properly:

- connectors,
- screen classes (with criteria, extraction rules and style sheets),
- transactions (with handlers, statements, style sheets, variables and test cases),
- pools,
- sequences (with steps, style sheets, variables and test cases).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Browsers	XMLVector	expert	Defines available browser types. This property is a table describing browser clients able to connect to Convertigo. It is used by Convertigo to choose the proper XSL style sheet for rendering HTML pages (for example for Mobile devices). For each browser signature, the **Browsers definition** table contains two columns: • **Keyword**: Each keyword is evaluated in turn by Convertigo. If any keyword set here is found in the browser signature by Convertigo, it is considered used by the client. • **Label**: Name of the browser used by the client. Note: A new browser definition can be added to the list using the blue keyboard icon. The browser definitions defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Exported date	long	standard	Specifies the last date the project was exported. This property is a read-only field containing a timestamp that is automatically set when saving, exporting or deploying the project. It displays the date and time of the last save of the project. Note: When clicking on this property in the **Properties** view, you can see the exact timestamp value.

Property	Type	Category	Description
HTTP session timeout	int	standard	Defines the inactivity time (in seconds) of incoming HTTP sessions. If no requests are sent from the same user on the project for the set lifetime, Convertigo automatically removes any context existing for this user and frees memory. When the context is freed, the **End Transaction** defined for the connector is automatically executed. You can set in this transaction any clean up code as a `logout` transaction to logout from the target application.
Namespace URI	String	expert	Defines the project's namespace URI to use in XSD and WSDL files. If this property is left empty, the default Convertigo project's namespace URI is used: `http://www.convertigo.com/convertigo/projects/<project_name>` with `<project_name>` the name of the current project.
Version	String	standard	Defines the project's version. This property is an editable field that allows the project's developer to set a project's version. The project version syntax is free, the developer can use it the way he wants. When exporting the project as an archive (`.car`) or deploying the project (on a server for example), the Convertigo Studio user interface proposes to the developer to update this project's version number field. It helps the developer to not forget editing it before a project's delivery.
WSDL inline schemas	boolean	expert	Specifies whether the project's WSDL should be generated by including schemas or not. The WSDL describing the services can import the schemas defined in a separate file or can describe inline these schemas. This property allows to specify the way the Convertigo developer wants schemas for the project's WSDL.
WSDL style	WsdlStyle	expert	Defines the project's WSDL style to use for the project (`DOC/LITERAL`, `RPC`, `ALL`).

EXAMPLES

The Convertigo Studio includes several sample projects described in *Getting started tutorials*, each corresponding to a module of the **Convertigo Enterprise Mobility Server** solution:

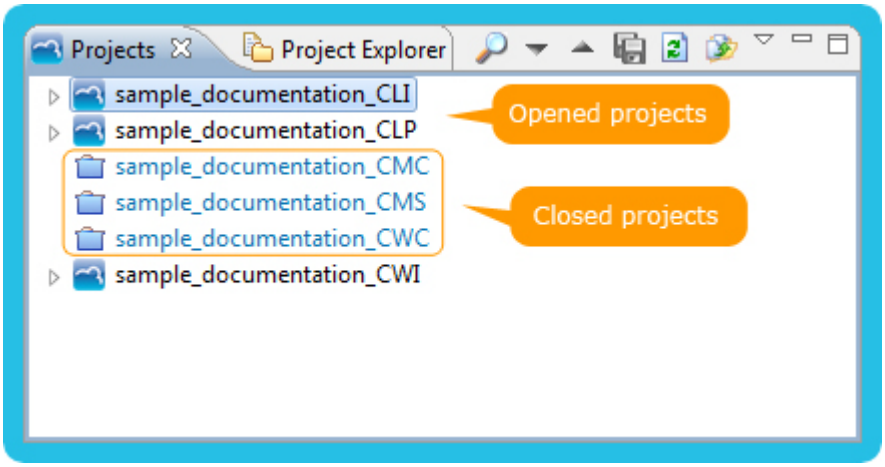


Figure 2 - 1: Project - Documentation sample projects in Projects view

You can find the complete example projects in the Studio. To open these projects, refer to the procedure described in each Getting started tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***either of the available projects*** *in the* ***New Project*** *wizard.*

The following description is an example of `sample_documentation_CLI` *Project* set on the *"Starting with Convertigo Legacy Integrator"* tutorial:

```
Project [
   comment="Legacy Integration project"
   HTTP session timeout=300
   browsers=[]
   WSDL inline schemas=true
   WSDL style=DOC/LITERAL
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

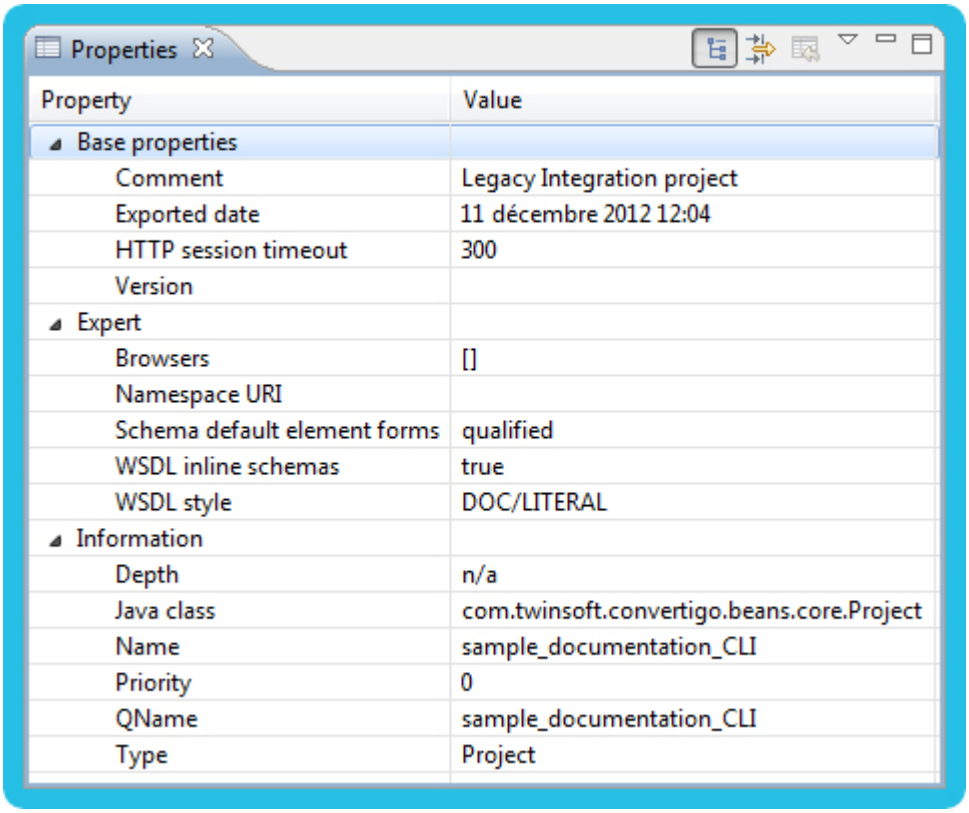


Figure 2 - 2: Project - Configuration example

The **HTTP session timeout** is set to 300 seconds by default, meaning that the session ends when no request is sent for 300 seconds. No browser has been defined in the **Browsers** property.

In case of at least one transaction set as public method in this project, it is exposed as a SOAP Web service (see the *"Starting with Convertigo Legacy Integrator"* tutorial for more information about exposing a project as a SOAP Web service). In this case, **WSDL inline schemas** property set to `true` implies that the transactions schemas are recopied from the XSD schema file to the WSDL, and not in the imported separate XSD file only. The **WSDL style** properties set to `DOC/LITERAL` defines the WSDL generation style, by opposition to `RPC/ENCODED`.

TEST CASE 

OBJECT DESCRIPTION

Defines a transaction's or sequence's test case.

A *Test Case* is a set of variables with predefined values. It allows testing its parent transaction or sequence in any particular case.

Test Cases are based on *Test single-valued variable* and *Test multi-valued variable* variables which are used as input when executing the tested transaction or sequence.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

EXAMPLES

Let's consider the `searchGoogle` transaction set in the context of the "*Starting with Convertigo Web Integrator*" tutorial. This transaction, which defines one variable called `keyword`, searches for this keyword in Google search engine and accumulates the results into an XML structure thanks to a *Table* extraction rule.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

A *Test Case* is defined on the `searchGoogle` transaction in order to test a case of variable value. A *Test Case* object has no properties to configure:

```
Test Case [
]
```

It appears as follows in the **Properties** view of the Convertigo Studio:

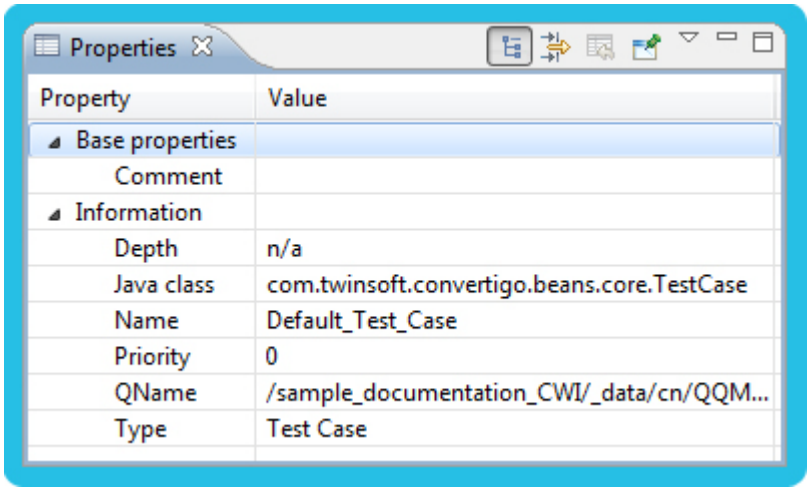


Figure 2 - 3: Test Case - Configuration example

The *Test Case* object, named `Default_Test_Case`, is created in the **Test cases** folder of the transaction, and appears as follows in the **Projects** view:

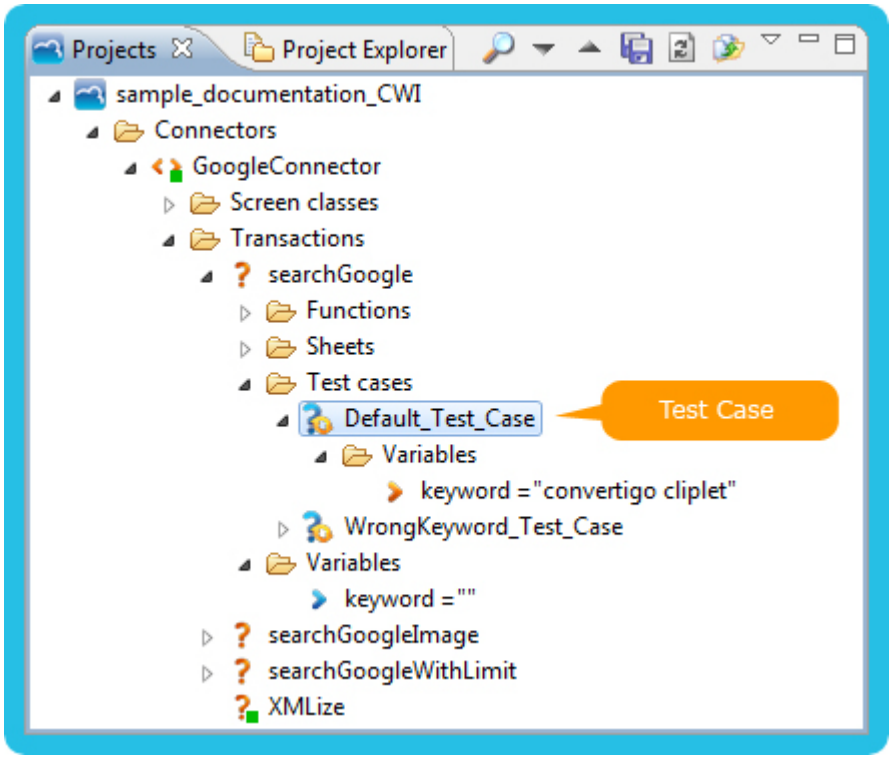


Figure 2 - 4: Test Case - Test Case object in Projects view

To complete the *Test Case*, a *Test single-value variable* is added, imported from the transaction's variables. For more information about *Test single-valued variable*, see "*Test single-valued variable*" documentation and examples.

Switch to a web browser displaying the test platform of this project. The test platform shows the `searchGoogle` transaction, with its `keyword` variable and empty default value. It also shows the `Default_Test_Case` *Test Case* (among other) with its variable and the test default value:

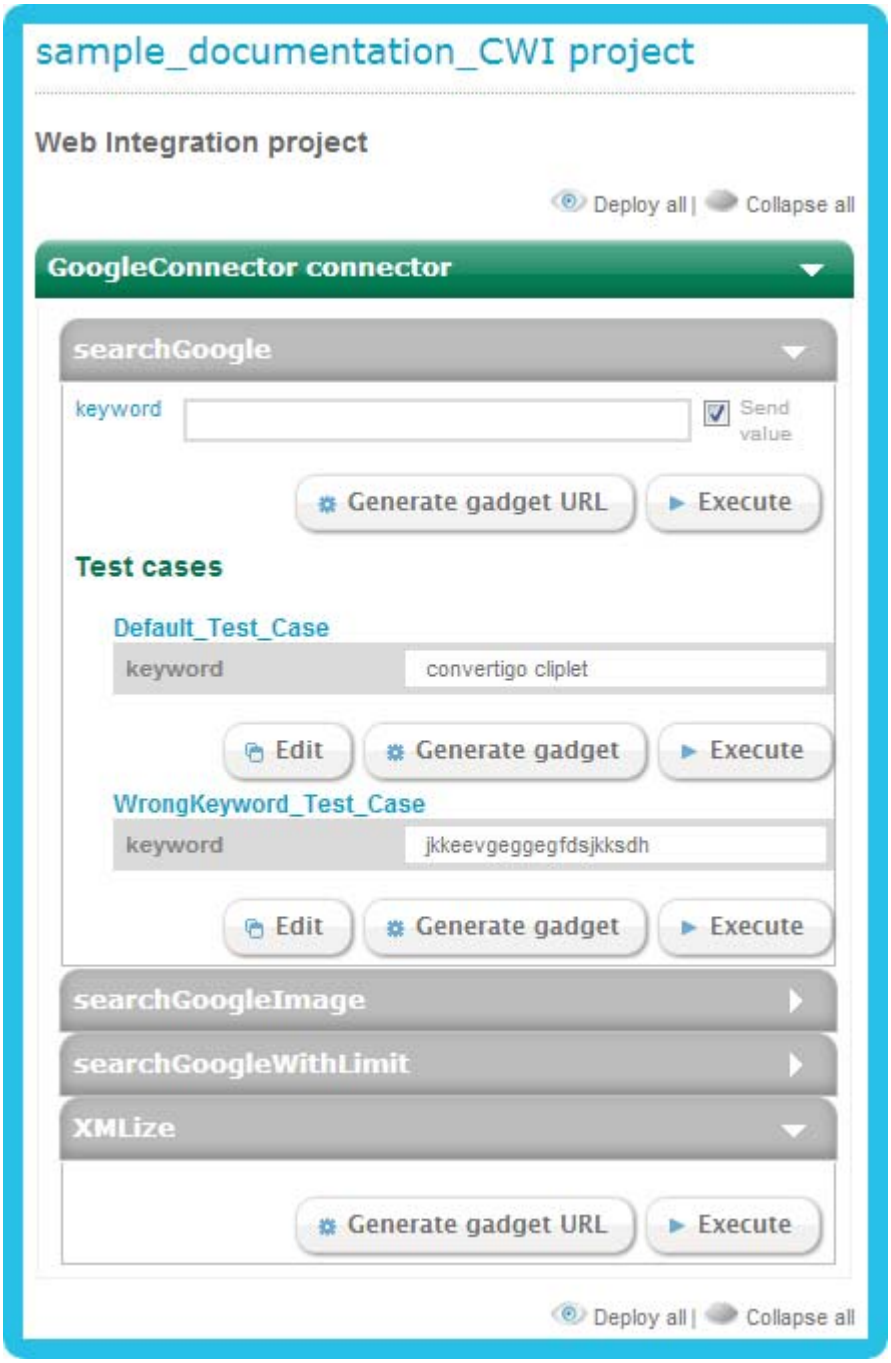


Figure 2 - 5: Test Case - Test Case object in test platform

At runtime, *Test variable* (with its default value) is inserted into the JavaScript scope of the transaction. Thus, when running the `Default_Test_Case` *Test Case* (by clicking on the **Execute** button), the `searchGoogle` transaction is executed with the test variable default value "`convertigo cliplet`". The result is displayed in the **Execution result** panel:

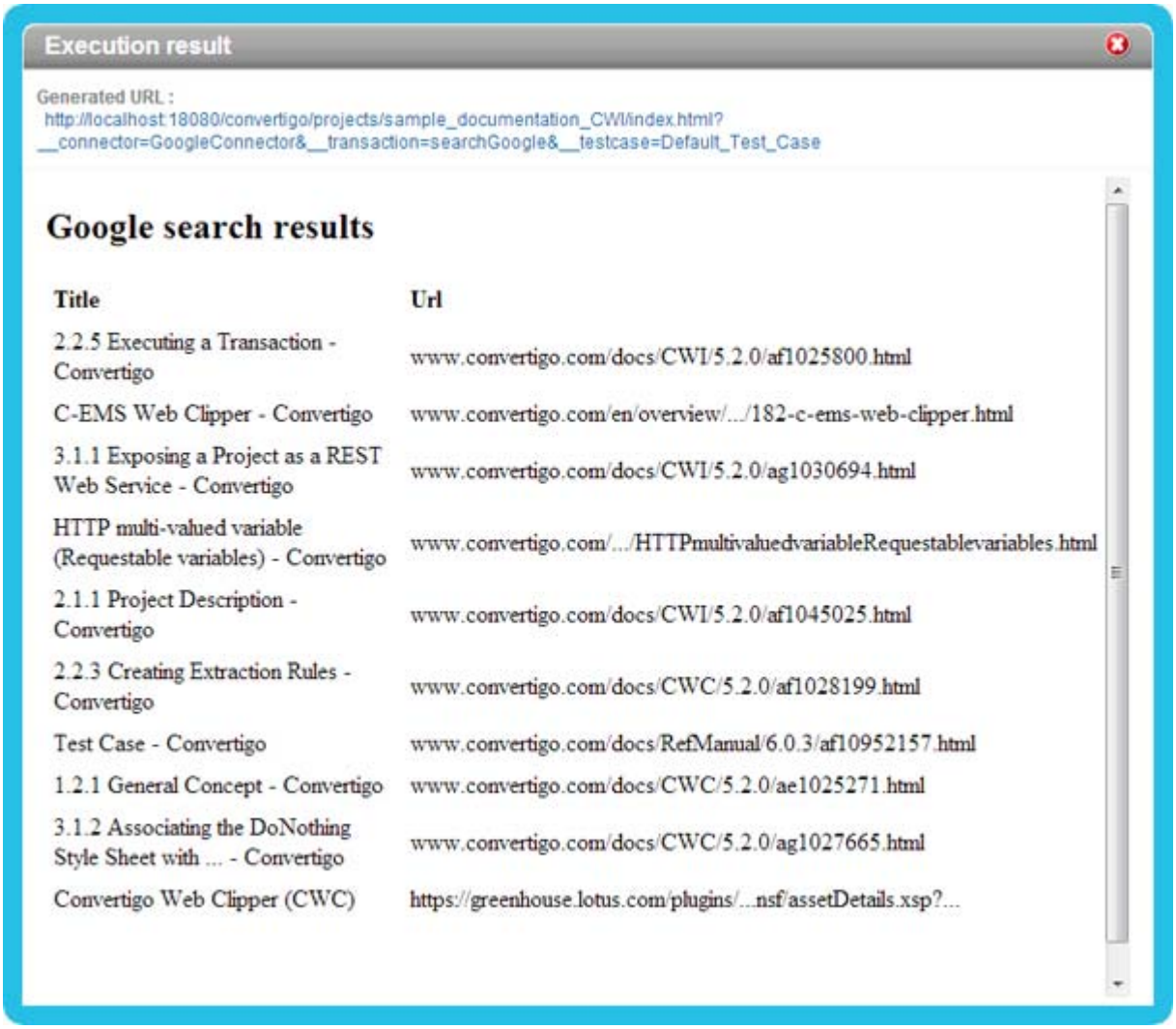


Figure 2 - 6: Test Case - Test Case execution result in test platform

STYLE SHEET 

OBJECT DESCRIPTION

Defines an XSL style sheet.

Style sheets are associated with screen classes or transactions and with sequences. They are used to define the display of data collected by the transaction or sequence, by performing a transformation of transaction or sequence XML outputs.

Moreover, *Style sheets* allow the Convertigo developer to manage the user interface's display depending on the client browser.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Browser	String	standard	Defines the target browser (* means all browsers). This property allows to define on which browser the *Style sheet* has to be applied. The list of available browsers is declared in the project's **Browsers** property. One *Style sheet* can be created for each declared browser on the same parent object (screen class, transaction or sequence).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
URL	String	standard	Defines the *Style sheet* URL (relative or absolute). This property allows to define the path to the XSL file represented by this *Style sheet* object.

EXAMPLES

Let's consider the `searchGoogle` transaction set in the context of the "*Starting With Convertigo Web Integrator*" *tutorial*. This transaction, which defines one variable called `keyword`, searches for this keyword in Google search engine and accumulates the results into an XML structure thanks to a *Table* extraction rule.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

When no *Style sheet* is attached to the `searchGoogle` transaction, its XML output is displayed in plain XML (here, in the test platform):

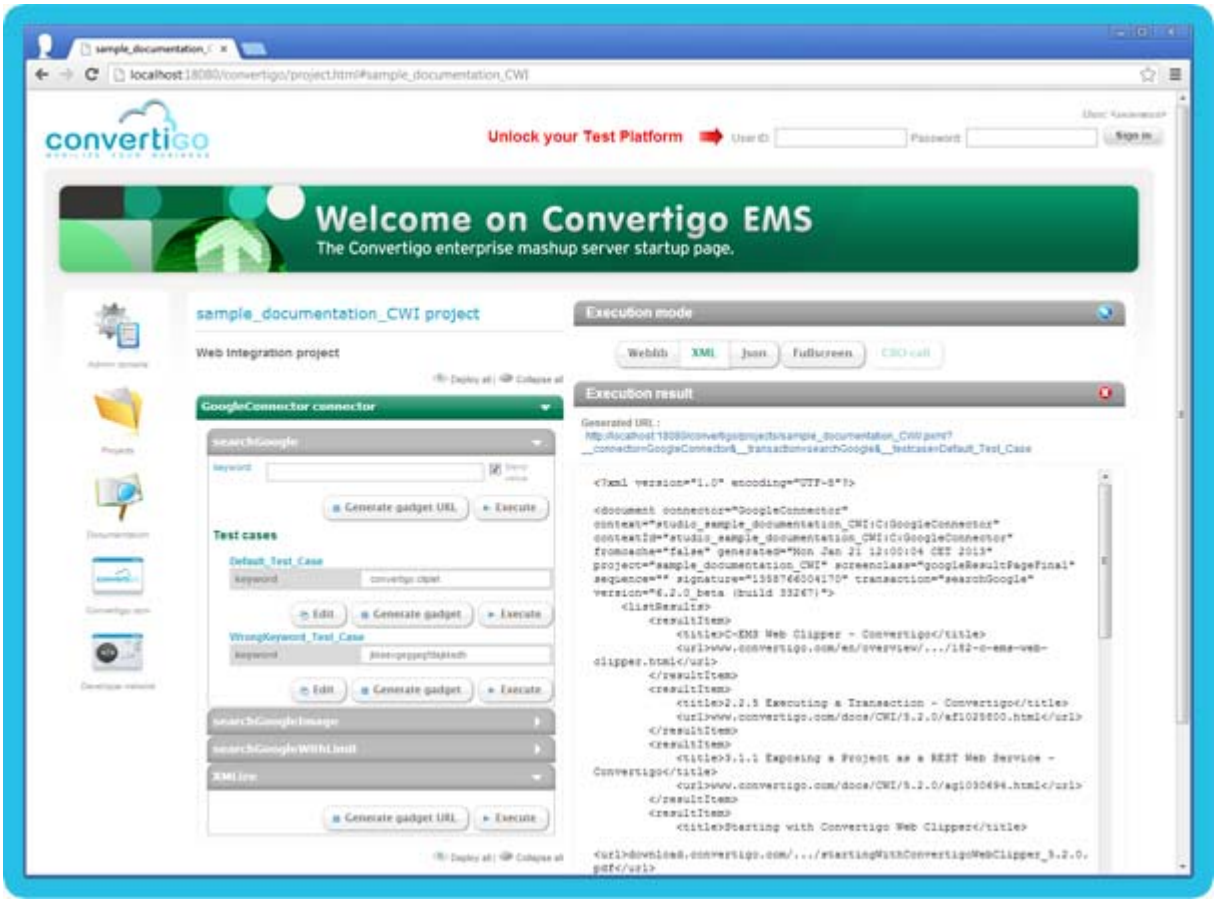


Figure 2 - 7: Style sheet - Transaction result with no stylesheet attached

In more details:

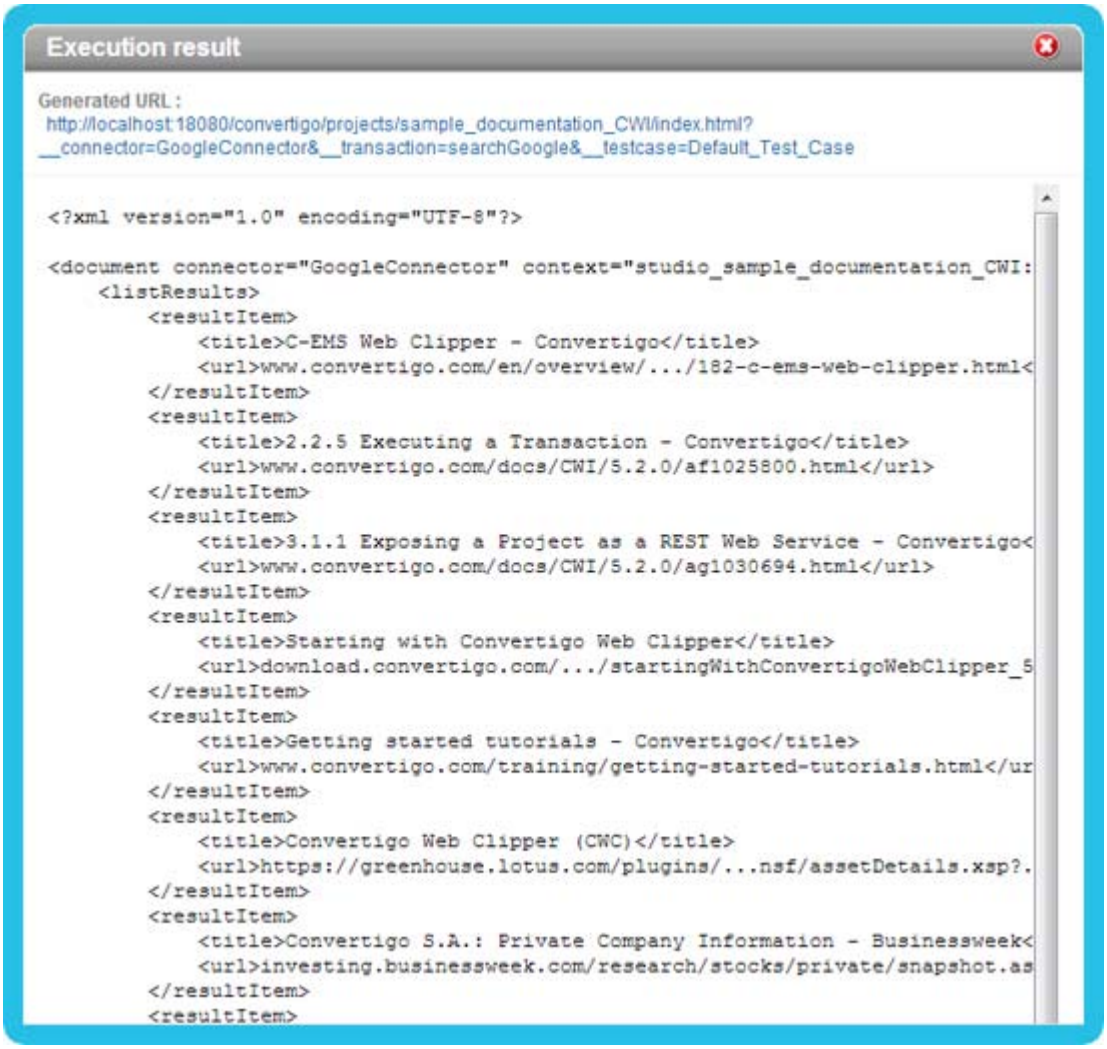


Figure 2 - 8: Style sheet - Transaction result with no stylesheet attached (zoom)

To manage the results display, a *Style sheet* object is created with the following parameters:

```
Style sheet [
  browser=*
  URL="results.xsl"
]
```

The *Style sheet* object is created in the **Sheets** folder of the transaction, and appears as follows in the **Projects** view:

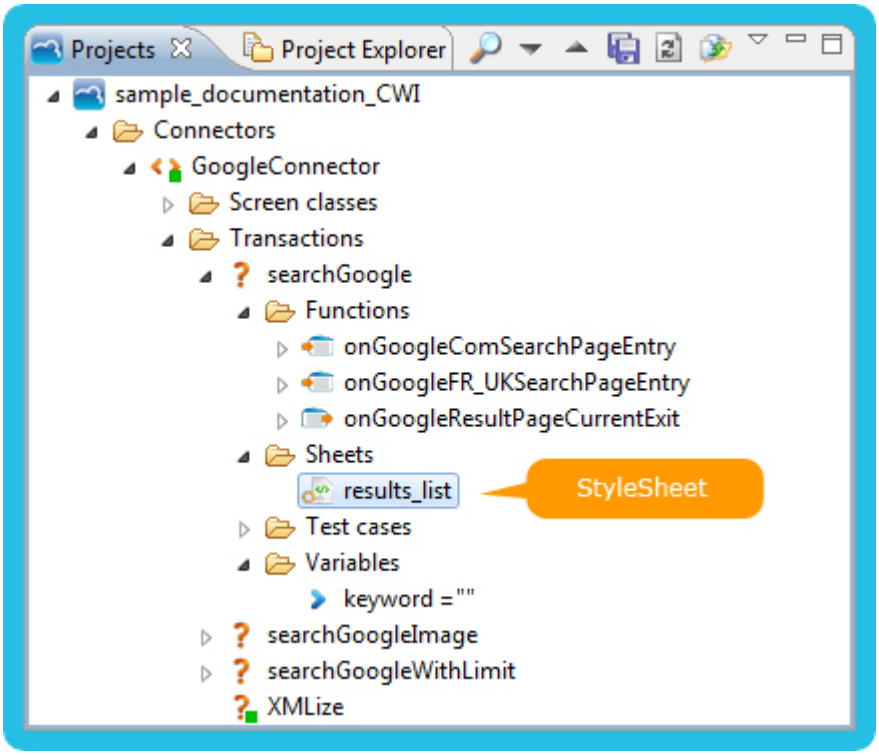


Figure 2 - 9: Style sheet - Style sheet object in Projects view

Its parameters are edited in the **Properties** view of the Convertigo Studio:

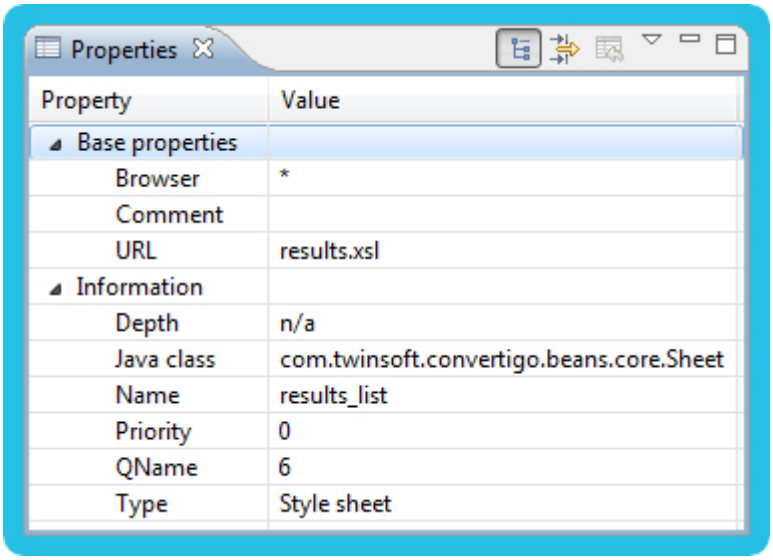


Figure 2 - 10: Style sheet - Configuration example

The **URL** property is relative to the project directory. It is set with no parent path, meaning that the style sheet file has to be stored at the root of the project directory.

To finish the configuration of the *Style sheet* displaying the transaction's output XML, two steps are performed:

- creating and editing the `results.xsl` file on which is pointing the *Style sheet* object,
- setting the transaction **Style sheet** property to use the defined *Style sheet* object.

Creating the XSL file

The `results.xsl` file is defined by the *Style sheet* object to be the style sheet performing the XSL transformation of the `searchGoogle` transaction XML output.

A `results.xsl` file is created in the project folder:

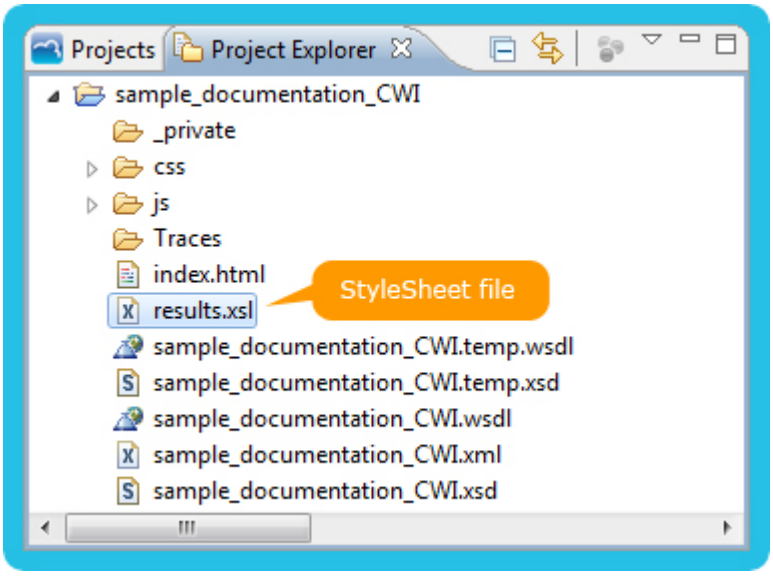


Figure 2 - 11: Style sheet - Creating results.xsl file on project root folder

It is then edited so that results are layed out in two columns headed by a title:

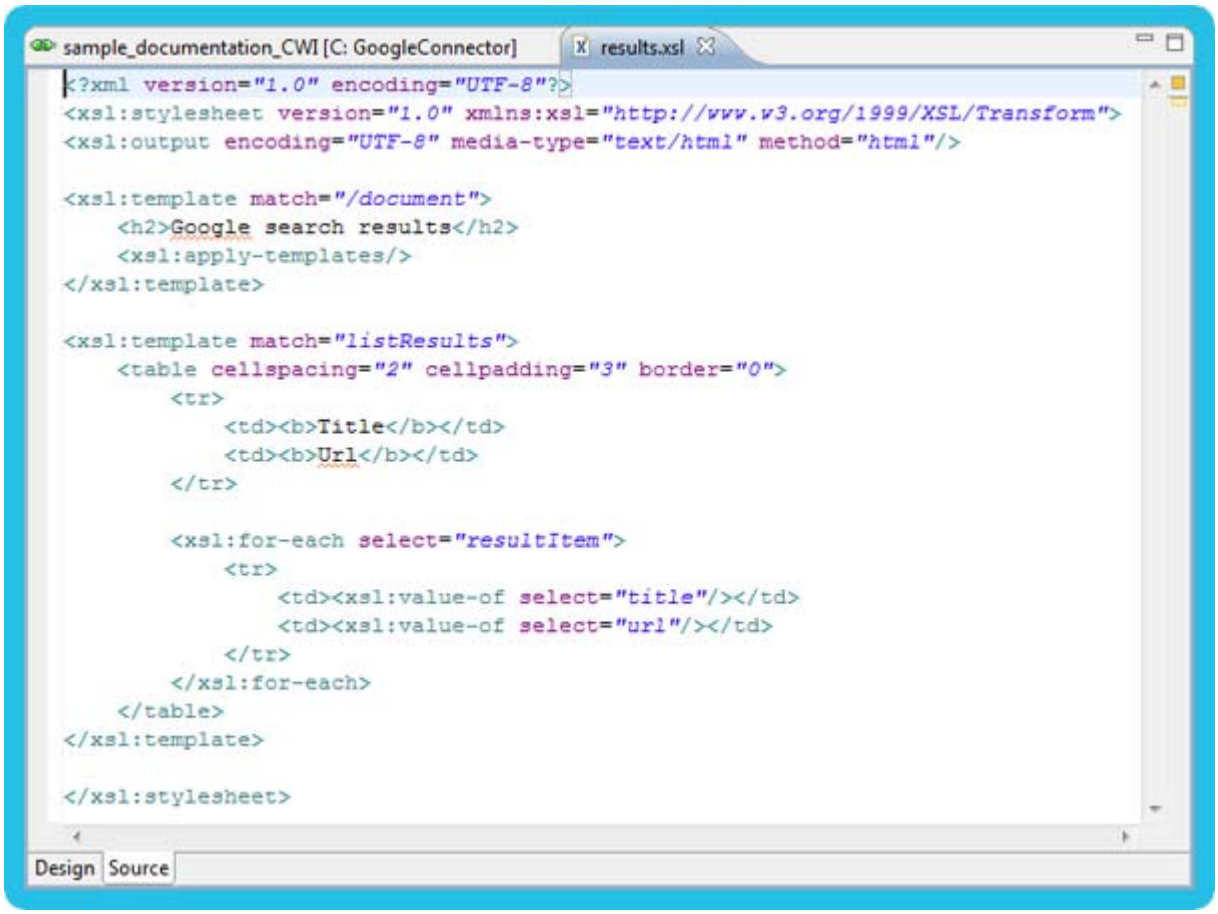


Figure 2 - 12: Style sheet - Editing results.xsl file

Setting the transaction property

For the transaction to use the *Style sheet* object defined in its *sheets* folder, its **Style sheet** property has to be set from `None` to `From transaction`:

```
Transaction [
  style sheet=From transaction
]
```

This parameter is edited in the **Properties** view of the Convertigo Studio:

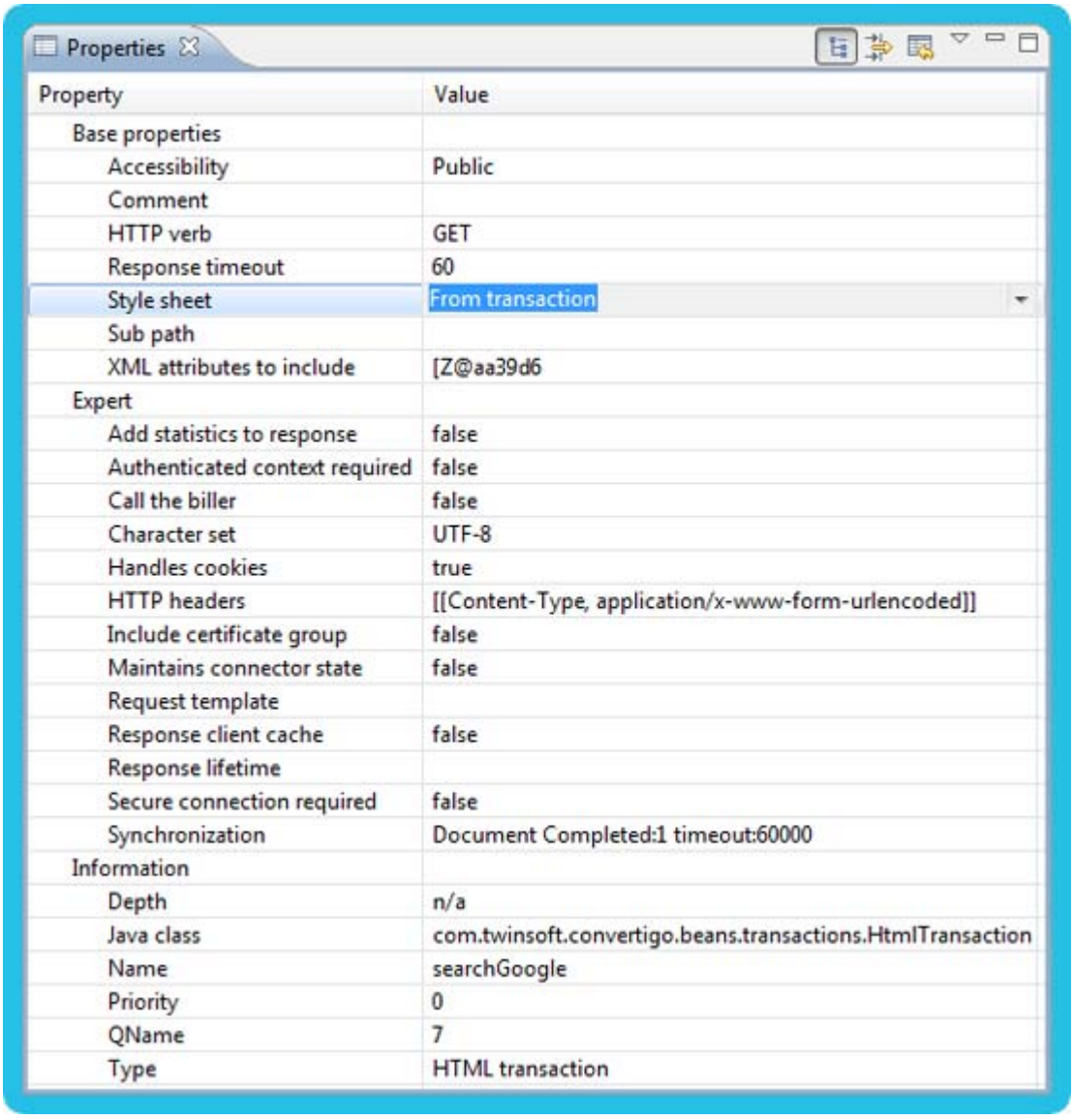


Figure 2 - 13: Style sheet - Transaction configuration example

Now all configuration is finished, when the transaction is executed again, results are displayed as required thanks to the XML transformation:

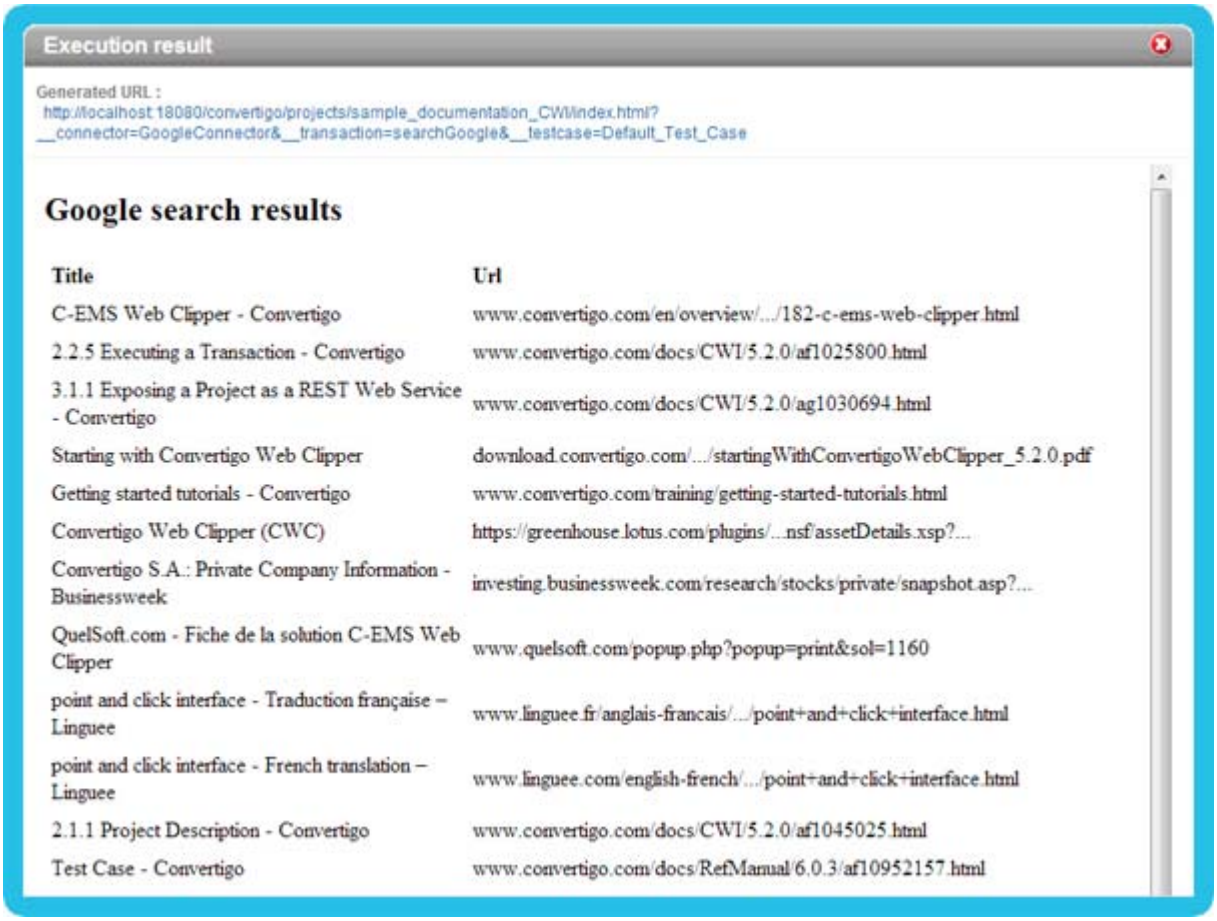


Figure 2 - 14: Style sheet - Transaction result displayed thanks to results.xsl stylesheet

POOL 

OBJECT DESCRIPTION

Creates a pool for a given Javelin connector.

A *Pool* is a set of preloaded Convertigo contexts on a defined connector.

When the Convertigo engine starts, the pool's contexts are loaded, executing a **starting transaction** defined in the properties.

Thanks to the execution of the **starting transaction**, the pool's contexts are led to a steady state.

Notes:

- The steady state can be defined as a particular screen class that has to be reached by the connector (*Javelin screen class*).
- Defining a pool is useful for performance optimization, in that it allows accessing to pre-established and advanced connections with the host.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Connection address	String	standard	Replaces the connection address (optional). If defined, this property overrides the connection address set as connector parameter for the loaded contexts.

Property	Type	Category	Description
Initial screen class	String	standard	Defines the steady state screen class. On connectors for which it applies (*Javelin connector*), this property allows to define the steady state screen class. This initial screen class has several implications: • it's the screen class that must be reached by the context thanks to the **starting transaction** execution, • the transactions that are executed thereafter on one of the pool's contexts must lead back to this screen class to let the context in steady state for a further use. If you want to realize non-atomic calls, i.e. call several successive transactions on the same context without restoring the context's steady state between calls, it is possible to lock the context by setting the `context.lockPooledContext` property to `true`. Note: be sure to reset the `context.lockPooledContext` property to `false` at the end of your non atomic calls, otherwise this locked context will remain unavailable.
Number of contexts	int	standard	Defines the size of the pool. This property defines the number of contexts to load for this pool.
Starting transaction	String	standard	Defines the transaction to be automatically executed when loading the Convertigo context. This transaction must lead the connector to a steady state (a particular screen class).
Starting transaction variables	XMLVector	standard	Defines the starting transaction variables. This property allows to define a list of variables that will be sent to the starting transaction executed on each context. For each variable, you have to describe three properties: • **Context number**: number between 1 and the size of the pool (defined in the the **Number of contexts** property). The variable will be sent with the associated value only for this context number. To define a variable for all loaded contexts, this property can take the following value: `*`. • **Parameter name**: name of the variable. • **Parameter value**:value of the variable. Note: A new variable can be added to the list using the blue keyboard icon. The variables defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.

EXAMPLES

The following is an example of *Pool* set on a Screen connector connecting to an IBM 5250 legacy application:

```
Pool [
  connection address=""
  initial screen class=ASMENU_mode
  number of contexts=3
```

```
   starting transaction=loginTransaction
   starting transaction variables=[[*, login, "user"],
   [*,password, "test"]]
]
```

The *Pool* object is created in the **Pools** folder of the screen connector, and appears as follows in the **Projects** view:

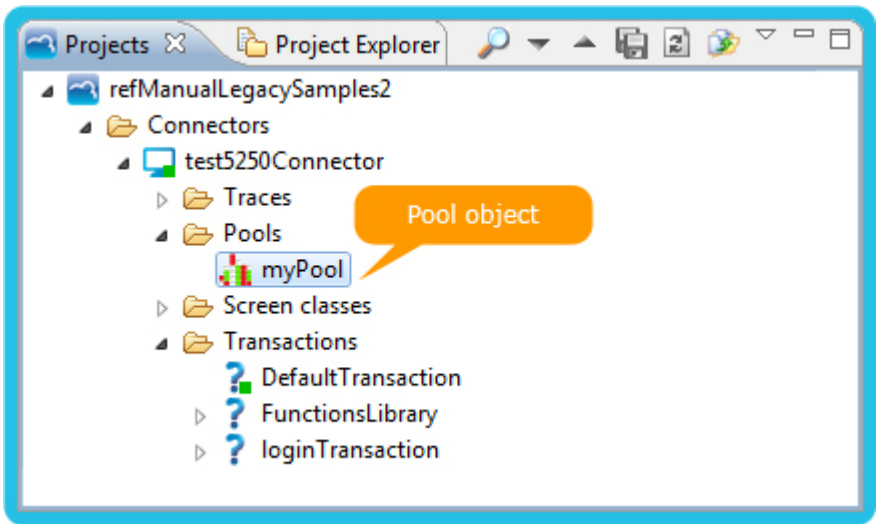


Figure 2 - 15: Pool - Pool object in Projects view

Its parameters are edited in the **Properties** view of the Convertigo Studio:

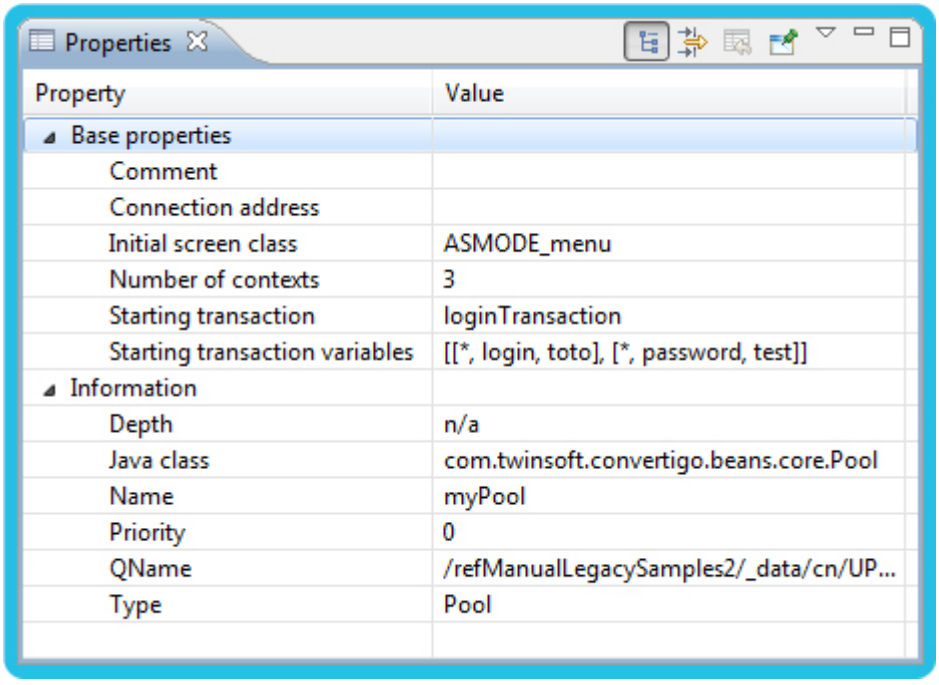


Figure 2 - 16: Pool - Configuration example

Starting transaction variables property is edited in the associated editor:

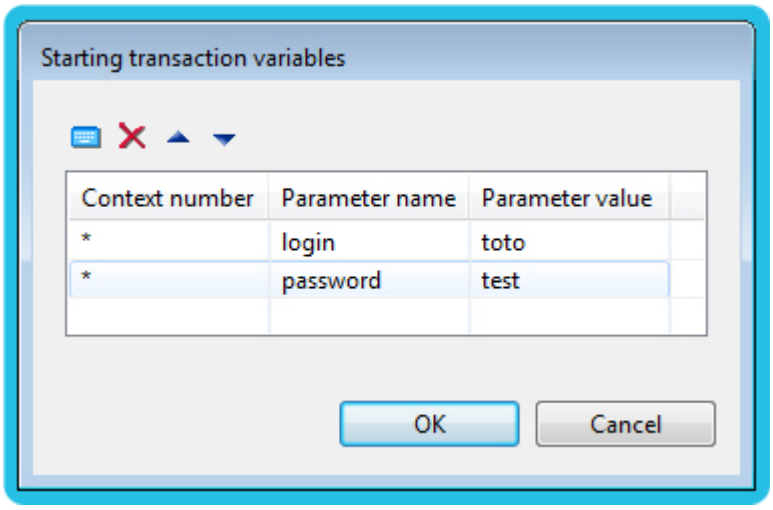


Figure 2 - 17: Pool - Starting transaction variables property edition

In **Convertigo Server Administration**, we can see the three contexts that are loaded by the declared *Pool*:

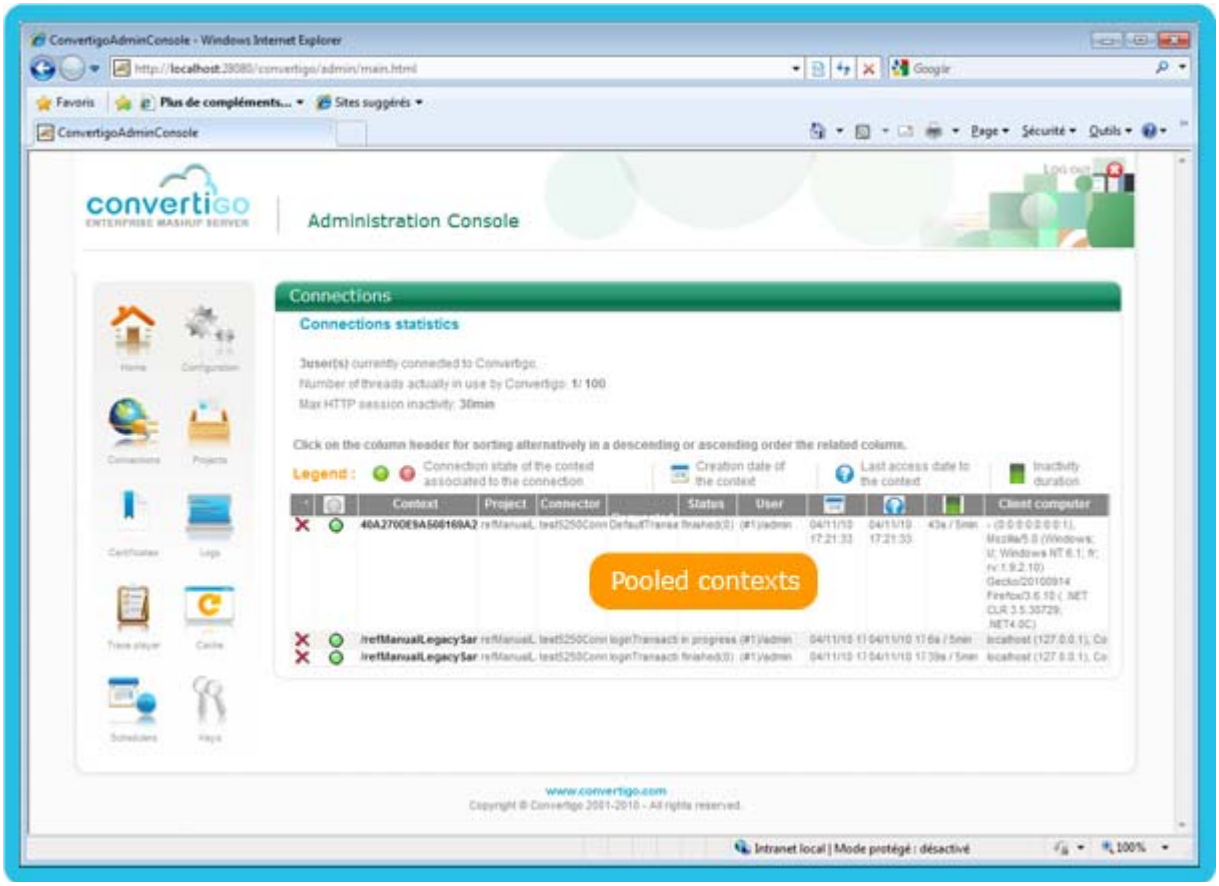


Figure 2 - 18: Pool - Connexions in Convertigo Server Administration

2.1.2 Variables

REQUESTABLE VARIABLES

REQUEST SINGLE-VALUED VARIABLE 

OBJECT DESCRIPTION

Defines a single-valued variable for a transaction/sequence.

A *Request single-valued variable* declares a variable which accepts a unique value to a transaction/sequence.

This variable is dedicated to the following transaction/sequence objects, except for HTTP-based transactions which use more specific variables:

- *Generic Sequence*,
- *Javelin transaction*,
- *SQL transaction*,
- *Site Clipper transaction*.

This variable object can define a default value, specified in the **Default value** property, that is used if no value is found for this variable.

At runtime, the variable value is calculated by Convertigo through the following steps:

- the value is received in the request to the transaction/sequence,
- if no value is received for this variable, the JavaScript value of the variable is chosen, if a variable of the same name exists in the JavaScript scope of current context,
- if no JavaScript value is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives a value, the default value is used,
- if no default value is specified, the variable is not defined and an Exception can be thrown when trying to access its value in the core of the transaction/sequence.

Note: In Convertigo Studio, when a *Request single-valued variable* is created in a transaction/sequence, it can be easily replaced by a *Request multi-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **MultiValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Cache key	boolean	expert	Defines whether the variable should be part of the cache key. If set to `true`, the variable and its value are added to the cache key which is used to determine whether the transaction's response (or sequence's response) should be pulled from the cache or not. A transaction's cached response (or sequence's cached response) is pulled from the cache when all cache key values are corresponding to a stored cache entry (may contain other data that variables, for example the certificate group defined by some transactions).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Customizable	boolean	expert	Defines whether the variable is customizable. If set to `true`, the variable is used as a customizable preference field in the widget generated from the parent transaction (or sequence) in Convertigo Mashup Composer or any other portal. Note: This property is used when applicable, i.e. when the widget is declared in a portal including customizable preference fields feature.
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.

Property	Type	Category	Description
Is a file upload	boolean	expert	Defines whether the variable is an uploaded file. When set to `true`, this property indicates that the transaction/sequence should receive an uploaded file in this variable. When received, the uploaded file is stored in a temporary folder and deleted at the end of the transaction/sequence. In the transaction/sequence execution context, the variable contains the path of the temporary file. Note: This property value is used only by the *Test Platform* to allow the developer testing the transaction/sequence. When receiving a multi-part request, Convertigo can set any variable as an uploaded file.
Schema type QName	XmlQName	expert	Assigned schema type qualified name
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.
WSDL exposed	boolean	expert	Defines whether the variable is exposed in web service. If set to `true`, variable definition is inserted in the project's WSDL as a method parameter. Note: This property value is ignored if the **Public method** property of the parent transaction (or sequence) is set to `false`, which means the method itself is not exposed in the web service.

EXAMPLES

Transaction or sequence objects may use variables useful for their execution process. To explain the best use of these variables and their overrides, the following examples are based on a Convertigo Mashup Sequencer project named `sample_refManual_variables`.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **Variables examples** in the **New Project** wizard.*

Let's consider we'd like to retrieve some user's details on a login process to return a welcoming message. Let's also assume that these details should be passed as input by the caller and are mandatory.

Example 1

A very simple *Generic Sequence*, named `userLoginWithDefault`, is created with a unique *jElement* step which writes a `<login>` XML element containing a message into the sequence's XML response.

To retrieve user's details from the caller to write them in the welcoming message, this sequence declares two variables:

- a *Request single-valued variable*, named `lastName`, with the following parameters:

```
Request single-valued variable [
   Default value="User"
]
```

- a *Request multi-valued variable*, named `firstNames`, with the following parameters:

```
Request multi-valued variable [
   Default value=["Anonymous"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

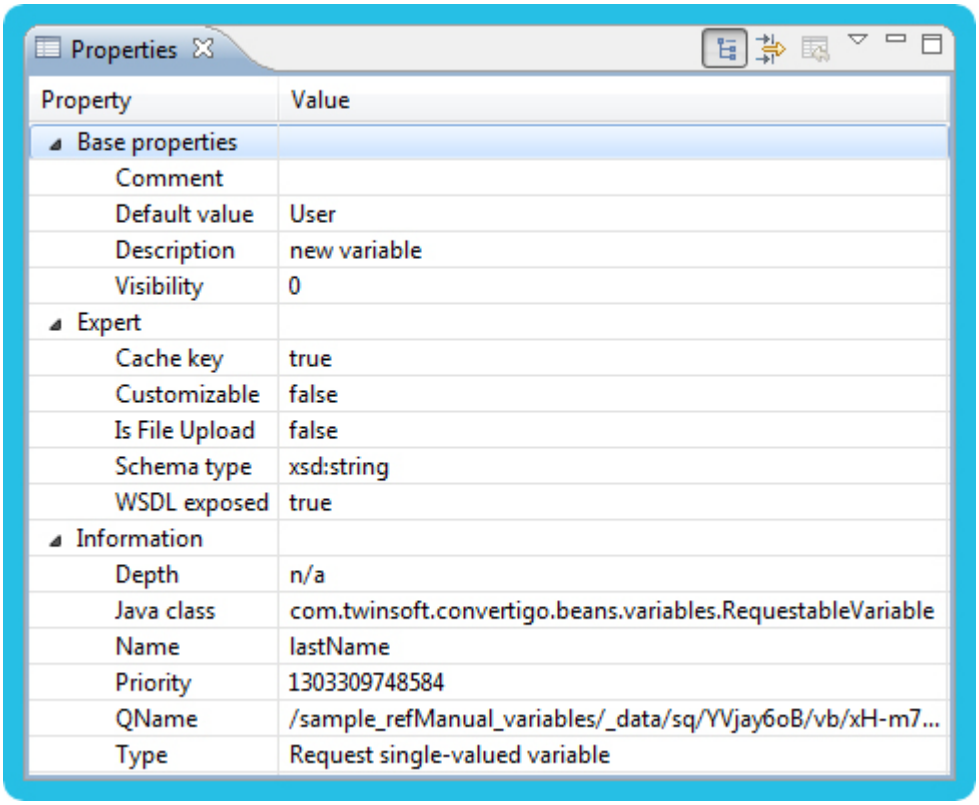


Figure 2 - 19: Request single-valued variable - Configuration example

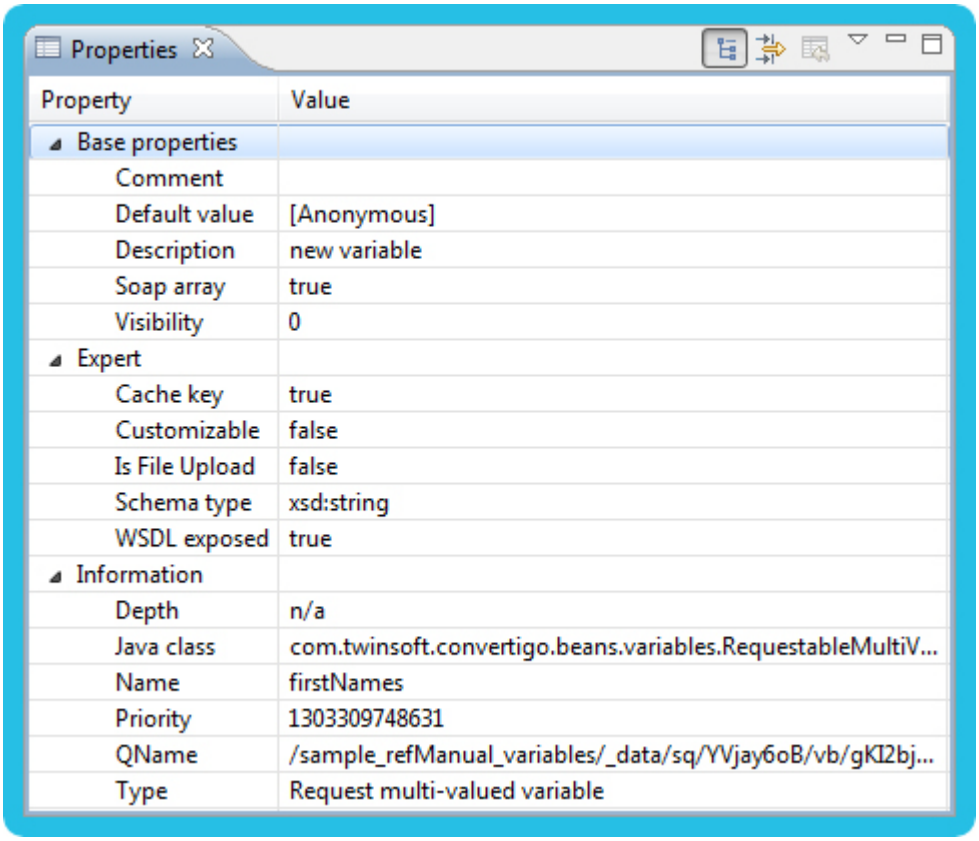


Figure 2 - 20: Request multi-valued variable - Configuration example

To meet the specifications, this sequence handles a default case where user's details are not provided. Thus, the **Default value** properties of both variables are set to fixed values (not `null`). For the *Request multi-valued variable*, it is edited in the **Array** editor:

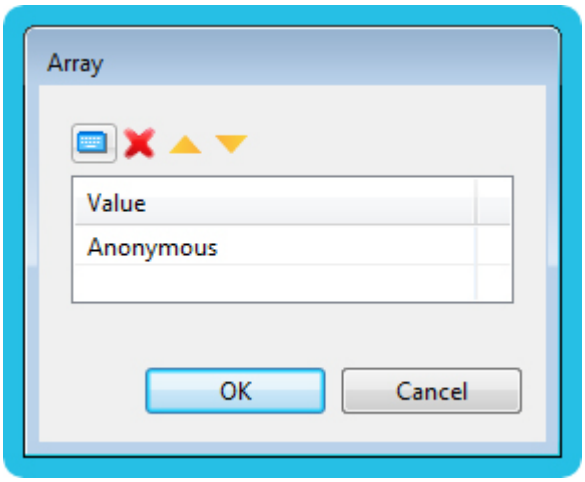


Figure 2 - 21: Request multi-valued variable - Default value property in Array editor

The sequence with its variables is created in the **Sequences** folder of the project and appears as follows in the **Projects** view:

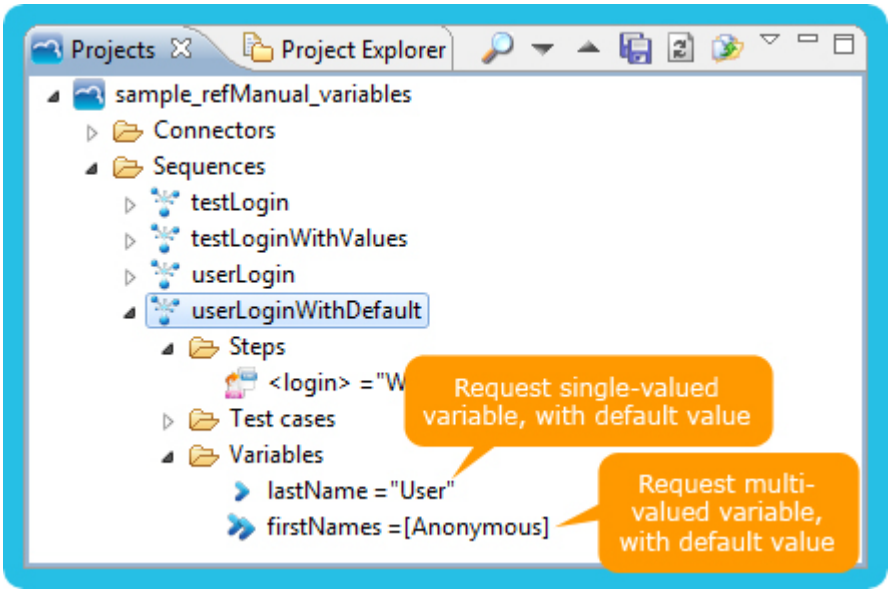


Figure 2 - 22: Request single-valued and multi-valued variables - Sequence and Request variables in Projects view

The JavaScript expression evaluated by the `<login>` message *jElement* step is:

```
"Welcome "+ firstNames[0] +" "+ lastName
```

This quite simple JavaScript expression leads, at runtime, to generate a message with the first firstname and the lastname of the user. For more information about *jElement* step, see "*jElement*" object documentation and examples.

At runtime, sequence variables (with their default values) are inserted into the JavaScript scope of the context. Thus, while executing the sequence in the Studio, the above expression don't throw any Exception and the following XML is returned (see **XML** tab of the sequence editor):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document connector=""
  context="studio_TestVariables:userLoginWithDefault"
  contextId="studio_TestVariables:userLoginWithDefault"
  fromcache="false" generated="Wed Apr 20 18:21:12 CEST 2011"
  project="TestVariables" sequence="userLoginWithDefault"
  signature="1303316472447" transaction="" version="5.5.0_alpha">
    <login>Welcome Anonymous User</login>
</document>
```

The sequence is fully implemented and ready to be tested. Let's switch to a web browser displaying the test platform of the project.

The sequence is presented with its variables already prefilled by their default values:

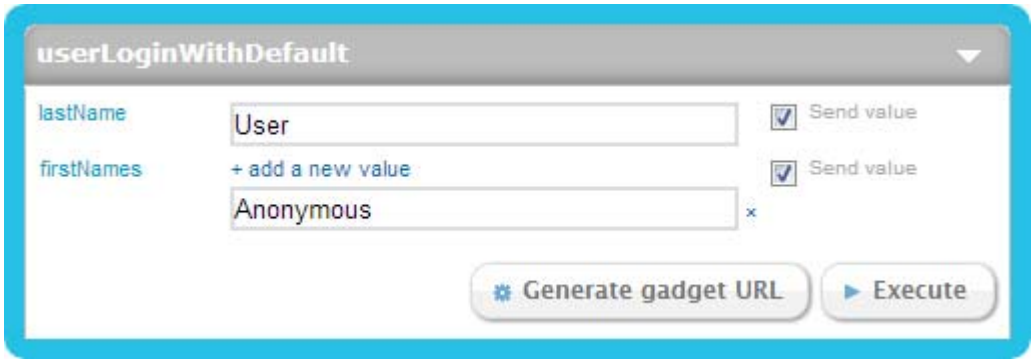


Figure 2 - 23: Request single-valued and multi-valued variables - Sequence and variables in test platform

Several modifications can be performed on the test platform to test the sequence execution. We can modify the `lastName` variable value, let's replace the "`User`" value to "`Person`", and we can uncheck the `firstNames` variable checkbox, for this variable not to be sent to the sequence call.

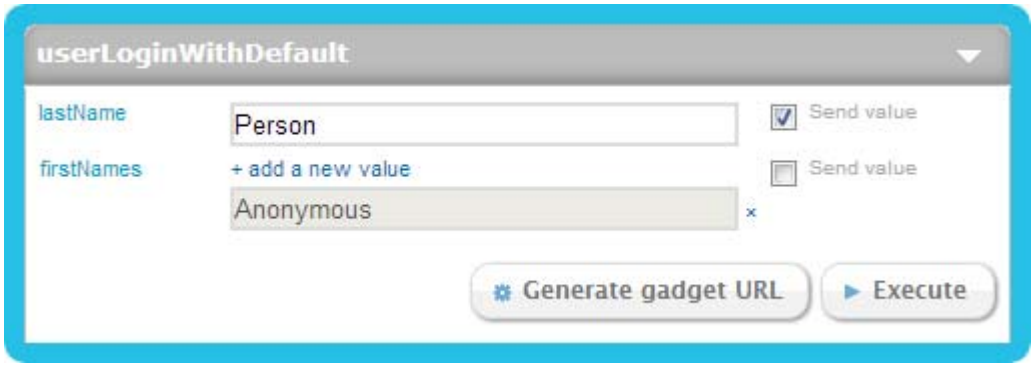


Figure 2 - 24: Request single-valued and multi-valued variables - Sequence and variables in test platform after testing modifications

Test the sequence with updated variables by clicking on the **Execute** button. The sequence is executed and the following XML is returned as result in the **Execution result** panel:

Figure 2 - 25: Request single-valued and multi-valued variables - Sequence execution result in test platform

The `<login>` message is different from the Studio execution, modified by the sequence execution to: "`Welcome Anonymous Person`".

WHAT HAPPENED?

The `lastName` variable was sent to the sequence as a request parameter, thus its default value was overridden with the received value.

The `firstNames` variable was not sent to the sequence, thus its default value ("`Anonymous`") was used.

If none of the two variables were sent, the message would remain unchanged.

Example 2

Unlike the previous one, this example does not handle a default case. Here, we force the user to give the required details by sending back an error message if necessary.

A *Generic Sequence*, named `userLogin`, is created with a *jIfThenElse* step which checks for the `lastName` and `firstNames` variables values validity. Depending on the result, a `<login>` or an `<error>` XML element containing a message is written into the sequence's XML response.

To retrieve user's details from the caller to write them in one of the messages, this sequence declares two variables:

- a *Request single-valued variable*, named `lastName`, with the following parameters:

```
Request single-valued variable [
   Default value=null
]
```

- a *Request multi-valued variable*, named `firstNames`, with the following parameters:

```
Request multi-valued variable [
   Default value=[null]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

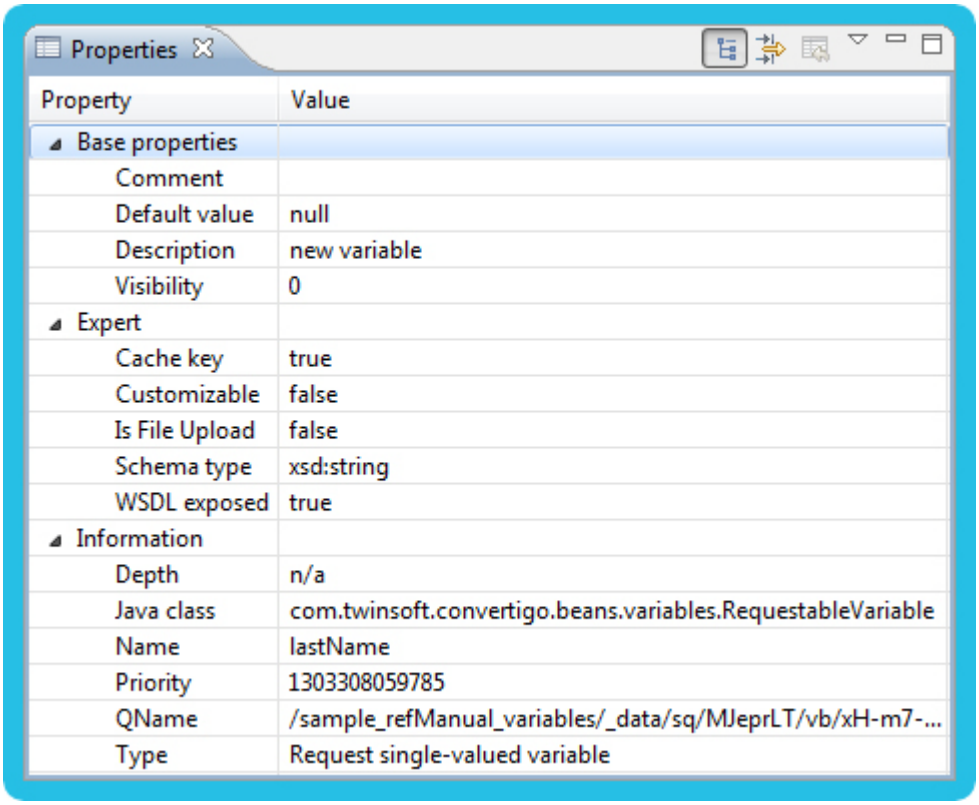


Figure 2 - 26: Request single-valued variable - Configuration example

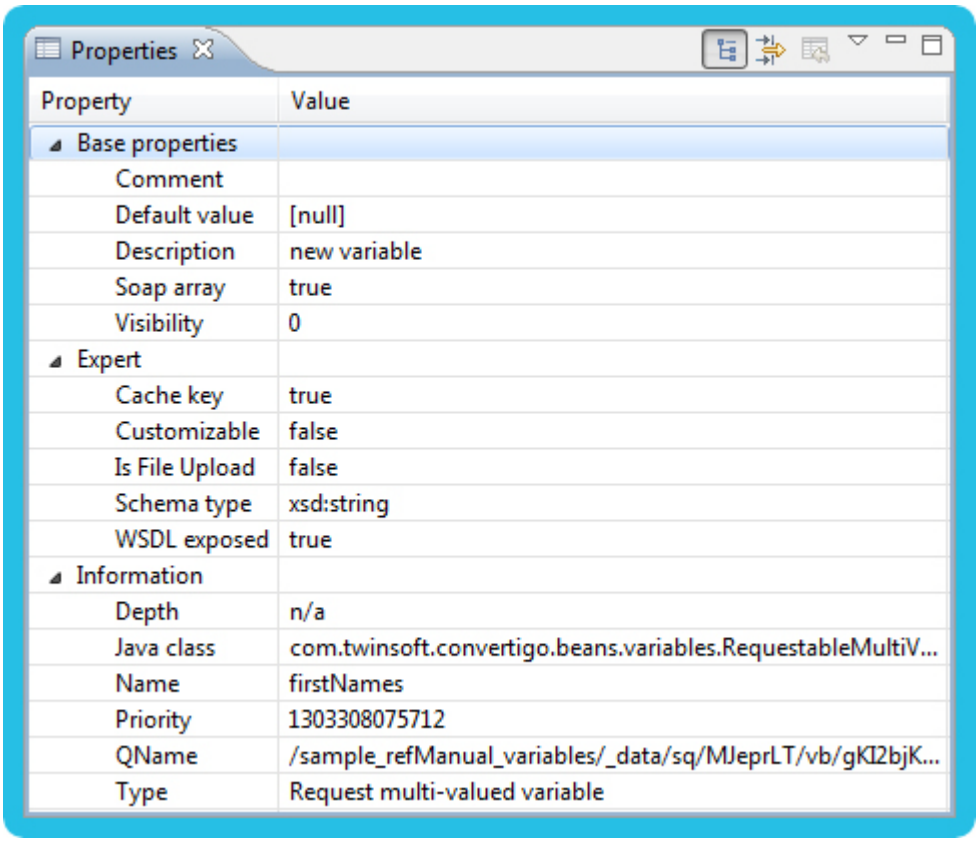


Figure 2 - 27: Request multi-valued variable - Configuration example

The **Default value** properties are left to `null` values, meaning variables are just declared.

This sequence with its variables is created in the **Sequences** folder of the project and appears as follows in the **Projects** view:

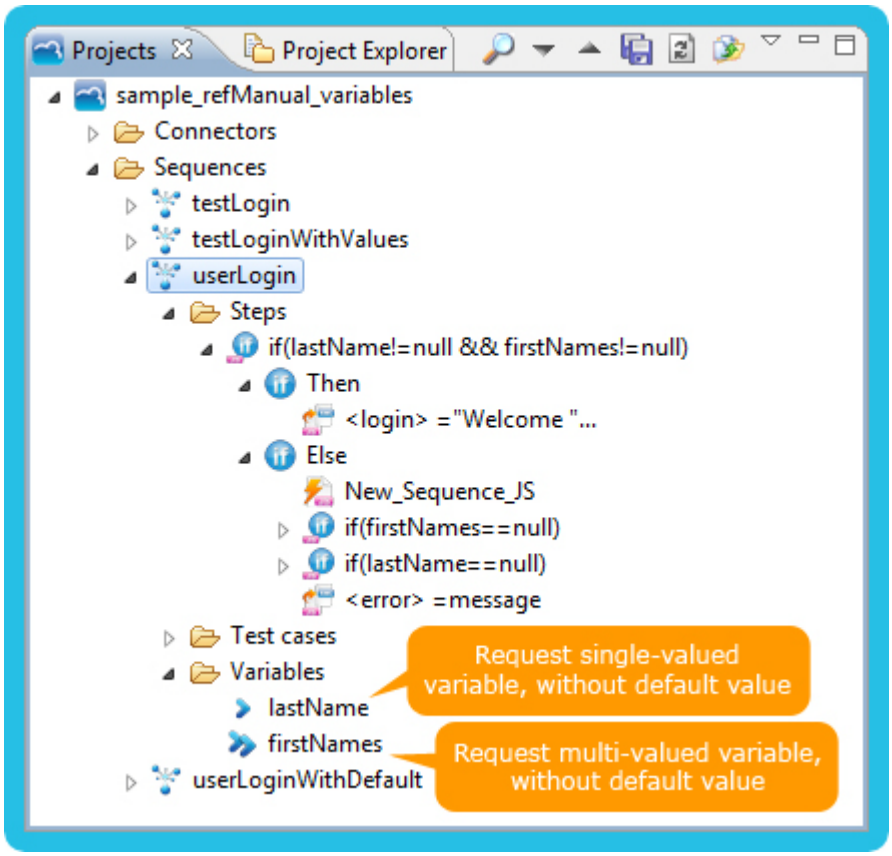


Figure 2 - 28: Request single-valued and multi-valued variables - Sequence and Request variables in Projects view

The JavaScript expression evaluated by the *jIfThenElse* step is:

```
if(lastName!=null && firstName!=null)
```

At runtime, variables (with their default values) are inserted into the JavaScript scope of the context. Thus, while executing the sequence in the Studio, the above expression is not verified and the following XML is returned (see **XML** tab of the sequence editor):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document connector="" context="studio_TestVariables:userLogin"
  contextId="studio_TestVariables:userLogin" fromcache="false"
  generated="Thu Apr 21 10:56:25 CEST 2011" project="TestVariables"
  sequence="userLogin" signature="1303376185017" transaction=""
  version="5.5.0_alpha">
    <error>firstname is required, lastname is required</error>
</document>
```

The sequence is fully implemented and ready to be tested. Let's switch to a web browser displaying the test platform of the project.

The sequence is presented with its variables already prefilled by empty default values:

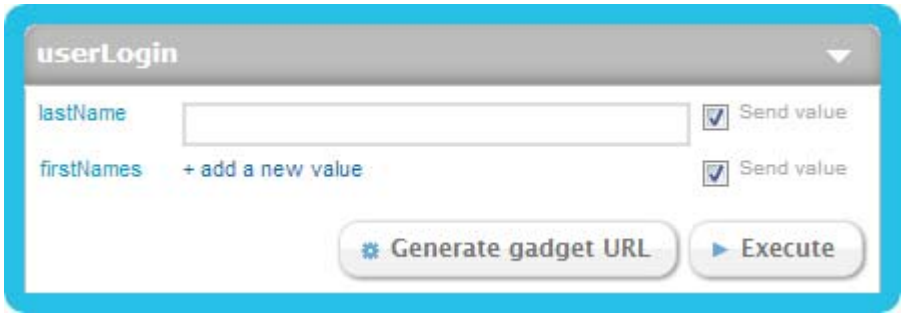


Figure 2 - 29: Request single-valued and multi-valued variables - Sequence and variables in test platform

Several modifications can be performed on the test platform to test the sequence execution. We can modify the `lastName` variable value, let's replace the empty string by "`User`" value, and we can uncheck the `firstNames` variable checkbox, for this variable not to be sent to the sequence call.

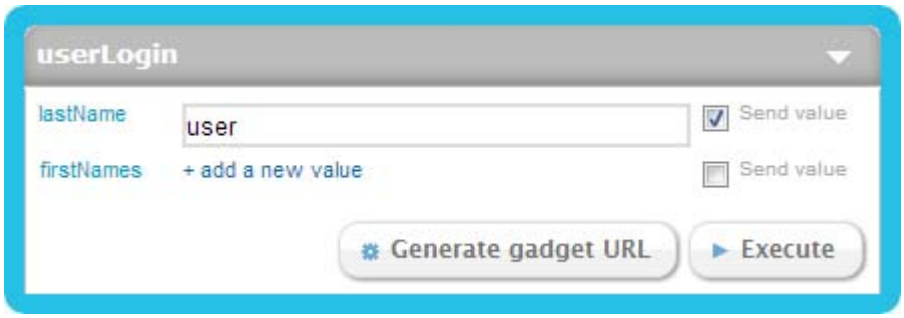


Figure 2 - 30: Request single-valued and multi-valued variables - Sequence and variables in test platform after testing modifications

Test the sequence with updated variables by clicking on the **Execute** button. The sequence is executed and the following XML is returned as result in the **Execution result** panel:

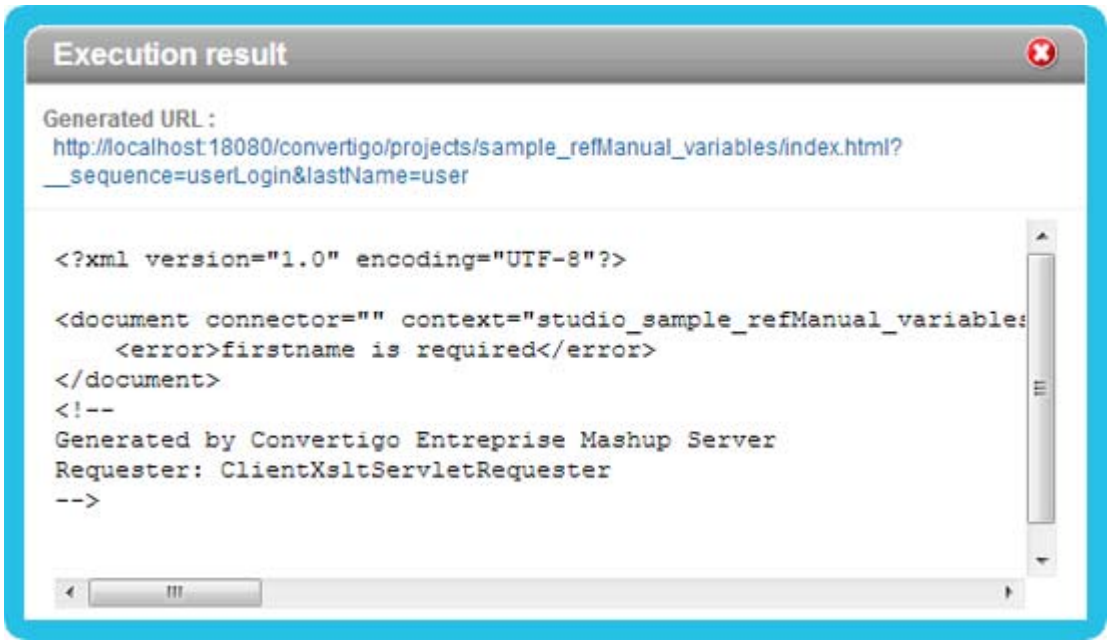


Figure 2 - 31: Request single-valued and multi-valued variables - Sequence execution result in test platform

An `<error>` message is returned, different from the Studio execution, modified by the sequence: "`firstname is required`".

WHAT HAPPENED?

The `lastName` variable was sent to the sequence as a request parameter, thus its default value was overridden with the received value.

The `firstNames` variable was not sent to the sequence, thus its default value (`null`) was used, making the test step fail.

For a second test, we can modify both variables values, checking the `firstNames` variable checkbox for this variable to be sent to the sequence call and replacing its empty string by "`Name`" value.

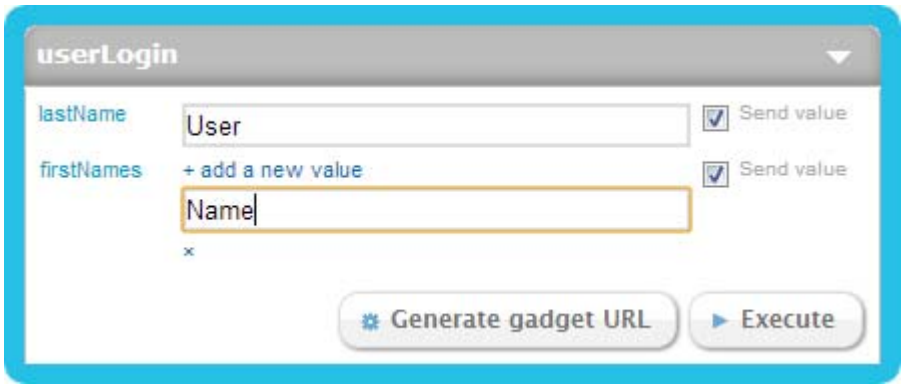


Figure 2 - 32: Request single-valued and multi-valued variables - Sequence and variables in test platform after testing modifications

Test the sequence with updated variables by clicking on the **Execute** button. The sequence is executed and the following XML is returned as result in the **Execution result** panel:

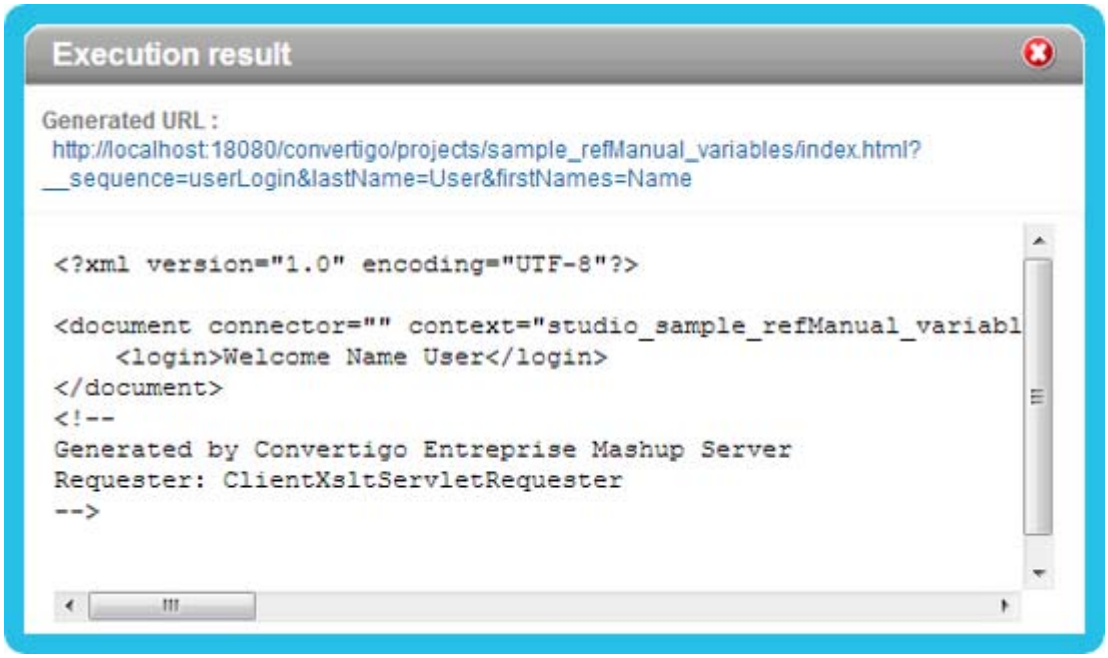


Figure 2 - 33: Request single-valued and multi-valued variables - Sequence execution result in test platform

A `<login>` message is returned, different from the previous execution, modified by the sequence: "`Welcome Name User`".

WHAT HAPPENED?

Both of the variables are sent to the sequence as request parameters, thus their default values are overridden with the received values, making the test succeed.

CONCLUSION

Using prefilled default value for a variable rather than the `null` value depends on the transaction or sequence purpose. We suggest the second solution as it is more constraining and will better help Convertigo developers develop their projects. Moreover, *Test Cases* can be used to test any particular case of transaction or sequence variables values.

REQUEST MULTI-VALUED VARIABLE 

OBJECT DESCRIPTION

Defines a multi-valued variable for a transaction/sequence.

A *Request multi-valued variable* declares a variable which accepts one or more values to a transaction/sequence.

This variable is dedicated to the following transaction/sequence objects, except for HTTP-based transactions which use more specific variables:

- *Generic Sequence*,
- *Javelin transaction*,
- *SQL transaction*,
- *Site Clipper transaction*.

This variable object can define a default list of value(s), specified in the **Default value** property, that is used if no value is is found for this variable.

At runtime, the variable values are calculated by Convertigo through the following steps:

- the values are received in the request to the transaction/sequence,
- if no value is received for this variable, the JavaScript value of the variable is chosen, if a variable of the same name exists in the JavaScript scope of current context (this JavaScript variable should be an array of values),
- if no JavaScript value is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives values, the default list of values is used,
- if no default value is specified, the variable is not defined and an Exception can be thrown when trying to access its values in the core of the transaction/sequence.

Note: In Convertigo Studio, when a *Request multi-valued variable* is created in a transaction/sequence, it can be easily replaced by a *Request single-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **SingleValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Cache key	boolean	expert	Defines whether the variable should be part of the cache key. If set to `true`, the variable and its value are added to the cache key which is used to determine whether the transaction's response (or sequence's response) should be pulled from the cache or not. A transaction's cached response (or sequence's cached response) is pulled from the cache when all cache key values are corresponding to a stored cache entry (may contain other data that variables, for example the certificate group defined by some transactions).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Customizable	boolean	expert	Defines whether the variable is customizable. If set to `true`, the variable is used as a customizable preference field in the widget generated from the parent transaction (or sequence) in Convertigo Mashup Composer or any other portal. Note: This property is used when applicable, i.e. when the widget is declared in a portal including customizable preference fields feature.
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.

Property	Type	Category	Description
Is a file upload	boolean	expert	Defines whether the variable is an uploaded file. When set to `true`, this property indicates that the transaction/sequence should receive an uploaded file in this variable. When received, the uploaded file is stored in a temporary folder and deleted at the end of the transaction/sequence. In the transaction/sequence execution context, the variable contains the path of the temporary file. Note: This property value is used only by the *Test Platform* to allow the developer testing the transaction/sequence. When receiving a multi-part request, Convertigo can set any variable as an uploaded file.
Schema type QName	XmlQName	expert	Assigned schema type qualified name
Soap array	boolean	standard	Defines if the multi-valued variable should be seen as a Soap Array of a occurrence of variables. In the case of transaction or sequence defined as a public SOAP method, this property allows to specify of the current multi-valued variable has to be seen in SOAP envelope as a Soap Array with multiple values inside it or as an occurrence of identical variables.
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.
WSDL exposed	boolean	expert	Defines whether the variable is exposed in web service. If set to `true`, variable definition is inserted in the project's WSDL as a method parameter. Note: This property value is ignored if the **Public method** property of the parent transaction (or sequence) is set to `false`, which means the method itself is not exposed in the web service.

EXAMPLES

Transaction or sequence objects may use variables useful for their execution process. To explain the best use of these variables and their overrides, the following examples are based

on a Convertigo Mashup Sequencer project named `sample_refManual_variables`.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

Let's consider we'd like to retrieve some user's details on a login process to return a welcoming message. Let's also assume that these details should be passed as input by the caller and are mandatory.

Example 1

A very simple *Generic Sequence*, named `userLoginWithDefault`, is created with a unique *jElement* step which writes a `<login>` XML element containing a message into the sequence's XML response.

To retrieve user's details from the caller to write them in the welcoming message, this sequence declares two variables:

- a *Request single-valued variable*, named `lastName`, with the following parameters:

```
Request single-valued variable [
   Default value="User"
]
```

- a *Request multi-valued variable*, named `firstNames`, with the following parameters:

```
Request multi-valued variable [
   Default value=["Anonymous"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

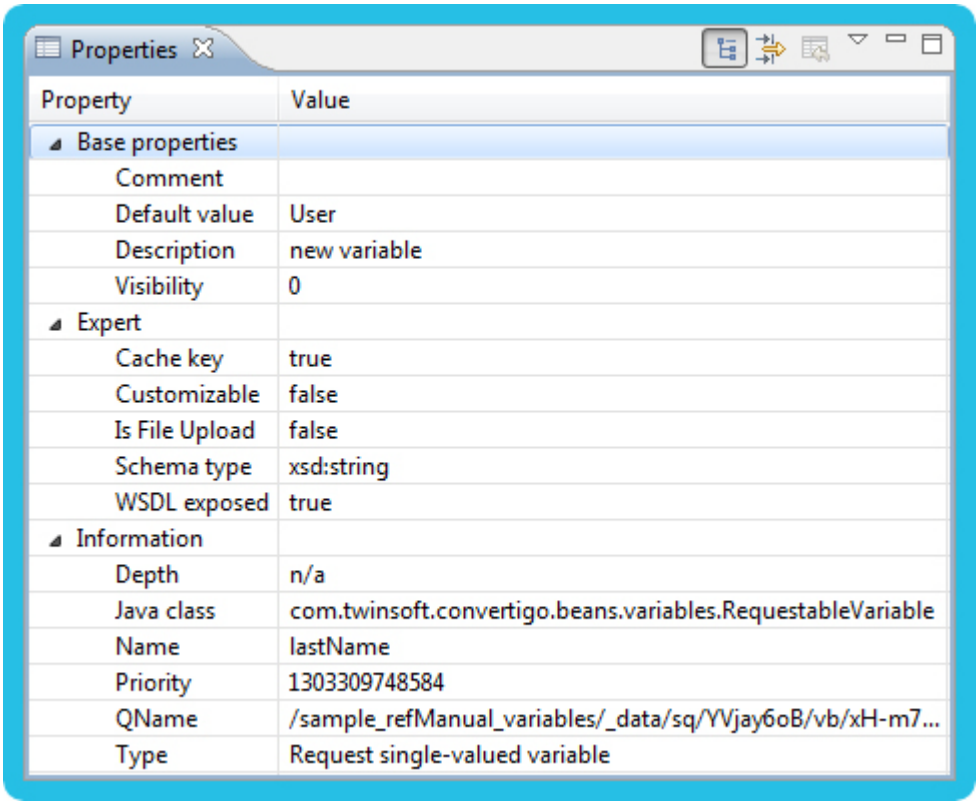


Figure 2 - 34: Request single-valued variable - Configuration example

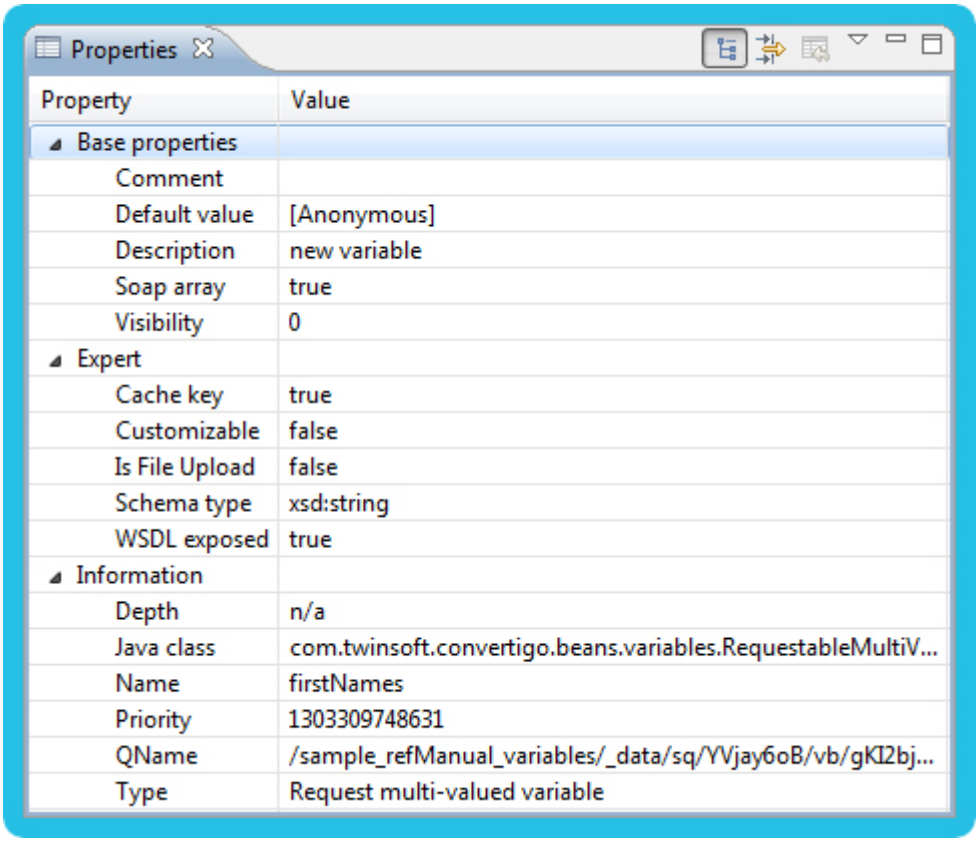


Figure 2 - 35: Request multi-valued variable - Configuration example

To meet the specifications, this sequence handles a default case where user's details are not provided. Thus, the **Default value** properties of both variables are set to fixed values (not

`null`). For the *Request multi-valued variable*, it is edited in the **Array** editor:

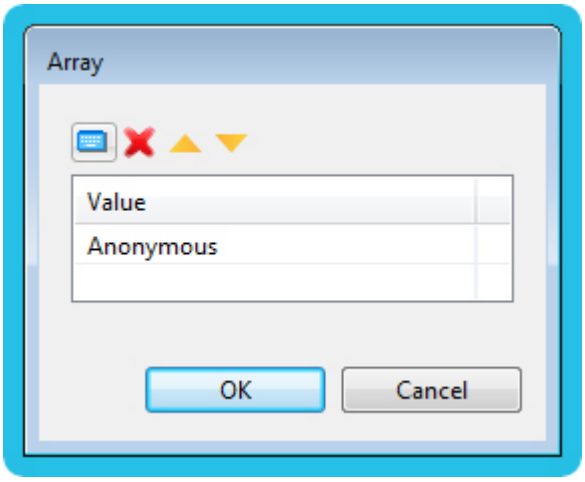


Figure 2 - 36: Request multi-valued variable - Default value property in Array editor

The sequence with its variables is created in the **Sequences** folder of the project and appears as follows in the **Projects** view:

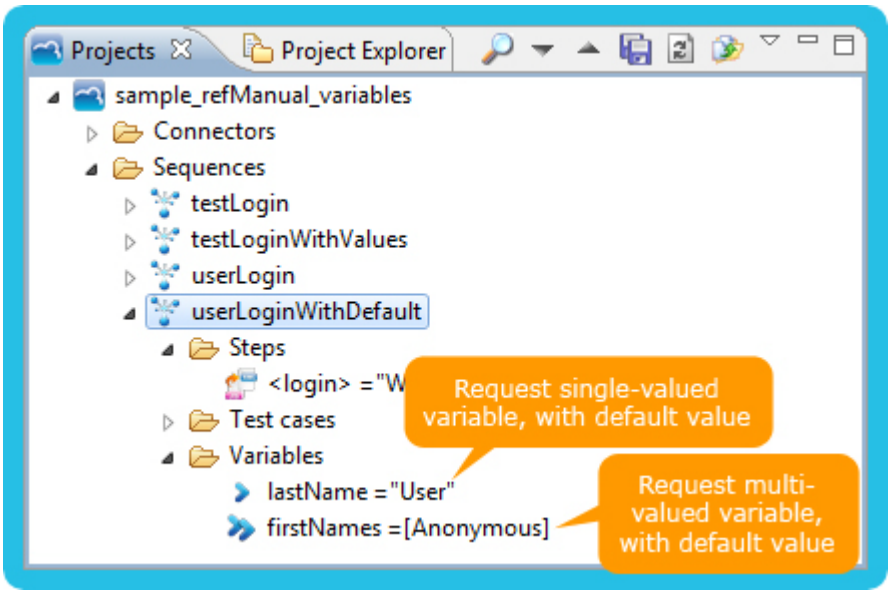


Figure 2 - 37: Request single-valued and multi-valued variables - Sequence and Request variables in Projects view

The JavaScript expression evaluated by the `<login>` message *jElement* step is:

```
"Welcome "+ firstNames[0] +" "+ lastName
```

This quite simple JavaScript expression leads, at runtime, to generate a message with the first firstname and the lastname of the user. For more information about *jElement* step, see "*jElement*" object documentation and examples.

At runtime, sequence variables (with their default values) are inserted into the JavaScript scope of the context. Thus, while executing the sequence in the Studio, the above expression don't throw any Exception and the following XML is returned (see **XML** tab of the sequence editor):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document connector=""
```

```
   context="studio_TestVariables:userLoginWithDefault"
   contextId="studio_TestVariables:userLoginWithDefault"
   fromcache="false" generated="Wed Apr 20 18:21:12 CEST 2011"
   project="TestVariables" sequence="userLoginWithDefault"
   signature="1303316472447" transaction="" version="5.5.0_alpha">
     <login>Welcome Anonymous User</login>
</document>
```

The sequence is fully implemented and ready to be tested. Let's switch to a web browser displaying the test platform of the project.

The sequence is presented with its variables already prefilled by their default values:

Figure 2 - 38: Request single-valued and multi-valued variables - Sequence and variables in test platform

Several modifications can be performed on the test platform to test the sequence execution. We can modify the `lastName` variable value, let's replace the "`User`" value to "`Person`", and we can uncheck the `firstNames` variable checkbox, for this variable not to be sent to the sequence call.

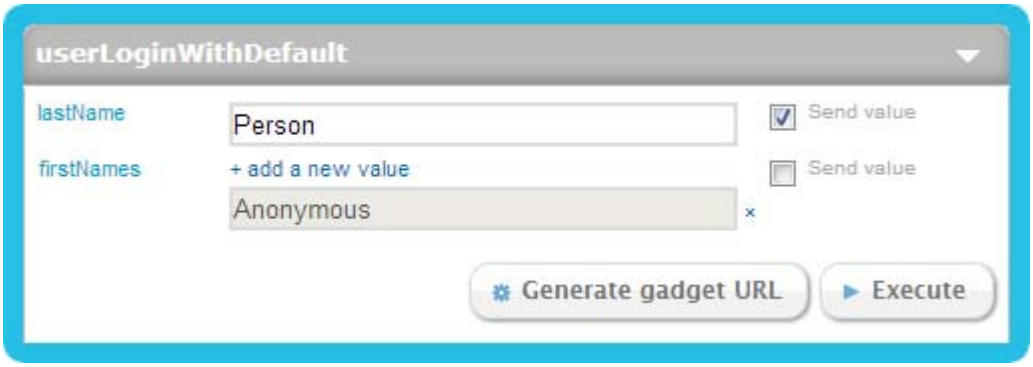


Figure 2 - 39: Request single-valued and multi-valued variables - Sequence and variables in test platform after testing modifications

Test the sequence with updated variables by clicking on the **Execute** button. The sequence is executed and the following XML is returned as result in the **Execution result** panel:

Figure 2 - 40: Request single-valued and multi-valued variables - Sequence execution result in test platform

The `<login>` message is different from the Studio execution, modified by the sequence execution to: "`Welcome Anonymous Person`".

WHAT HAPPENED?

The `lastName` variable was sent to the sequence as a request parameter, thus its default value was overridden with the received value.

The `firstNames` variable was not sent to the sequence, thus its default value ("`Anonymous`") was used.

If none of the two variables were sent, the message would remain unchanged.

Example 2

Unlike the previous one, this example does not handle a default case. Here, we force the user to give the required details by sending back an error message if necessary.

A *Generic Sequence*, named `userLogin`, is created with a *jIfThenElse* step which checks for the `lastName` and `firstNames` variables values validity. Depending on the result, a `<login>` or an `<error>` XML element containing a message is written into the sequence's XML response.

To retrieve user's details from the caller to write them in one of the messages, this sequence declares two variables:

- a *Request single-valued variable*, named `lastName`, with the following parameters:

```
Request single-valued variable [
   Default value=null
]
```

- a *Request multi-valued variable*, named `firstNames`, with the following parameters:

```
Request multi-valued variable [
   Default value=[null]
```

]

These parameters are edited in the **Properties** view of the Convertigo Studio:

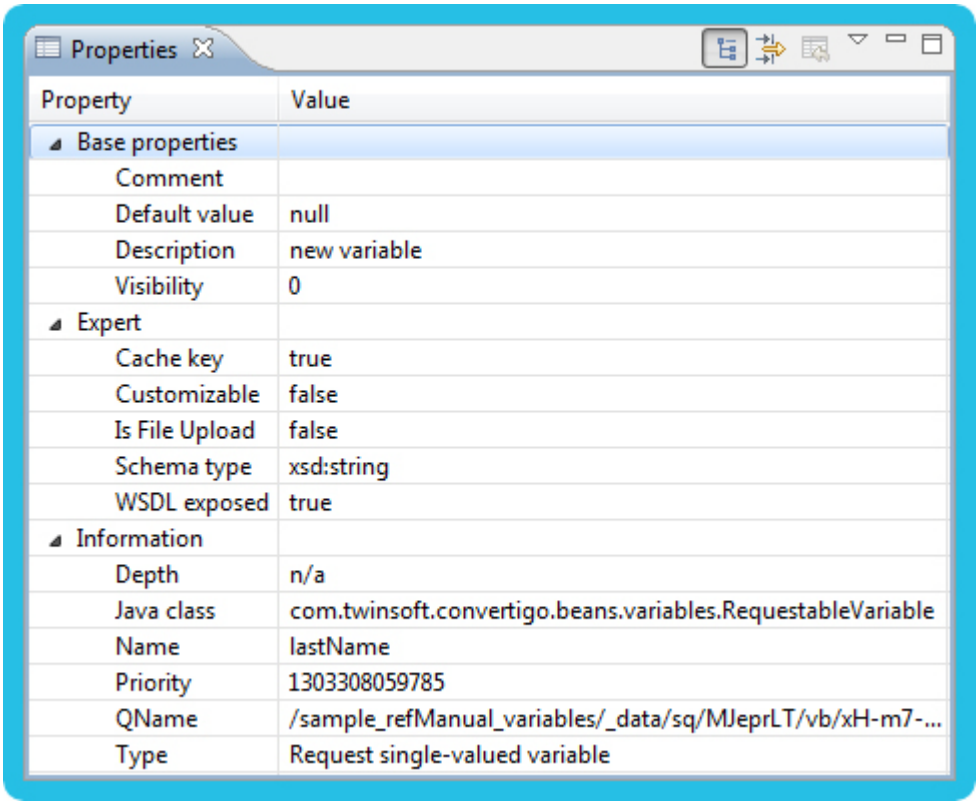


Figure 2 - 41: Request single-valued variable - Configuration example

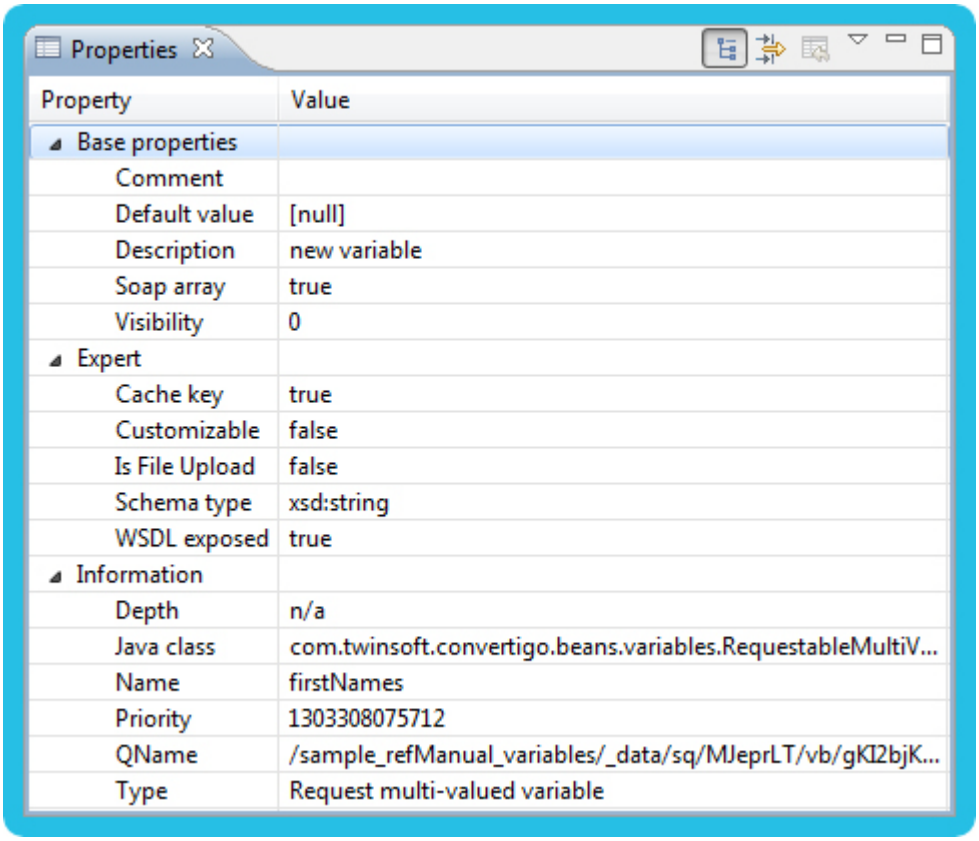


Figure 2 - 42: Request multi-valued variable - Configuration example

The **Default value** properties are left to `null` values, meaning variables are just declared.

This sequence with its variables is created in the **Sequences** folder of the project and appears as follows in the **Projects** view:

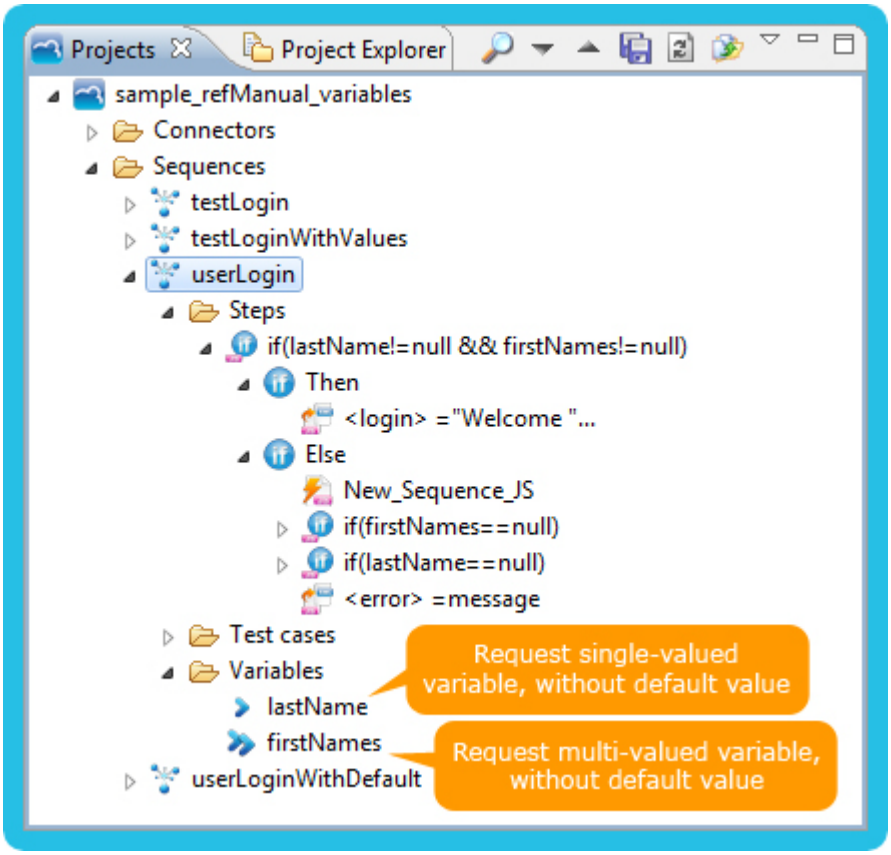


Figure 2 - 43: Request single-valued and multi-valued variables - Sequence and Request variables in Projects view

The JavaScript expression evaluated by the *jIfThenElse* step is:

```
if(lastName!=null && firstName!=null)
```

At runtime, variables (with their default values) are inserted into the JavaScript scope of the context. Thus, while executing the sequence in the Studio, the above expression is not verified and the following XML is returned (see **XML** tab of the sequence editor):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document connector="" context="studio_TestVariables:userLogin"
  contextId="studio_TestVariables:userLogin" fromcache="false"
  generated="Thu Apr 21 10:56:25 CEST 2011" project="TestVariables"
  sequence="userLogin" signature="1303376185017" transaction=""
  version="5.5.0_alpha">
    <error>firstname is required, lastname is required</error>
</document>
```

The sequence is fully implemented and ready to be tested. Let's switch to a web browser displaying the test platform of the project.

The sequence is presented with its variables already prefilled by empty default values:

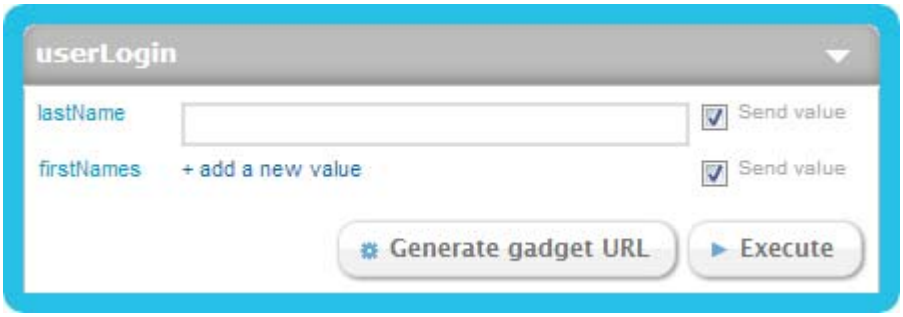


Figure 2 - 44: Request single-valued and multi-valued variables - Sequence and variables in test platform

Several modifications can be performed on the test platform to test the sequence execution. We can modify the `lastName` variable value, let's replace the empty string by "`User`" value, and we can uncheck the `firstNames` variable checkbox, for this variable not to be sent to the sequence call.

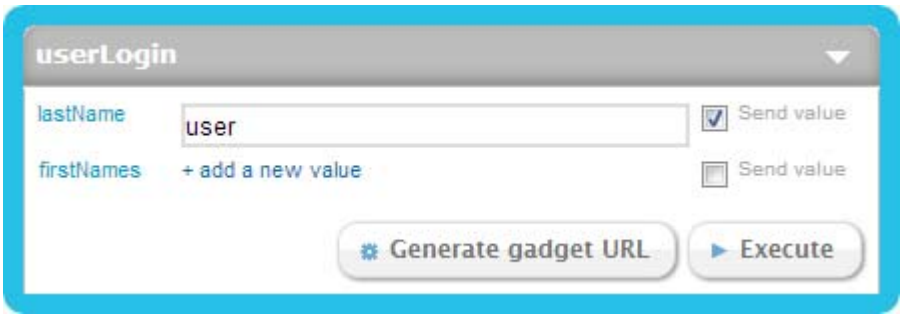


Figure 2 - 45: Request single-valued and multi-valued variables - Sequence and variables in test platform after testing modifications

Test the sequence with updated variables by clicking on the **Execute** button. The sequence is executed and the following XML is returned as result in the **Execution result** panel:

Figure 2 - 46: Request single-valued and multi-valued variables - Sequence execution result in test platform

An `<error>` message is returned, different from the Studio execution, modified by the sequence: "`firstname is required`".

WHAT HAPPENED?

The `lastName` variable was sent to the sequence as a request parameter, thus its default value was overridden with the received value.

The `firstNames` variable was not sent to the sequence, thus its default value (`null`) was used, making the test step fail.

For a second test, we can modify both variables values, checking the `firstNames` variable checkbox for this variable to be sent to the sequence call and replacing its empty string by "`Name`" value.

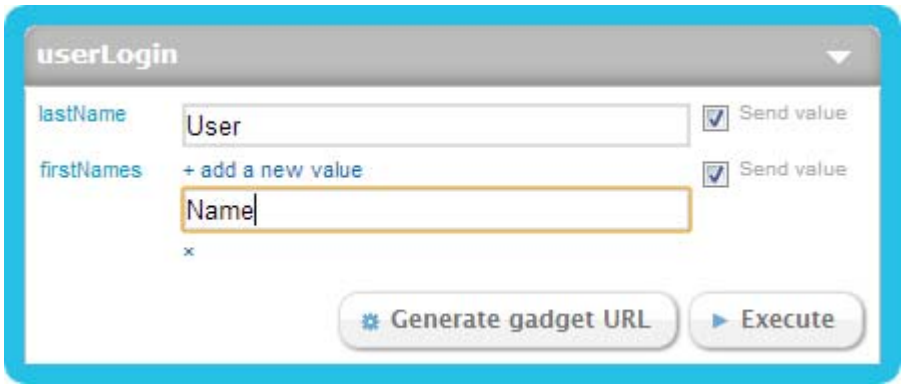


Figure 2 - 47: Request single-valued and multi-valued variables - Sequence and variables in test platform after testing modifications

Test the sequence with updated variables by clicking on the **Execute** button. The sequence is executed and the following XML is returned as result in the **Execution result** panel:

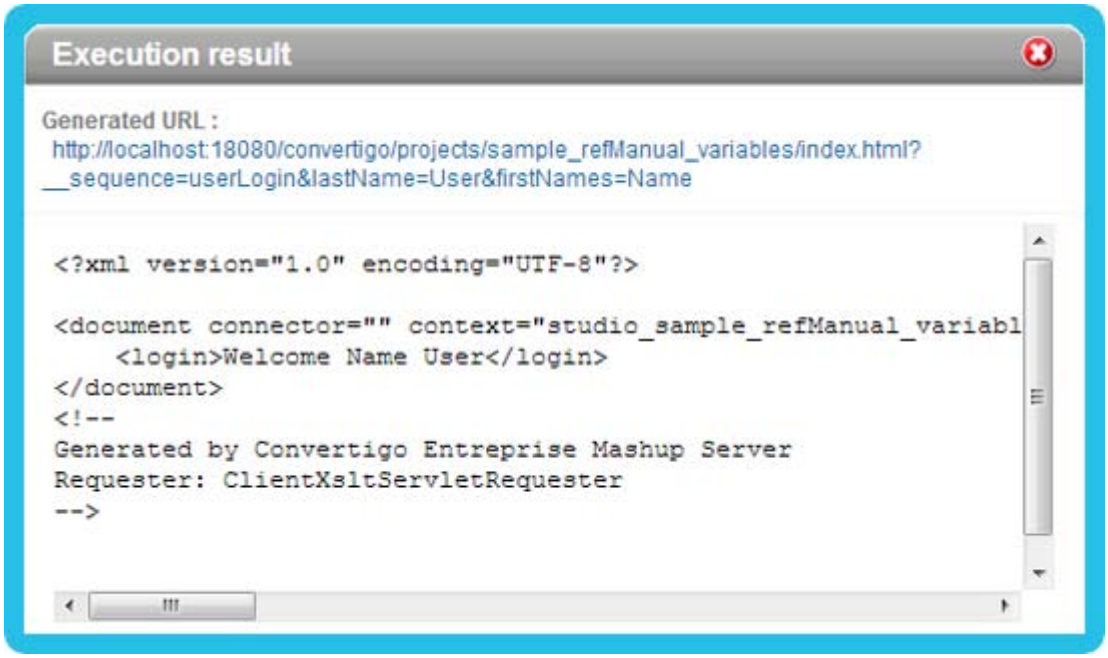


Figure 2 - 48: Request single-valued and multi-valued variables - Sequence execution result in test platform

A `<login>` message is returned, different from the previous execution, modified by the sequence: "`Welcome Name User`".

WHAT HAPPENED?

Both of the variables are sent to the sequence as request parameters, thus their default values are overridden with the received values, making the test succeed.

CONCLUSION

Using prefilled default value for a variable rather than the `null` value depends on the transaction or sequence purpose. We suggest the second solution as it is more constraining and will better help Convertigo developers develop their projects. Moreover, *Test Cases* can be used to test any particular case of transaction or sequence variables values.

HTTP SINGLE-VALUED VARIABLE (REQUESTABLE VARIABLES) 

OBJECT DESCRIPTION

Defines a single-valued variable for an HTTP-based transaction.

An *HTTP single-valued variable* declares a variable which accepts a unique value to an HTTP-based transaction.

This variable is dedicated to HTTP-based transactions only:

- *HTTP transaction*,
- *XML HTTP transaction*,
- *JSON HTTP transaction*,
- *HTML transaction*.

This variable object allows defining HTTP request parameter through the **HTTP name** and **HTTP method** properties.

It can define a default value, specified in the **Default value** property, that is used as HTTP parameter value if no value is found for this variable.

At runtime, the HTTP parameter value is calculated by Convertigo through the following steps:

- the value is received in the request to the transaction,
- if no value is received for this variable, the JavaScript value of the variable is chosen, if a variable of the same name exists in the JavaScript scope of current context,
- if no JavaScript value is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives a value, the default value is used,
- if no default value is specified, the variable is not defined and an Exception can be thrown when trying to access its value in the core of the transaction.

Notes:

- In Convertigo Studio, when an *HTTP single-valued variable* is created in a transaction/sequence, it can be easily replaced by an *HTTP multi-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **MultiValued variable**.
- *HTTP connector* supports OAuth authentication. To enable OAuth, you simply need to provide four variables to any kind of *HTTP transaction*: `__header_oAuthKey`, `__header_oAuthSecret`, `__header_oAuthToken` and `__header_oAuthTokenSecret`. For more information about OAuth in *HTTP connector*, refer to the following article in our Technical Blog: *http://www.convertigo.com/en/how-to/technical-blog/entry/using-oauth-with-convertigo-http-connector.html*

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Cache key	boolean	expert	Defines whether the variable should be part of the cache key. If set to `true`, the variable and its value are added to the cache key which is used to determine whether the transaction's response (or sequence's response) should be pulled from the cache or not. A transaction's cached response (or sequence's cached response) is pulled from the cache when all cache key values are corresponding to a stored cache entry (may contain other data that variables, for example the certificate group defined by some transactions).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Customizable	boolean	expert	Defines whether the variable is customizable. If set to `true`, the variable is used as a customizable preference field in the widget generated from the parent transaction (or sequence) in Convertigo Mashup Composer or any other portal. Note: This property is used when applicable, i.e. when the widget is declared in a portal including customizable preference fields feature.
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.

Property	Type	Category	Description
HTTP method	String	standard	Defines the HTTP method to use for this variable. This property allows choosing which HTTP method has to be used to send the variable in the HTTP request. The following values are available: • `GET`: the transaction is executed as an HTTP GET request and the variable is added to the query string as follows: `?<initial_query_string>&<variable_name>=<variable_value>`. • `POST`: the variable is added to the data sent in the HTTP request as a standard `POST FORM`. `GET` and `POST` method variables can be mixed in a same transaction. If at least one `POST` variable is used, the transaction's HTTP verb is overridden to `POST` by Convertigo.
HTTP name	String	standard	Defines the HTTP parameter name. This property allows defining the name of the HTTP variable sent in the request by Convertigo executing the parent transaction. If the HTTP-based transaction emulates a form submission, this property can match the `name` attribute of an HTML input field. If the HTTP-based transaction emulates a resource access by URL, this property can match one of the variables names from the query string (between "&" and "=" characters).
Is a file upload	boolean	expert	Defines whether the variable is an uploaded file. When set to `true`, this property indicates that the transaction/sequence should receive an uploaded file in this variable. When received, the uploaded file is stored in a temporary folder and deleted at the end of the transaction/sequence. In the transaction/ sequence execution context, the variable contains the path of the temporary file. Note: This property value is used only by the *Test Platform* to allow the developer testing the transaction/sequence. When receiving a multi-part request, Convertigo can set any variable as an uploaded file.
Schema type QName	XmlQName	expert	Assigned schema type qualified name

Property	Type	Category	Description
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.
WSDL exposed	boolean	expert	Defines whether the variable is exposed in web service. If set to `true`, variable definition is inserted in the project's WSDL as a method parameter. Note: This property value is ignored if the **Public method** property of the parent transaction (or sequence) is set to `false`, which means the method itself is not exposed in the web service.

EXAMPLES

HTTP and HTML transactions may use variables useful for their execution process. To explain the best use of these variables and their overrides, the following examples are based on Convertigo Web Integrator connectors declared in several projects.

Example 1

Let's consider the `searchGoogle` transaction set in the context of the "*Starting With Convertigo Web Integrator*" tutorial. This transaction searches for an input keyword in Google search engine and accumulates the results into an XML structure thanks to a *Table* extraction rule.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

To do the previously described behavior, this transaction defines one single-valued variable called `keyword`. Being an *HTML transaction*, the variable it uses is an *HTTP single-valued variable*. It is created with the following parameters:

```
HTTP single-valued variable [
```

```
  description=Keyword to be searched
  default value=""
  HTTP method=GET
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

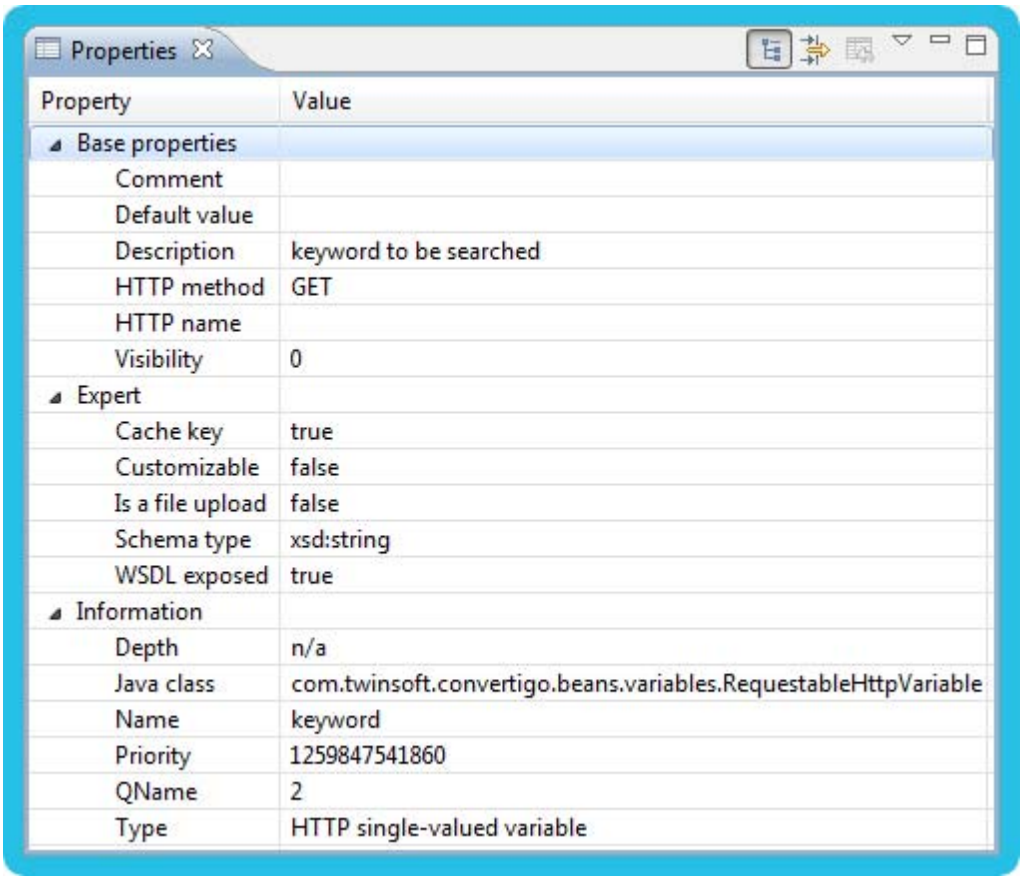


Figure 2 - 49: HTTP single-valued variable - Configuration example

As no **HTTP name** is defined for this variable, the **HTTP method** property value (set to `GET` by default) is ignored at transaction execution, i.e. the variable is not added to the HTTP request to the target website when the transaction starts.

The *HTTP single-valued variable* object, named `keyword`, is created in the **Variables** folder of the transaction, and appears as follows in the **Projects** view:

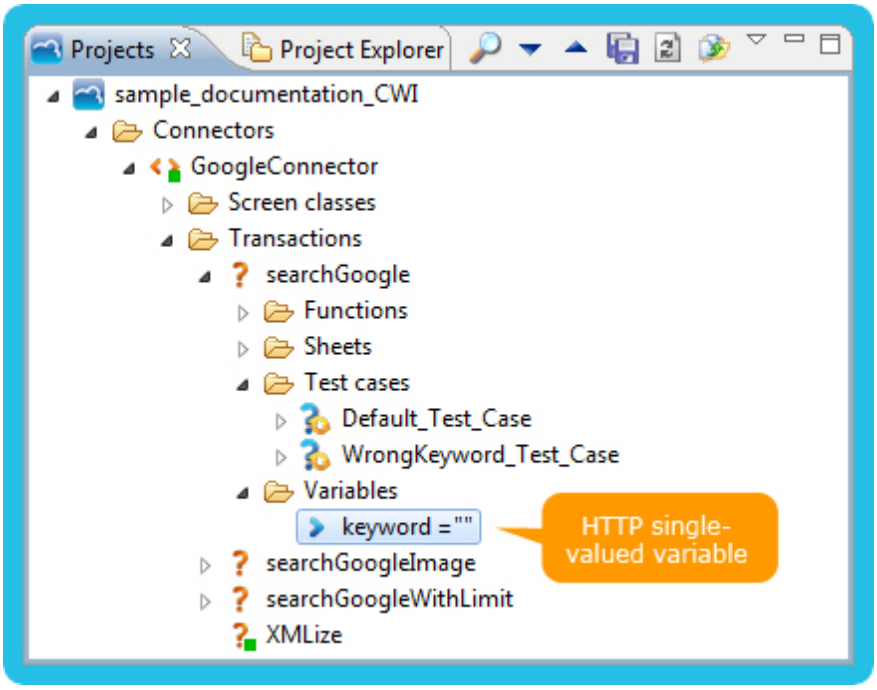


Figure 2 - 50: HTTP single-valued variable - HTTP Variable and parent transaction in Projects view

The variable's default value is displayed in the **Projects** tree, after the "=" character next to the object. Here the value is an empty string, not `null`. Some test cases are implemented to define test values for this variable. For more information about *Test cases*, see "*Test case*" documentation and examples.

At runtime, *HTTP variable* (with its received value) is inserted into the JavaScript scope of the transaction. If executing the transaction directly in the Studio, the research is executed with the variable default value, which is an empty string, and fails in timeout.

Now switch to a web browser displaying the test platform of this project. The test platform shows the `searchGoogle` transaction, with its `keyword` variable (empty string default value) and its test cases:

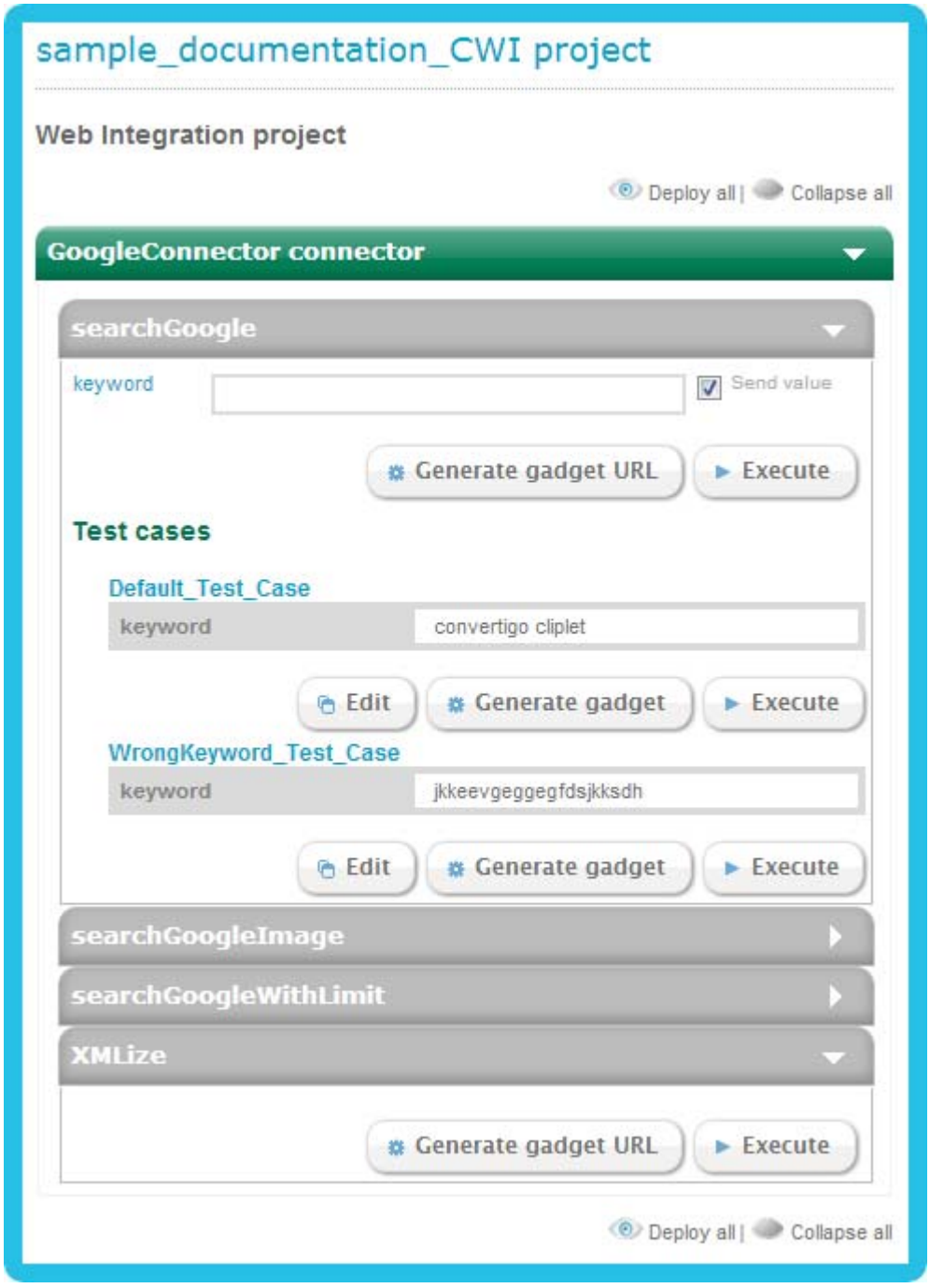


Figure 2 - 51: HTTP single-valued variable - HTTP Variable and parent transaction in test platform

Variable value can be modified on the test platform to test the transaction execution. We can set the `keyword` variable value to "`convertigo sequencer`" for example:

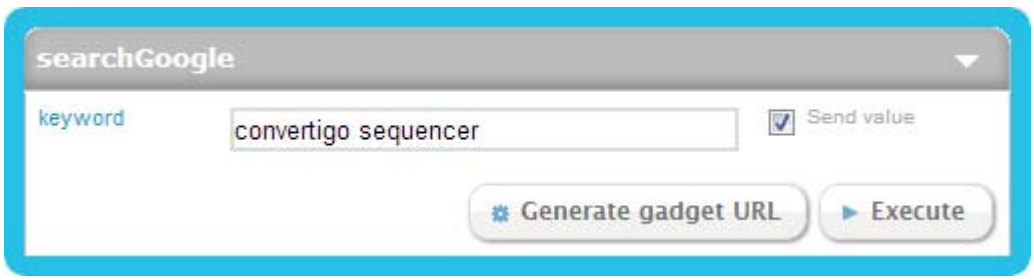


Figure 2 - 52: HTTP single-valued variable - Updating variable value in test platformfor testing

Test the transaction with updated variable by clicking on the **Execute** button. The transaction is executed with the variable "`convertigo sequencer`" value sent through the test platform, and the result is displayed in the **Execution result** panel:

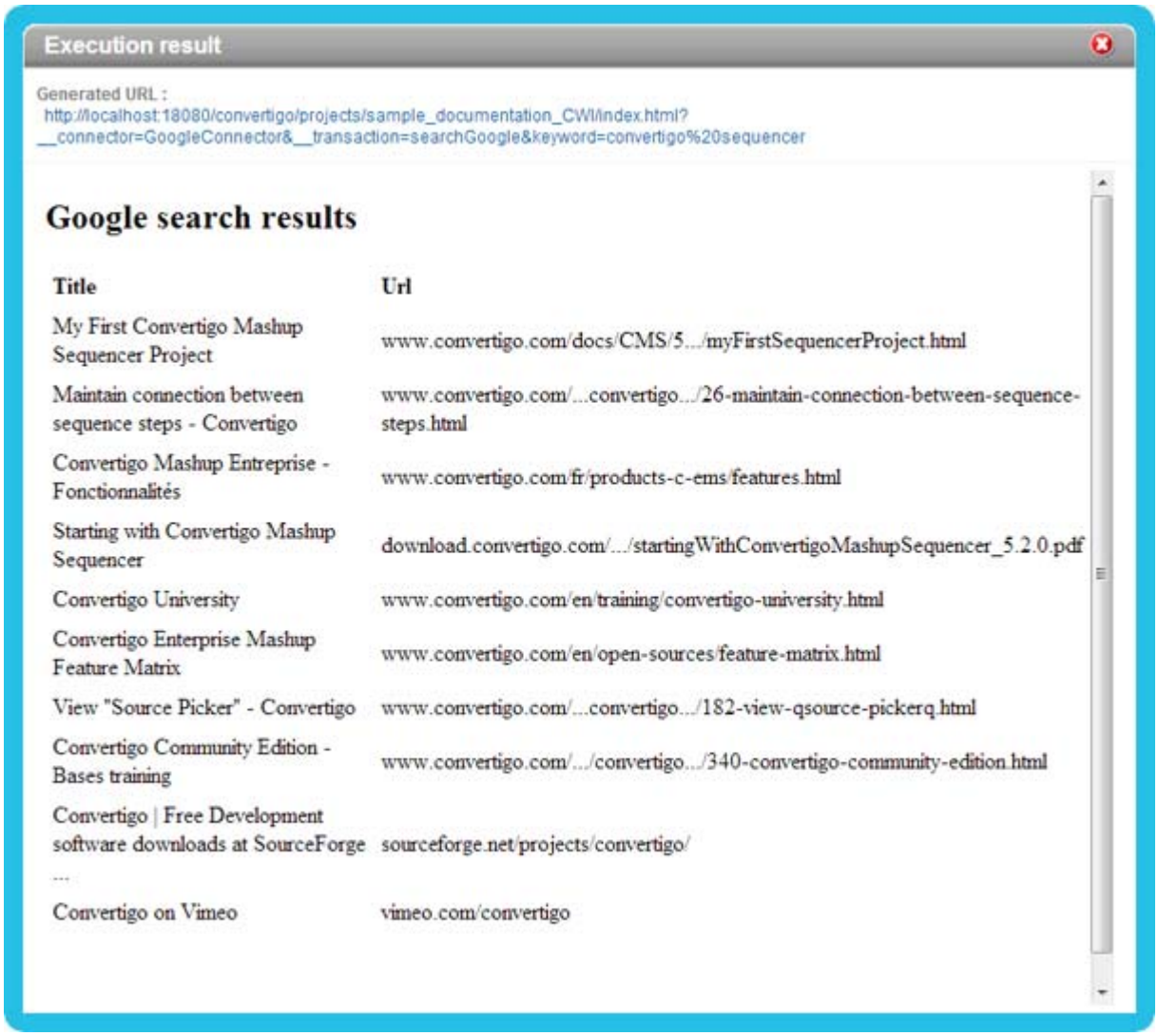


Figure 2 - 53: HTTP single-valued variable - Transaction execution result in test platform

In the Studio, the resulting XML is displayed in the **XML** tab of the **Output** part of the connector editor:

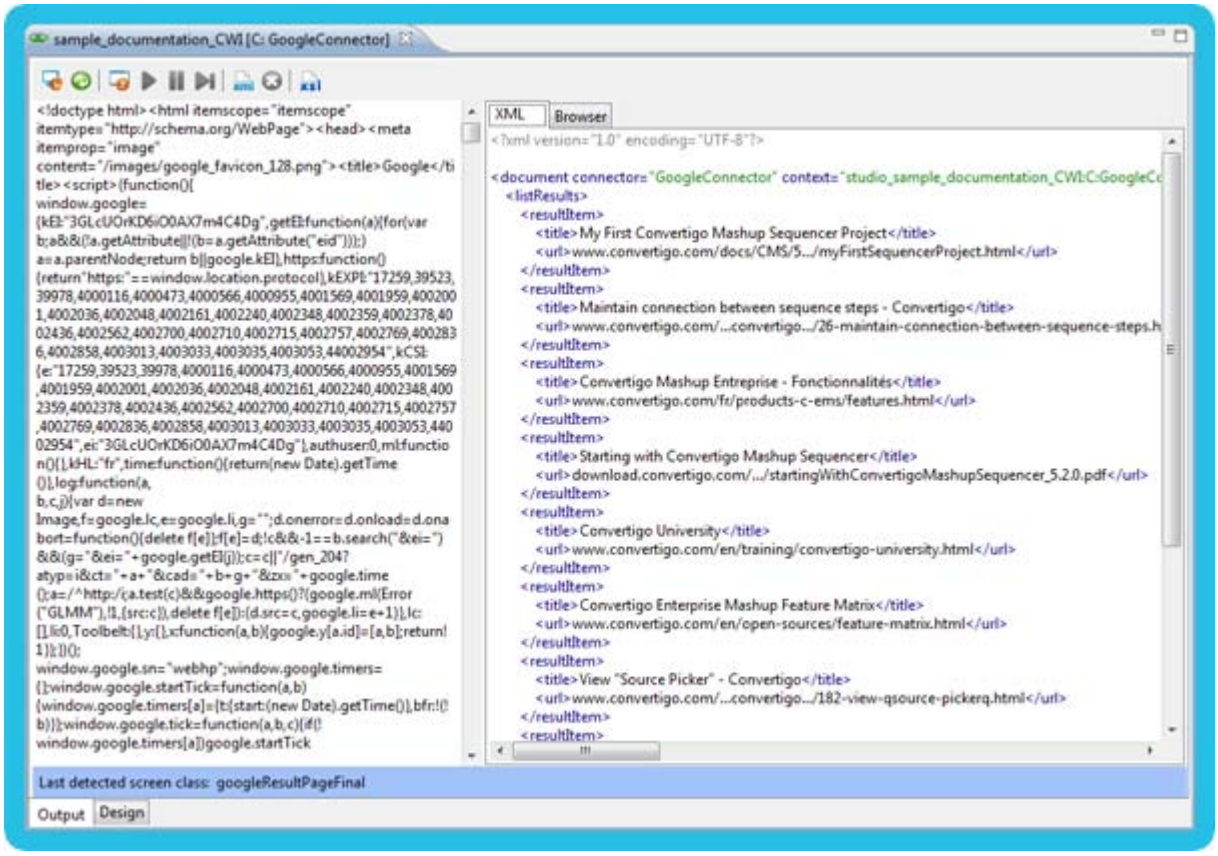


Figure 2 - 54: HTTP single-valued variable - Transaction execution result in connector editor

The results retrieved from Google results page are those corresponding to the research performed on the dynamically passed keyword.

WHAT HAPPENED?

The `keyword` variable was sent to the transaction as a request parameter, thus its default empty string value was overridden with the received value.

Example 2

The second example is based on an *HTML connector*, reaching Google website, defined in a project named `sample_refManual_variables`.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

Let's consider an *HTML transaction*, named `searchGoogleHTTP`, similar to previous `searchGoogle` transaction, but, instead of using statements to input the keyword in the search field and click on the button, this transaction uses its **Subpath** property and its variable to directly make the research on the given keyword through an HTTP request.

The Google request URL to perform a research is:

`http://www.google.com/search?q=<keyword>`

where q is the keyword HTTP parameter.

The transaction implements this HTTP request thanks to its properties and directly executes the research when launched. It declares an *HTTP single-valued variable*, named `keyword`, created with the followings parameters:

```
HTTP single-valued variable [
  description=Keyword to be searched
  default value="convertigo"
  HTTP method=GET
  HTTP name=q
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

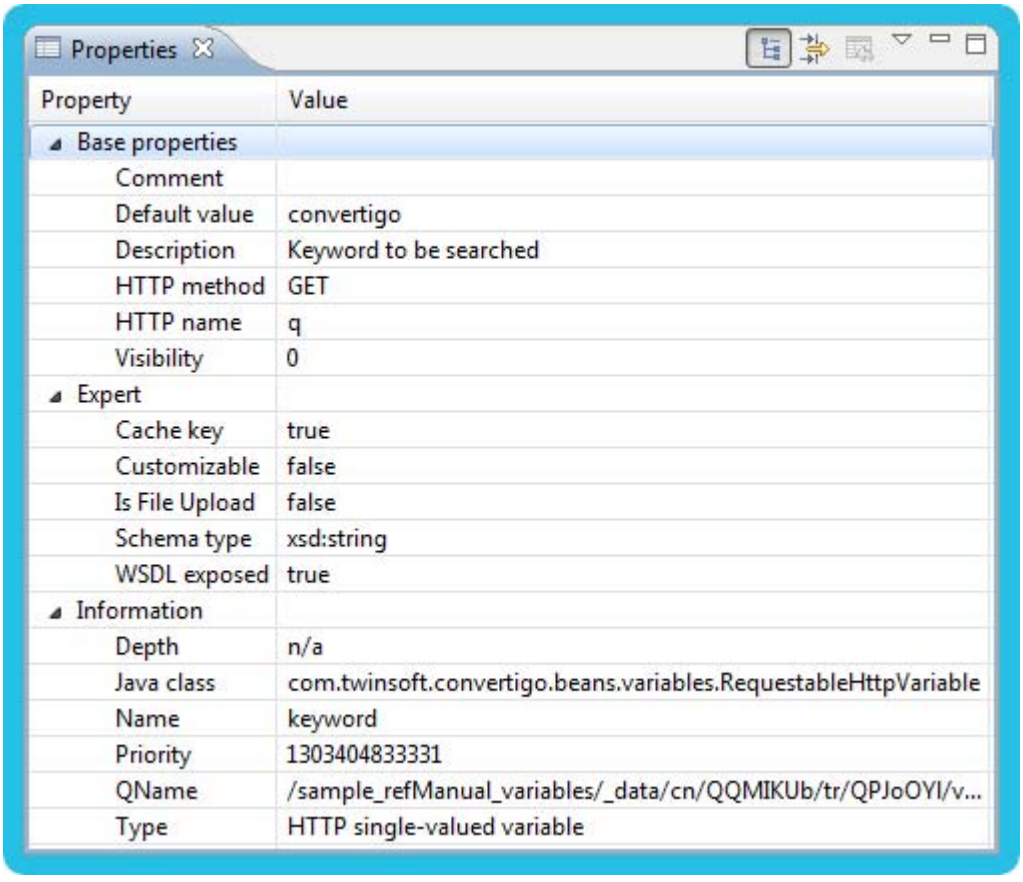


Figure 2 - 55: HTTP single-valued variable - Configuration example

As an **HTTP name** is defined for this variable, the **HTTP method** property value (set to `GET`) defines that the variable value will be sent as a request parameter in the query string, with the name "q" defined by the **HTTP name** property.

The *HTTP single-valued variable* object, named `keyword`, is created in the **Variables** folder of the transaction, and appears as follows in the **Projects** view:

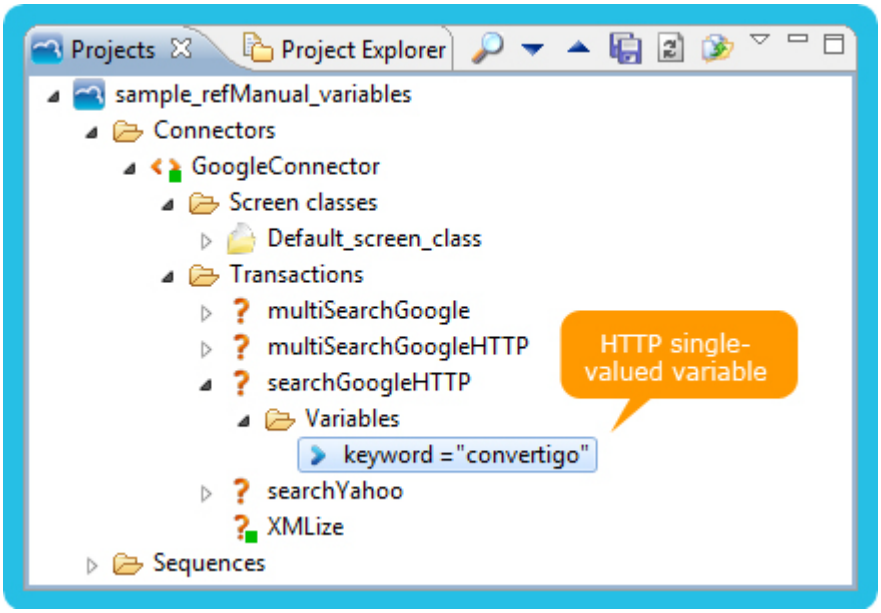


Figure 2 - 56: HTTP single-valued variable - Variable object in Projects view

The variable's default value is displayed in the **Projects** tree, after the "=" character next to the object.

The transaction is implemented and can be tested in the Studio. To do so, the first step is to open the connector editor by double-clicking on the *HTML connector* in the **Projects** view (if not already open). The internal browser connects to `http://www.google.com/` as seen in **Design** tab of the editor:

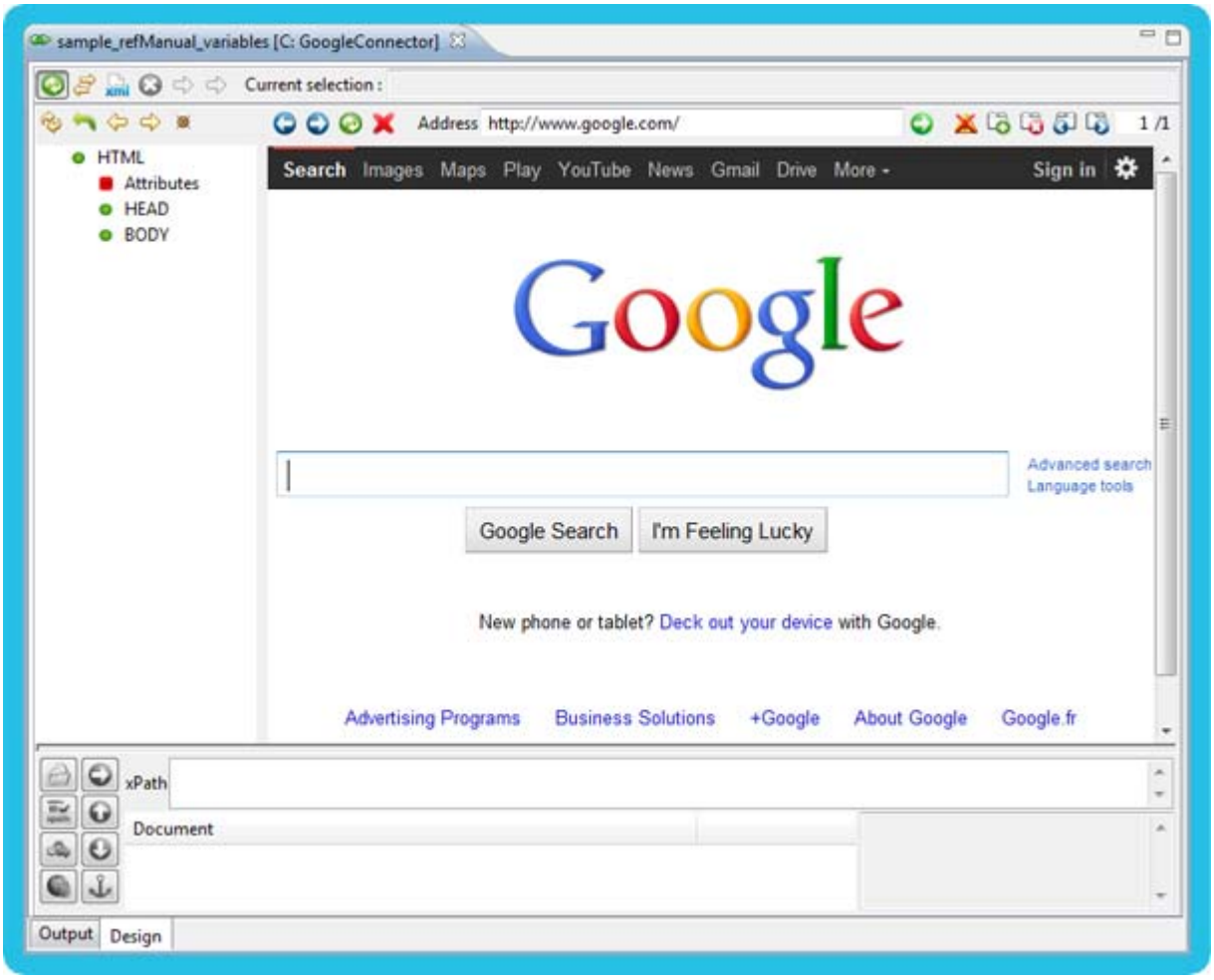


Figure 2 - 57: HTTP single-valued variable - Connector editor

Now execute the `searchGoogleHTTP` transaction by pressing F5 key on the transaction object in the **Projects** view. The internal browser connects to `http://www.google.com/search?q=convertigo` as seen in **Design** tab of the editor:

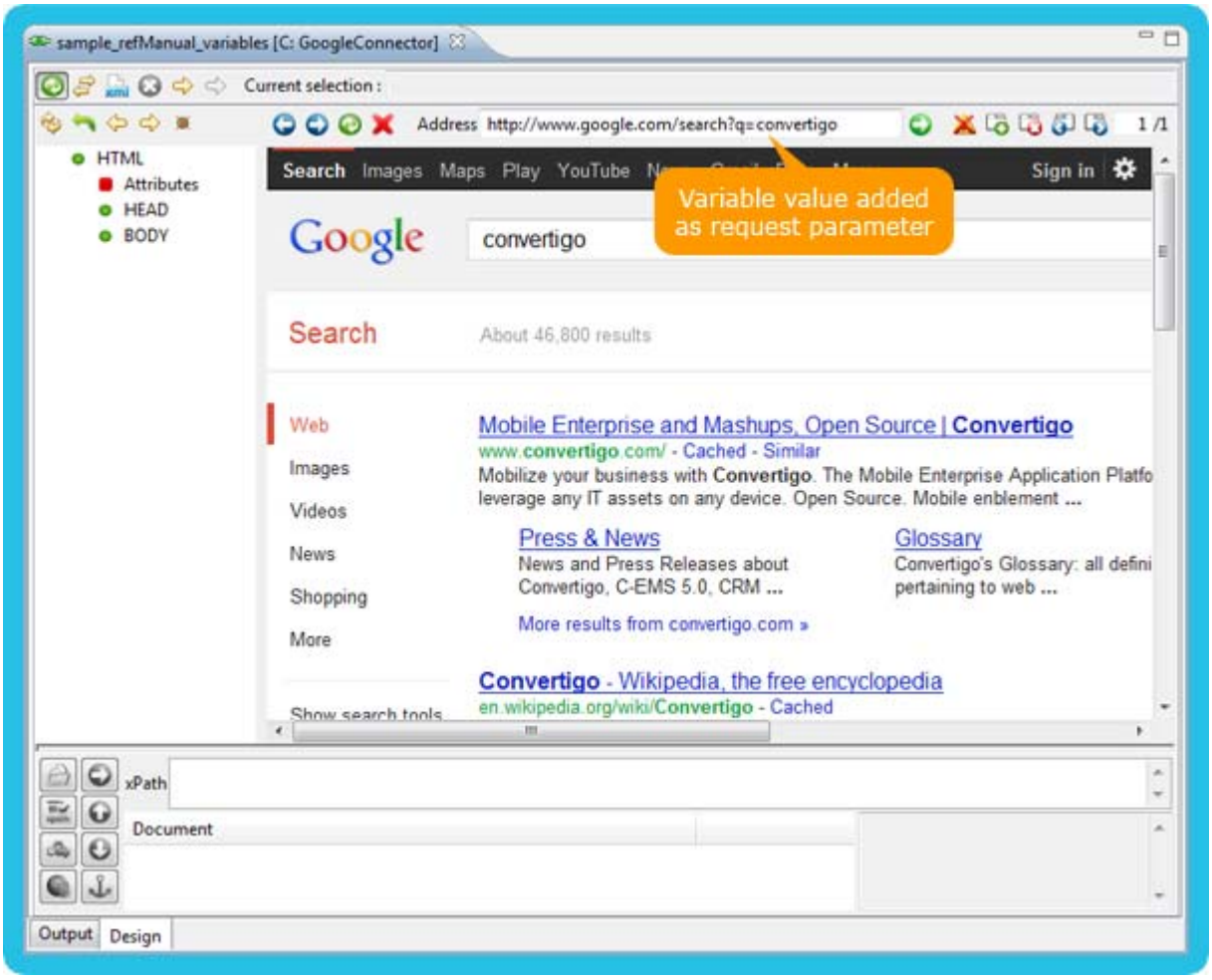


Figure 2 - 58: HTTP single-valued variable - Connector editor after transaction execution

WHAT HAPPENED?

The connector has built the request URL by concatenating the server address, the transaction subpath and a query string. This query string was formed with the variable details.

Each variable which **HTTP method** property set to `GET` is added to the query string as a parameter:

- which name is set by the variable's **HTTP name** property,
- which value is the variable's value.

In this case, the parameter name is q and its value is the transaction's `keyword` default value "`convertigo`".

Notes:

- If the **HTTP name** property is empty, the parameter is not added, thus not sent.
- If the **HTTP method** property is set to `POST`, the parameter is added to the body of request (POST data) rather than to the query string.

HTTP multi-valued variable (Requestable variables) 

Object Description

Defines a multi-valued variable for an HTTP-based transaction.

An *HTTP multi-valued variable* declares a variable which accepts one or more values to an HTTP-based transaction.

This variable is dedicated to HTTP-based transactions only:

- *HTTP transaction*,
- *XML HTTP transaction*,
- *JSON HTTP transaction*,
- *HTML transaction*.

This variable object allows defining HTTP request parameters through the **HTTP name** and **HTTP method** properties.

It can define a default list of value(s), specified in the **Default value** property, that are used as HTTP parameters values if no value is found for this variable.

At runtime, the HTTP parameters values are calculated by Convertigo through the following steps:

- the values are received in the request to the transaction,
- if no value is received for this variable, the JavaScript value of the variable is chosen, if a variable of the same name exists in the JavaScript scope of current context (this JavaScript variable should be an array of values),
- if no JavaScript value is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives values, the default list of values is used,
- if no default value is specified, the variable is not defined and an Exception can be thrown when trying to access its values in the core of the transaction.

Notes:

- In Convertigo Studio, when an *HTTP multi-valued variable* is created in a transaction/sequence, it can be easily replaced by an *HTTP single-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **SingleValued variable**.
- *HTTP connector* supports OAuth authentication. To enable OAuth, you simply need to provide four variables to any kind of *HTTP transaction*: `__header_oAuthKey`, `__header_oAuthSecret`, `__header_oAuthToken` and `__header_oAuthTokenSecret`. For more information about OAuth in *HTTP connector*, refer to the following article in our Technical Blog: *http://www.convertigo.com/en/how-to/technical-blog/entry/using-oauth-with-convertigo-http-connector.html*

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Cache key	boolean	expert	Defines whether the variable should be part of the cache key. If set to `true`, the variable and its value are added to the cache key which is used to determine whether the transaction's response (or sequence's response) should be pulled from the cache or not. A transaction's cached response (or sequence's cached response) is pulled from the cache when all cache key values are corresponding to a stored cache entry (may contain other data that variables, for example the certificate group defined by some transactions).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Customizable	boolean	expert	Defines whether the variable is customizable. If set to `true`, the variable is used as a customizable preference field in the widget generated from the parent transaction (or sequence) in Convertigo Mashup Composer or any other portal. Note: This property is used when applicable, i.e. when the widget is declared in a portal including customizable preference fields feature.
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.

Property	Type	Category	Description
HTTP method	String	standard	Defines the HTTP method to use for this variable. This property allows choosing which HTTP method has to be used to send the variable in the HTTP request. The following values are available: • `GET`: the transaction is executed as an HTTP GET request and the variable is added to the query string as follows: `?<initial_query_string>&<variable_name>=<variable_value>`. • `POST`: the variable is added to the data sent in the HTTP request as a standard `POST FORM`. `GET` and `POST` method variables can be mixed in a same transaction. If at least one `POST` variable is used, the transaction's HTTP verb is overridden to `POST` by Convertigo.
HTTP name	String	standard	Defines the HTTP parameter name. This property allows defining the name of the HTTP variable sent in the request by Convertigo executing the parent transaction. If the HTTP-based transaction emulates a form submission, this property can match the `name` attribute of an HTML input field. If the HTTP-based transaction emulates a resource access by URL, this property can match one of the variables names from the query string (between "&" and "=" characters).
Is a file upload	boolean	expert	Defines whether the variable is an uploaded file. When set to `true`, this property indicates that the transaction/sequence should receive an uploaded file in this variable. When received, the uploaded file is stored in a temporary folder and deleted at the end of the transaction/sequence. In the transaction/ sequence execution context, the variable contains the path of the temporary file. Note: This property value is used only by the *Test Platform* to allow the developer testing the transaction/sequence. When receiving a multi-part request, Convertigo can set any variable as an uploaded file.
Schema type QName	XmlQName	expert	Assigned schema type qualified name
Soap array	boolean	standard	Defines if the multi-valued variable should be seen as a Soap Array of a occurrence of variables. In the case of transaction or sequence defined as a public SOAP method, this property allows to specify of the current multi-valued variable has to be seen in SOAP envelope as a Soap Array with multiple values inside it or as an occurrence of identical variables.

Property	Type	Category	Description
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.
WSDL exposed	boolean	expert	Defines whether the variable is exposed in web service. If set to `true`, variable definition is inserted in the project's WSDL as a method parameter. Note: This property value is ignored if the **Public method** property of the parent transaction (or sequence) is set to `false`, which means the method itself is not exposed in the web service.

EXAMPLES

HTTP and HTML transactions may use variables useful for their execution process. To explain the best use of these variables and their overrides, the following examples are based on a Convertigo Web Integrator connector declared in a project named `sample_refManual_variables`.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **Variables examples** in the **New Project** wizard.*

Example 1

Let's consider a `multiSearchGoogle` transaction, similar to `searchGoogle` transaction set in the context of the "*Starting With Convertigo Web Integrator*" Quick Guide, but declaring a multi-valued variable instead of the single-valued variable of the Quick guide's transaction. This transaction searches in Google search engine for a keyword built from several input keywords and accumulates the results into an XML structure thanks to a *Table* extraction rule. It is created in an *HTML connector* of the `sample_refManual_variables` project.

As said in introduction, this transaction defines one multi-valued variable called `keywords`. Being an *HTML transaction*, the variable it uses is an *HTTP multi-valued variable*. It is created

with the following parameters:

```
HTTP multi-valued variable [
  description=Keywords to be searched
  default value=["convertigo", "cliplet"]
  HTTP method=GET
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

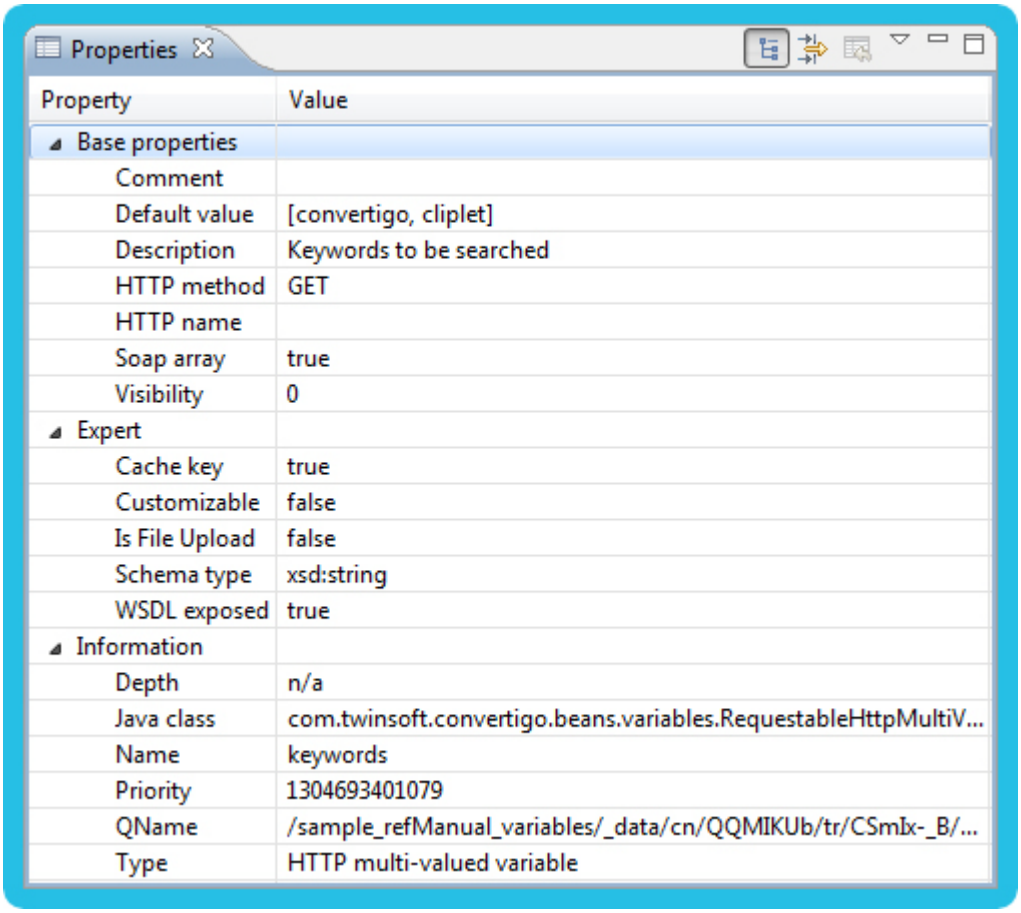


Figure 2 - 59: HTTP multi-valued variable - Configuration example

As no **HTTP name** is defined for this variable, the **HTTP method** property value (set to `GET` by default) is ignored at transaction execution, i.e. the variables are not added to the HTTP request to the target website when the transaction starts.

The **Default value** property of this *HTTP multi-valued variable* is set to fixed values (not `null`). This property is edited in the **Array** editor:

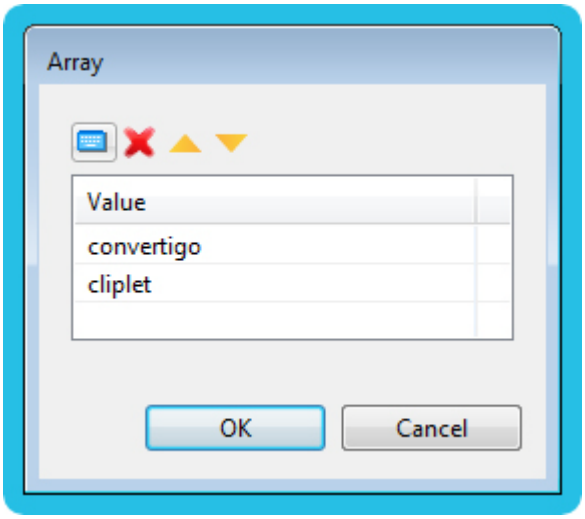


Figure 2 - 60: HTTP multi-valued variable - Default value property in Array editor

The *HTTP multi-valued variable* object, named `keywords`, is created in the **Variables** folder of the transaction, and appears as follows in the **Projects** view:

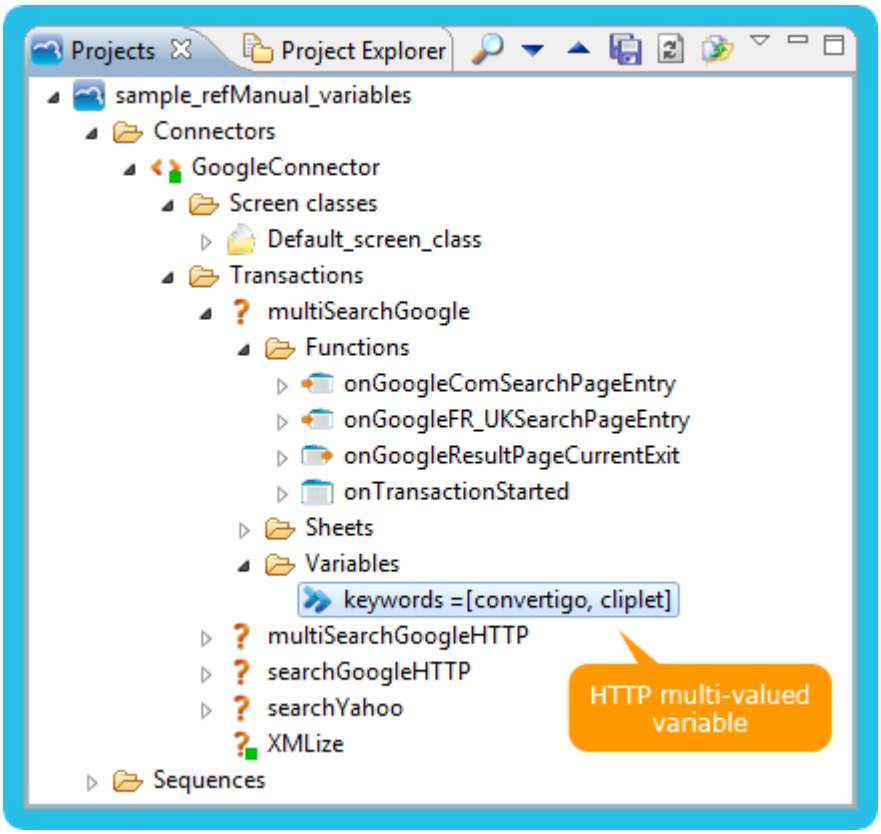


Figure 2 - 61: HTTP multi-valued variable - HTTP Variable and parent transaction in Projects view

The variable's default values are displayed in the **Projects** tree, after the "=" character next to the object.

At runtime, *HTTP variables* (with its default values) are inserted into the JavaScript scope of the transaction. Thus, while executing the transaction in the Studio, the research is executed with the variable default values "`convertigo`" and "`cliplet`".

In the Studio, the resulting XML is displayed in the **XML** tab of the **Output** part of the connector

editor:

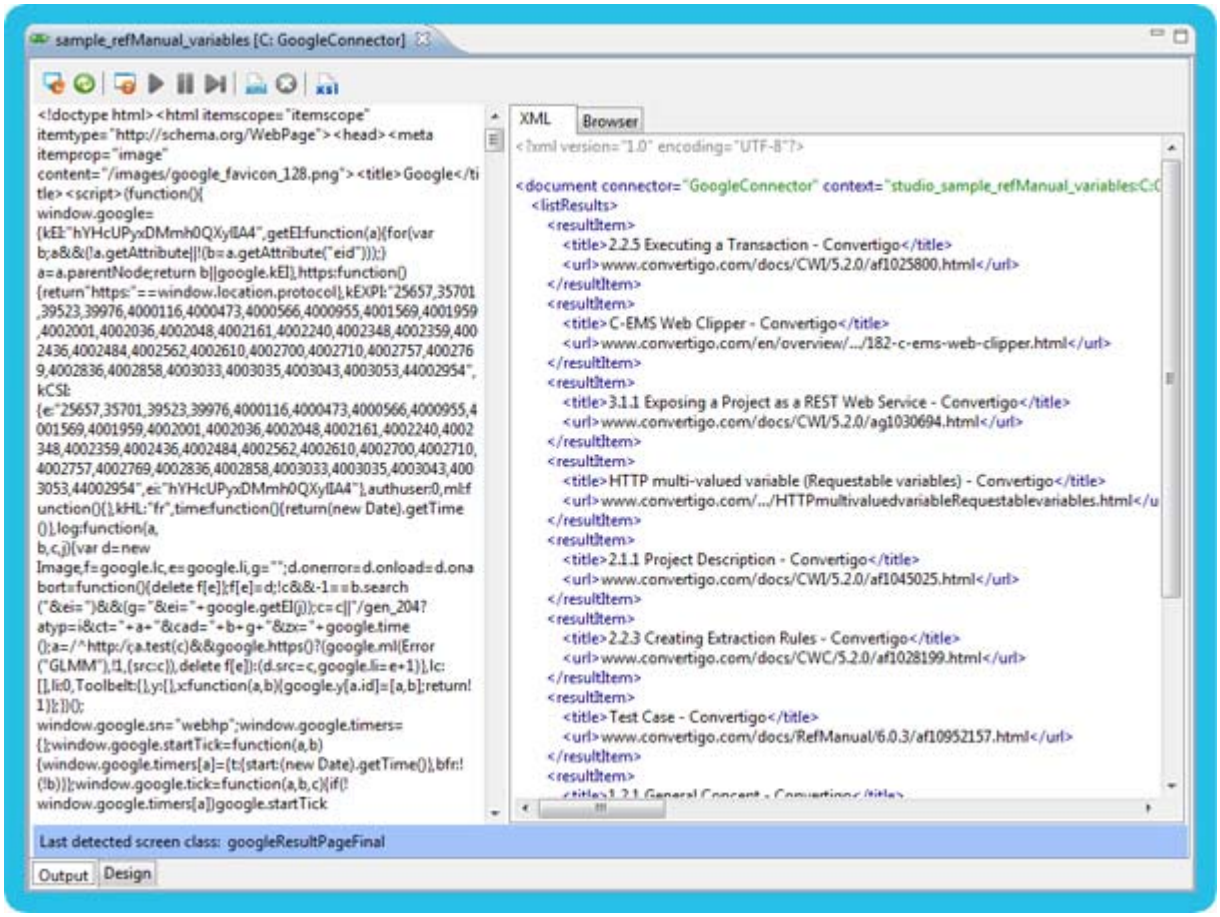


Figure 2 - 62: HTTP multi-valued variable - Transaction execution result in connector editor

The results retrieved from Google results page are those corresponding to the research performed on the variable default value as keyword.

Now switch to a web browser displaying the test platform of this project. The test platform shows the `multiSearchGoogle` transaction, with its `keywords` multi-valued variable and its "`convertigo`" and "`cliplet`" default values:

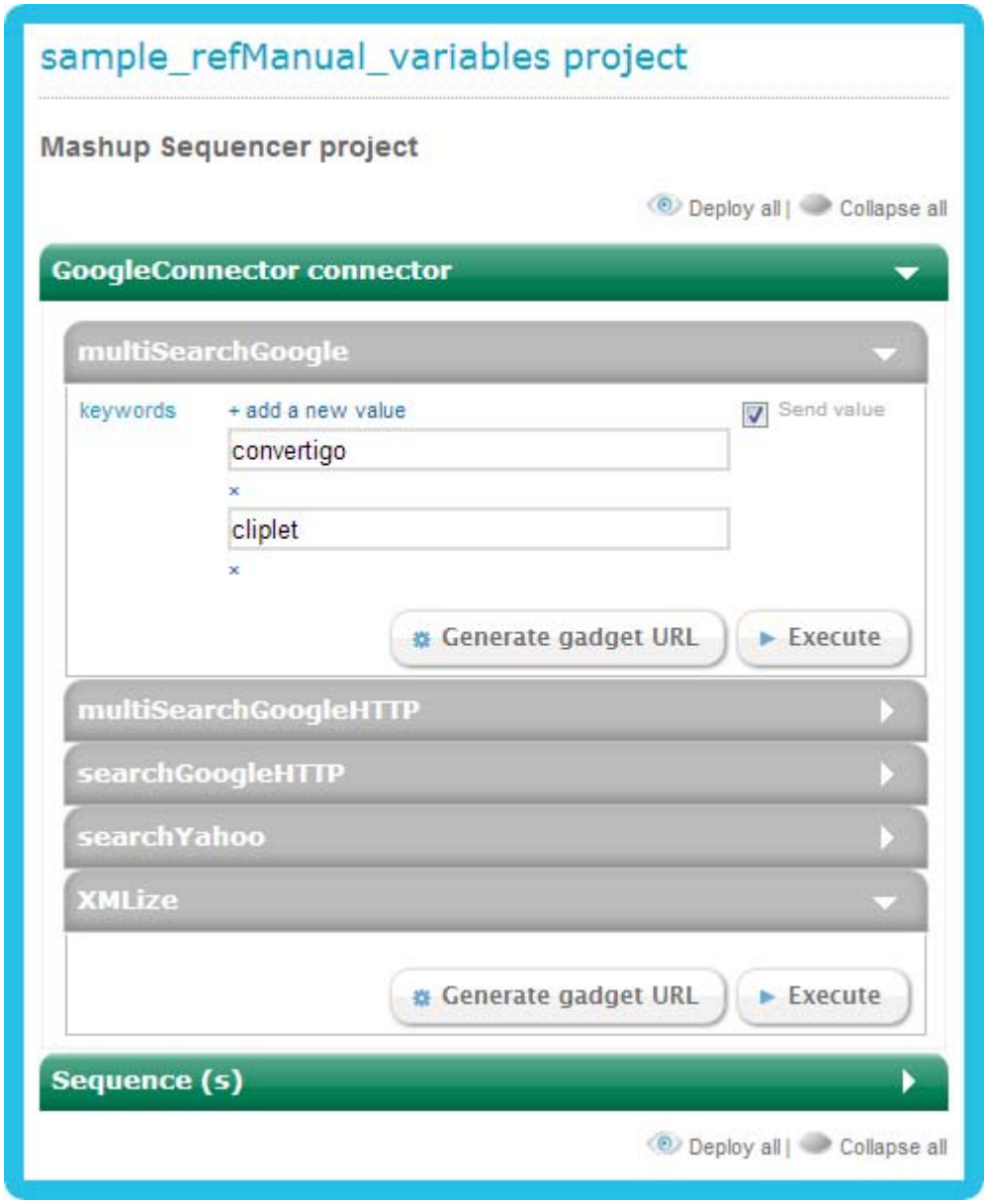


Figure 2 - 63: HTTP multi-valued variable - HTTP Variable and parent transaction in test platform

Variable values can be modified on the test platform to test the transaction execution. We can update the `keywords` variable values to "`convertigo`", "`mobility`" and add a third value "`sequencer`" for example:

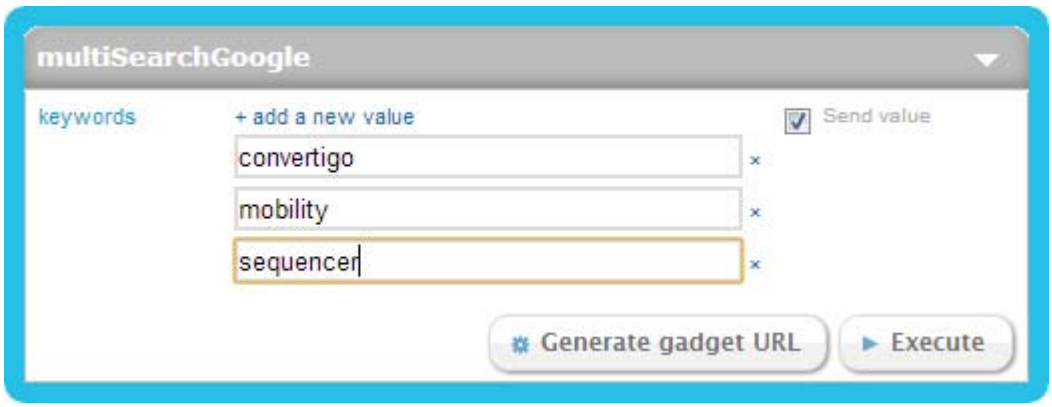


Figure 2 - 64: HTTP multi-valued variable - Updating variable values in test platformfor testing

Test the transaction with updated variable by clicking on the **Execute** button. The transaction is executed with the variable "`convertigo`", "`mobility`" and "`sequencer`" values sent through the test platform, and the result is displayed in the **Execution result** panel:

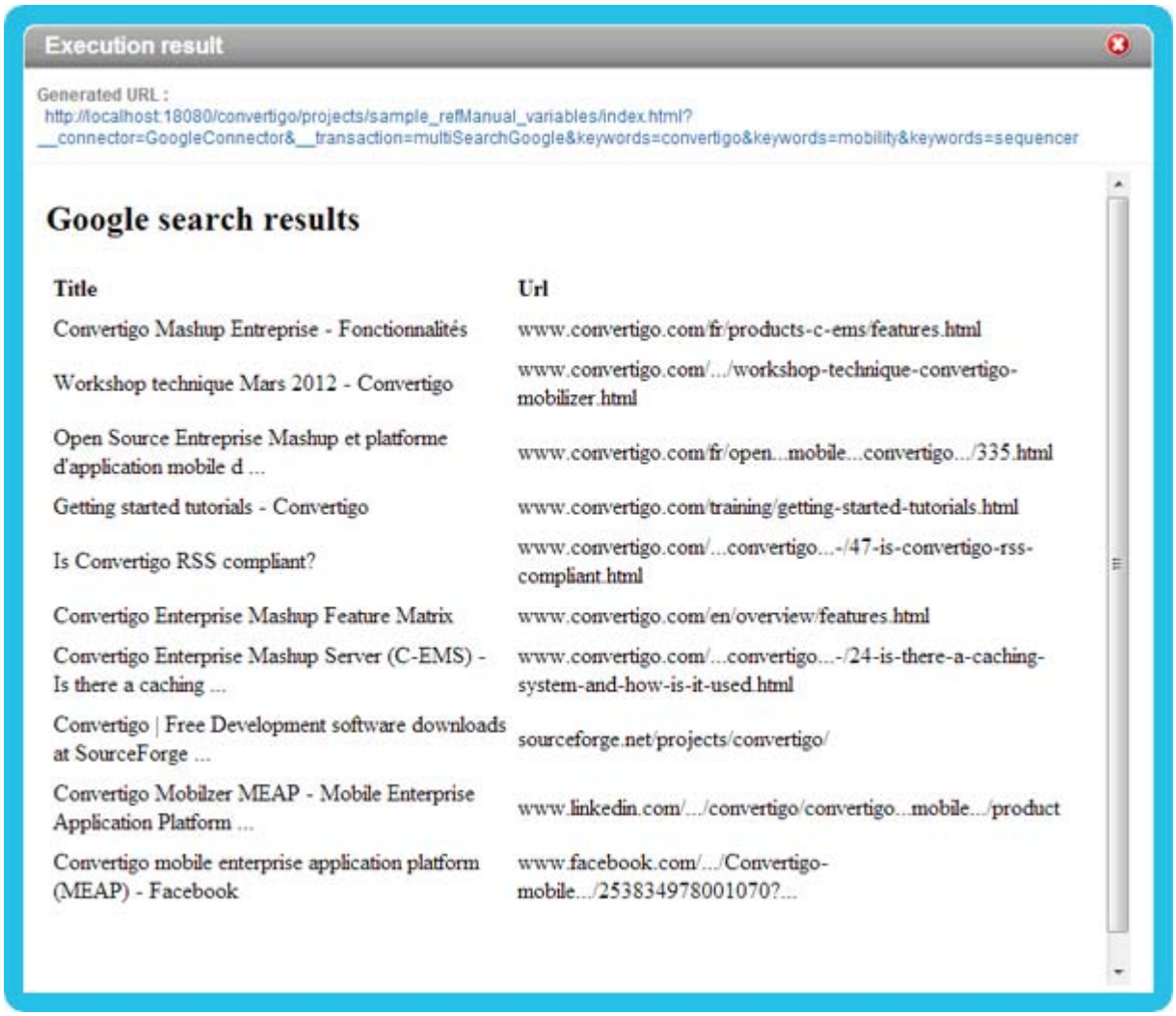


Figure 2 - 65: HTTP multi-valued variable - Transaction execution result in test platform

The results retrieved from Google results page are those corresponding to the research performed on the dynamically passed keywords.

WHAT HAPPENED?

The `keywords` variable were sent to the transaction as request parameters, thus its default values were overridden with the received values.

Example 2

Let's consider a transaction, named `multiSearchGoogleHTTP`, similar to previous `multiSearchGoogle` transaction, but, instead of using statements to input the keywords in the search field and click on the button, this transaction uses its **Subpath** property and its variables to directly make the research on the given keywords through an HTTP request.

The Google request URL to perform a research is:

```
http://www.google.com/search?q=<keyword>
```

where q is the keyword HTTP parameter.

This *HTML transaction* implements this HTTP request and directly executes the research when launched. It is created in an *HTML connector* of the `sample_refManual_variables` project.

This transaction declares an *HTTP multi-valued variable*, named `keywords`, created with the followings parameters:

```
HTTP single-valued variable [
  description=Keywords to be searched
  default value=["convertigo", "cliplet"]
  HTTP method=GET
  HTTP name=q
  cache key=true
  customizable=false
  schema type=xsd:string
  WSDL exposed=true
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

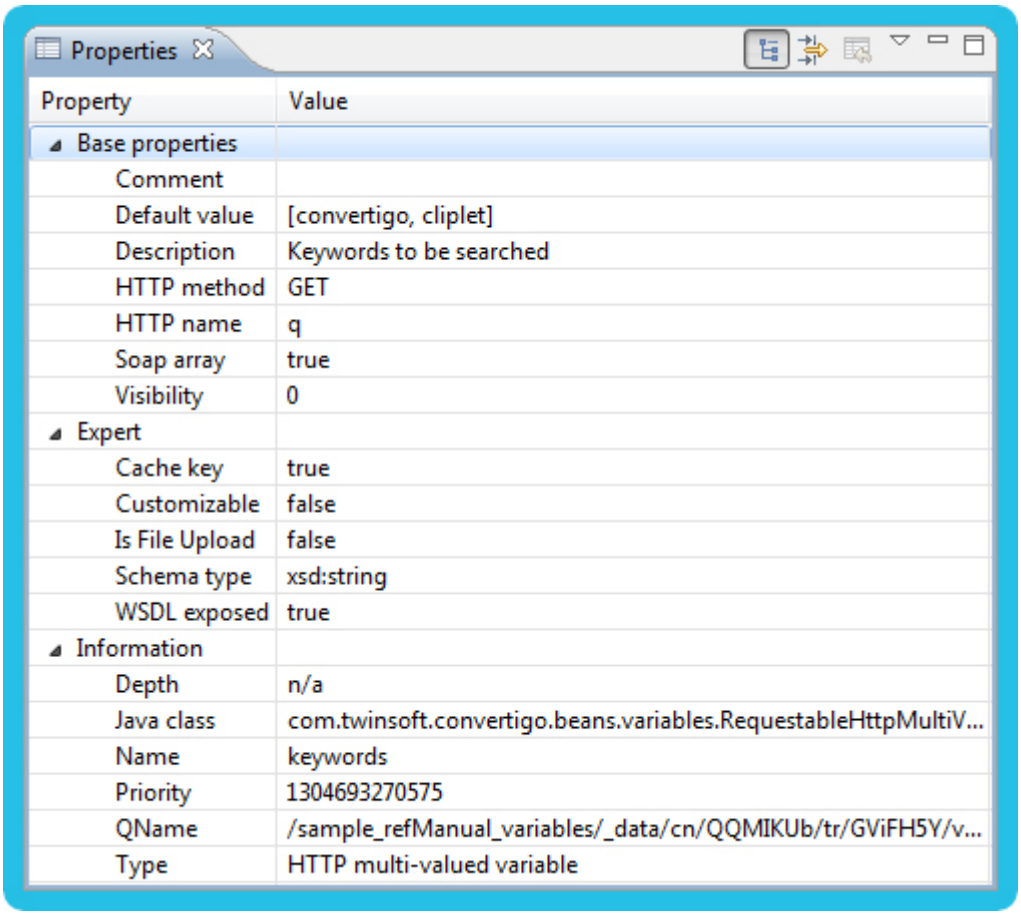


Figure 2 - 66: HTTP multi-valued variable - Configuration example

As an **HTTP name** is defined for this variable, the **HTTP method** property value (set to `GET`) defines that the variable values will be sent as request parameters in the query string, with the name "q" defined by the **HTTP name** property.

The **Default value** property of this *HTTP multi-valued variable* is set to fixed values (not `null`). This property is edited in the **Array** editor:

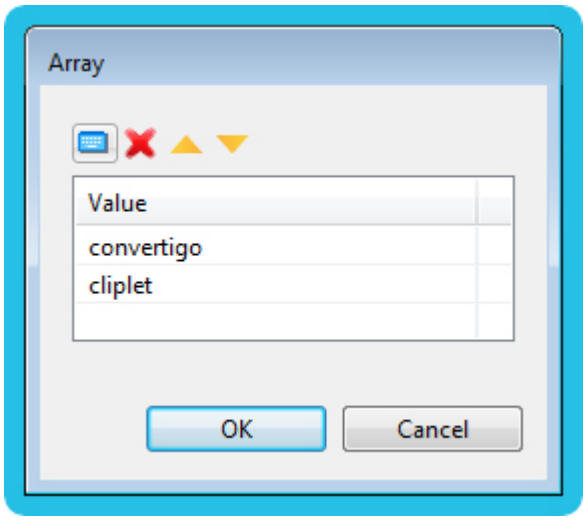


Figure 2 - 67: HTTP multi-valued variable - Default value property in Array editor

The *HTTP multi-valued variable* object, named `keywords`, is created in the **Variables** folder of the transaction, and appears as follows in the **Projects** view:

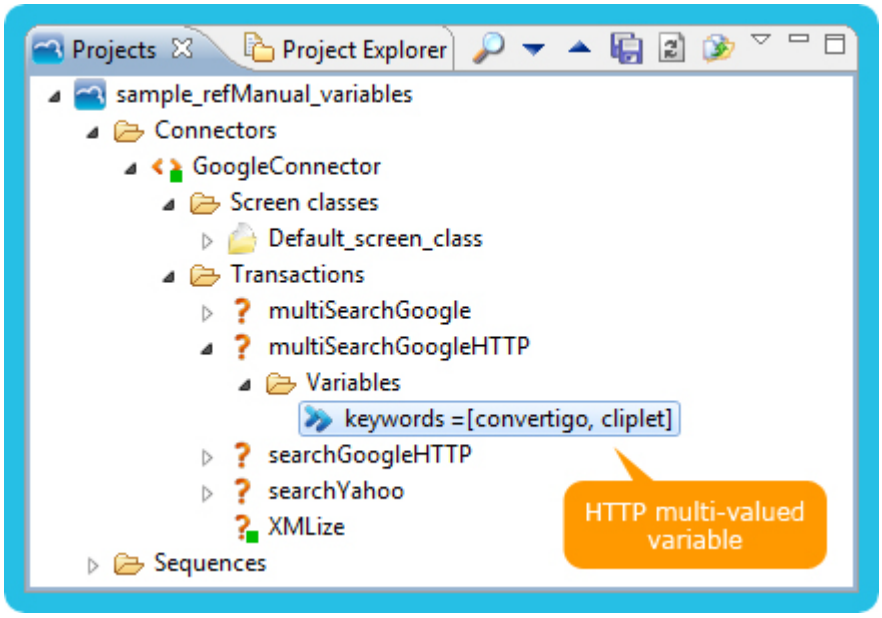


Figure 2 - 68: HTTP multi-valued variable - HTTP Variable and parent transaction in Projects view

The variable's default values are displayed in the **Projects** tree, after the "=" character next to the object.

The transaction is implemented and can be tested in the Studio. To do so, the first step is to open the connector editor by double-clicking on the *HTML connector* in the **Projects** view (if not already open). The internal browser connects to `http://www.google.com/` as seen in **Design** tab of the editor:

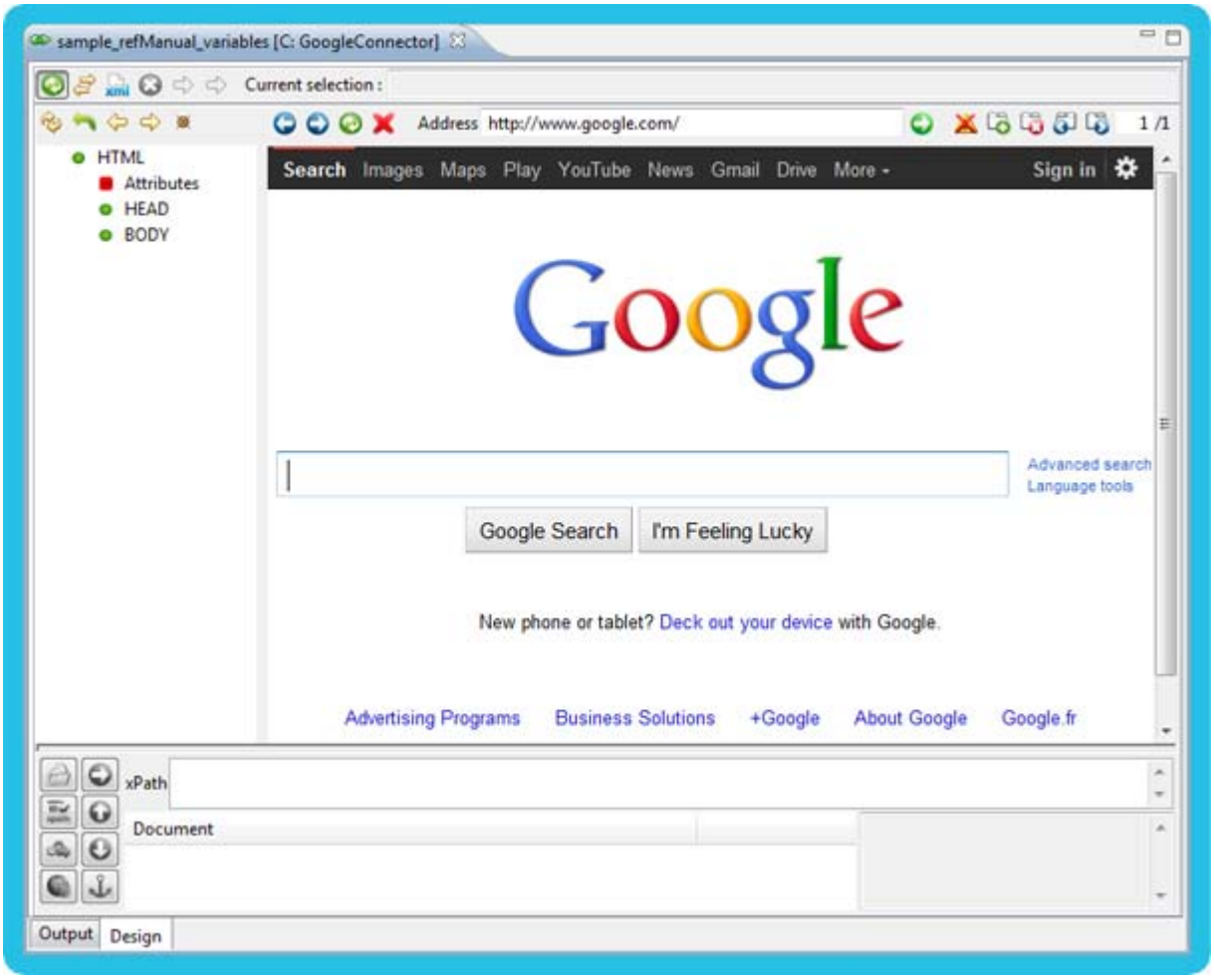


Figure 2 - 69: HTTP multi-valued variable - Connector editor

Now execute the `multiSearchGoogleHTTP` transaction by pressing F5 key on the transaction object in the **Projects** view. The internal browser connects to `http://www.google.com/search?q=convertigo&q=cliplet` as seen in **Design** tab of the editor:

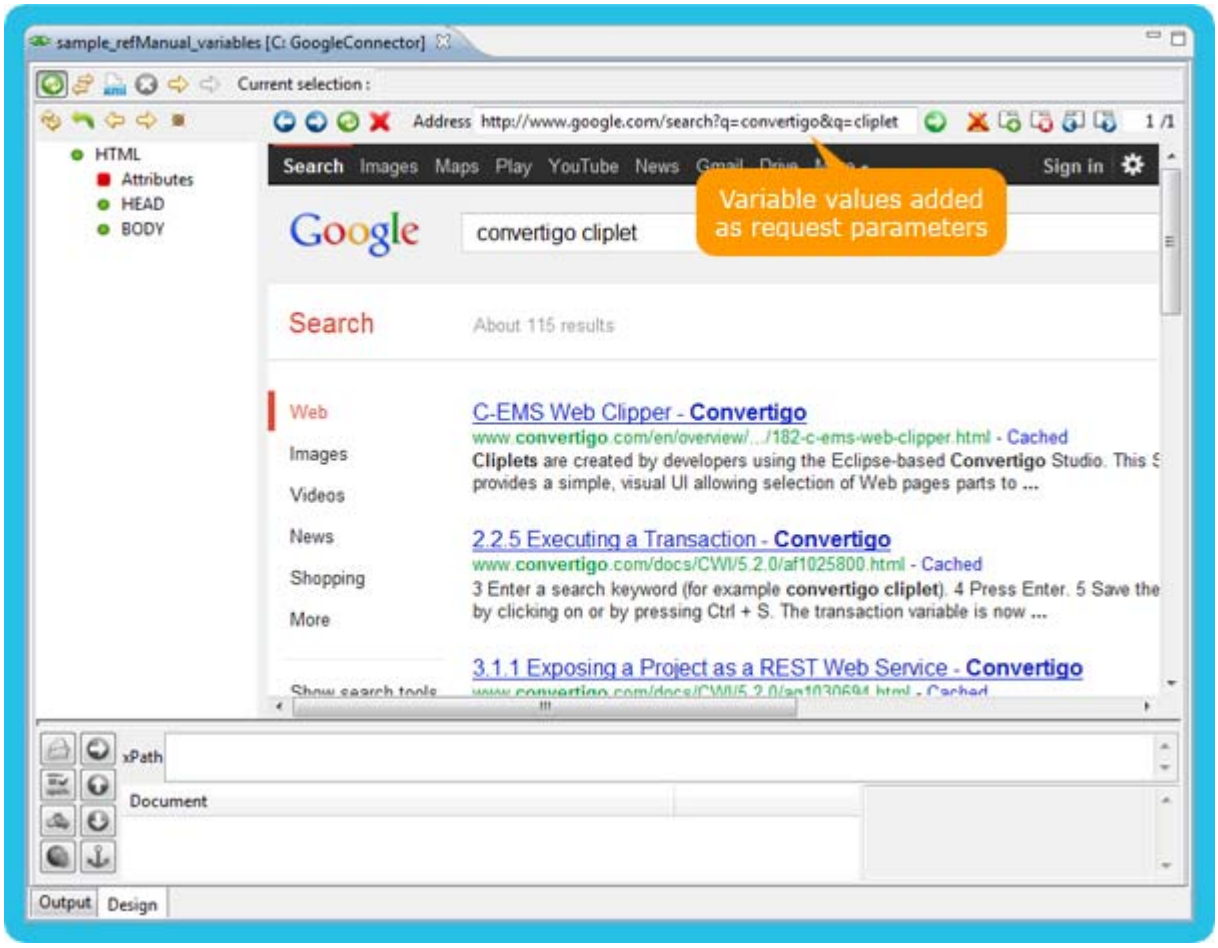


Figure 2 - 70: HTTP multi-valued variable - Connector editor

WHAT HAPPENED?

The connector has built the request URL by concatenating the server address, the transaction subpath and a query string. This query string was formed with the variable details.

Each value of the variable which **HTTP method** property set to `GET` is added to the query string as a parameter:

- which name is set by the variable's **HTTP name** property,
- which value is the variable's value.

In this case, the parameter name is `q` and its values are the transaction's `keywords` default values "`convertigo`" and "`cliplet`".

Notes:

- If the **HTTP name** property is empty, the parameters are not added, thus not sent.
- If the **HTTP method** property is set to `POST`, the parameters are added to the body of request (POST data) rather than to the query string.

STATEMENT VARIABLES

HTTP SINGLE-VALUED VARIABLE (STATEMENT VARIABLES) 

OBJECT DESCRIPTION

Defines a single-valued variable for an HTTP-based statement.

An *HTTP single-valued variable* declares a variable which accepts a unique value to an HTTP-based statement.

This variable is dedicated to HTTP-based statements only:

- *HTTP request* statement.

This variable object allows defining an HTTP request parameter through the **HTTP name** and **HTTP method** properties.

It can define a default value, specified in the **Default value** property, that is used as HTTP parameter value if no value is found for this variable.

At runtime, the HTTP parameter value is calculated by Convertigo through the following steps:

- if a variable of the same name exists in the JavaScript scope of current context, the JavaScript value of the variable is chosen,
- if no JavaScript variable is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives a value, the default value defined in **Default value** property is used,
- if no default value is specified, the variable is not sent in the HTTP request.

Note: In Convertigo Studio, when an *HTTP single-valued variable* is created in an *HTTP request* statement, it can be easily replaced by an *HTTP multi-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **MultiValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.
HTTP method	String	standard	Defines the HTTP method to use for this variable. This property allows choosing which HTTP method has to be used to send the variable in the HTTP request. The following values are available: • `GET`: the variable is added to the query string as follows: `?<initial_query_string>&<variable_name>=<variable_value>`. • `POST`: the variable is added to the post data sent in the HTTP request as a standard `POST FORM`. `GET` and `POST` method variables can be mixed in a same statement. If at least one `POST` variable is used, the statement's HTTP verb is overridden to `POST` by Convertigo.
HTTP name	String	standard	Defines the HTTP parameter name. This property allows defining the name of the HTTP parameter sent in the request by Convertigo executing the statement. The HTTP parameter named by this property is added to the query string or to post data, depending on the **HTTP method** property value. If the HTTP-based statement emulates a form submission, this property can match the `name` attribute of an HTML input field. If the HTTP-based statement emulates a resource access by URL, this property can match one of the variables names from the query string (between "&" and "=" characters).

Property	Type	Category	Description
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.

EXAMPLES

This example is based on a connector, reaching Google website, defined in a project named `sample_refManual_variables`.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **Variables examples** in the **New Project** wizard.*

Let's consider an *HTML transaction*, named `searchYahoo`, similar to `searchGoogle` transaction developed in the context of the "*Starting with Convertigo Web Integrator*" tutorial, but performing the research on Yahoo website instead of Google.

It declares an *HTTP single-value variable*, named `keyword`, with a **Default value** property set to "`Convertigo`". For more information about *HTML transaction* or *HTTP single-valued variable* object, see "*HTML transaction*" and "*HTTP single-valued variable*" documentation and example.

The `searchYahoo` transaction appears as follows in the **Projects** view of the Convertigo Studio:

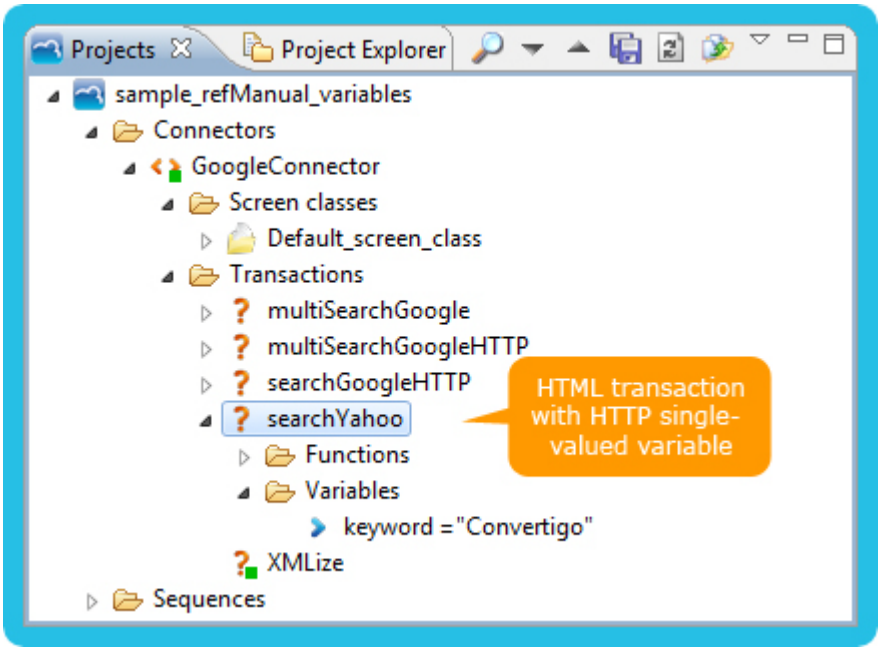


Figure 2 - 71: HTTP single-valued variable - HTML transaction with HTTP variable in Projects view

The `searchYahoo` transaction is created with its **Subpath** property left empty, so it starts executing its statements after the connector has established the connection to target website.

Being in a connector reaching Google website, the transaction has to redirect to Yahoo website when Google search page is loaded. Then, it makes the research on the given keyword.

The Yahoo request URL for a research is:

`http://fr.search.yahoo.com/search?p=<keyword>`

where p is the keyword HTTP parameter.

To perform the redirection, an *HTTP request* statement is implemented on the *Entry handler* matching Google search page. The *HTTP request* statement may declare variables, in order to send them as request parameters. For more information about the configuration of the *HTTP request* statement, see "*HTTP request*" statement documentation and examples.

This statement declares an *HTTP single-valued variable*, named `keyword`, created with the followings parameters:

```
HTTP single-valued variable [
   Description=Keyword to send to Yahoo
   Default value="Orsay"
   HTTP method=GET
   HTTP name=p
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

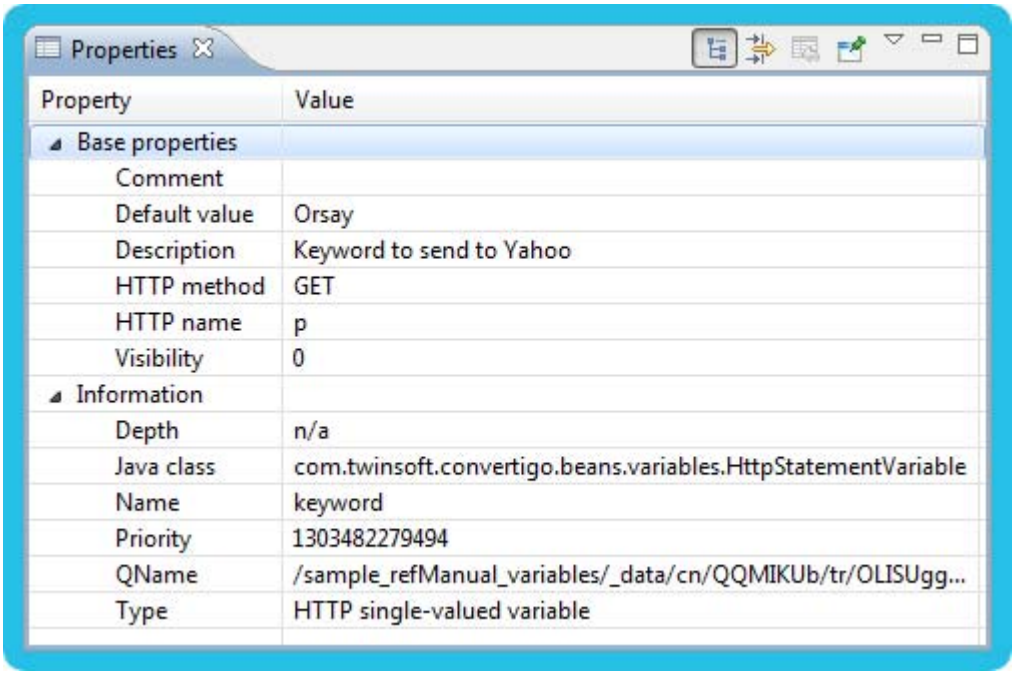


Figure 2 - 72: HTTP single-valued variable - Configuration example with default value

At runtime, the variable value is retrieved from the JavaScript scope, if a variable of the same name exists in scope of current context. In this case, the transaction variable has the same name as the statement variable. As transaction variables are inserted in scope at the begining of the transaction execution, the statement variable will find its value in scope.

The *HTTP request* statement is created in the **Statements** folder of the handler and appears, with its variable, as follows in the **Projects** view:

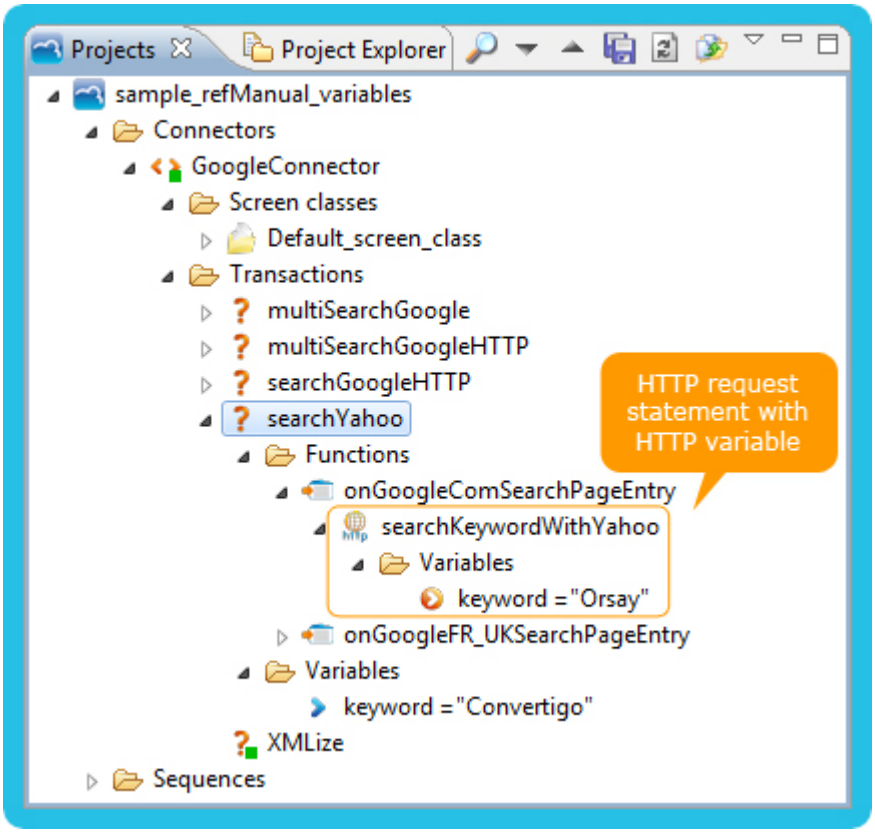


Figure 2 - 73: HTTP single-valued variable - HTTP request statement with HTTP variable in Projects view

The variable's default value is displayed in the **Projects** tree, after the "=" character next to the object.

The transaction is implemented and can be tested in the Studio. To do so, the first step is to open the connector editor by double-clicking on the *HTML connector* in the **Projects** view (if not already open). The internal browser connects to `http://www.google.com/` as seen in **Design** tab of the editor:

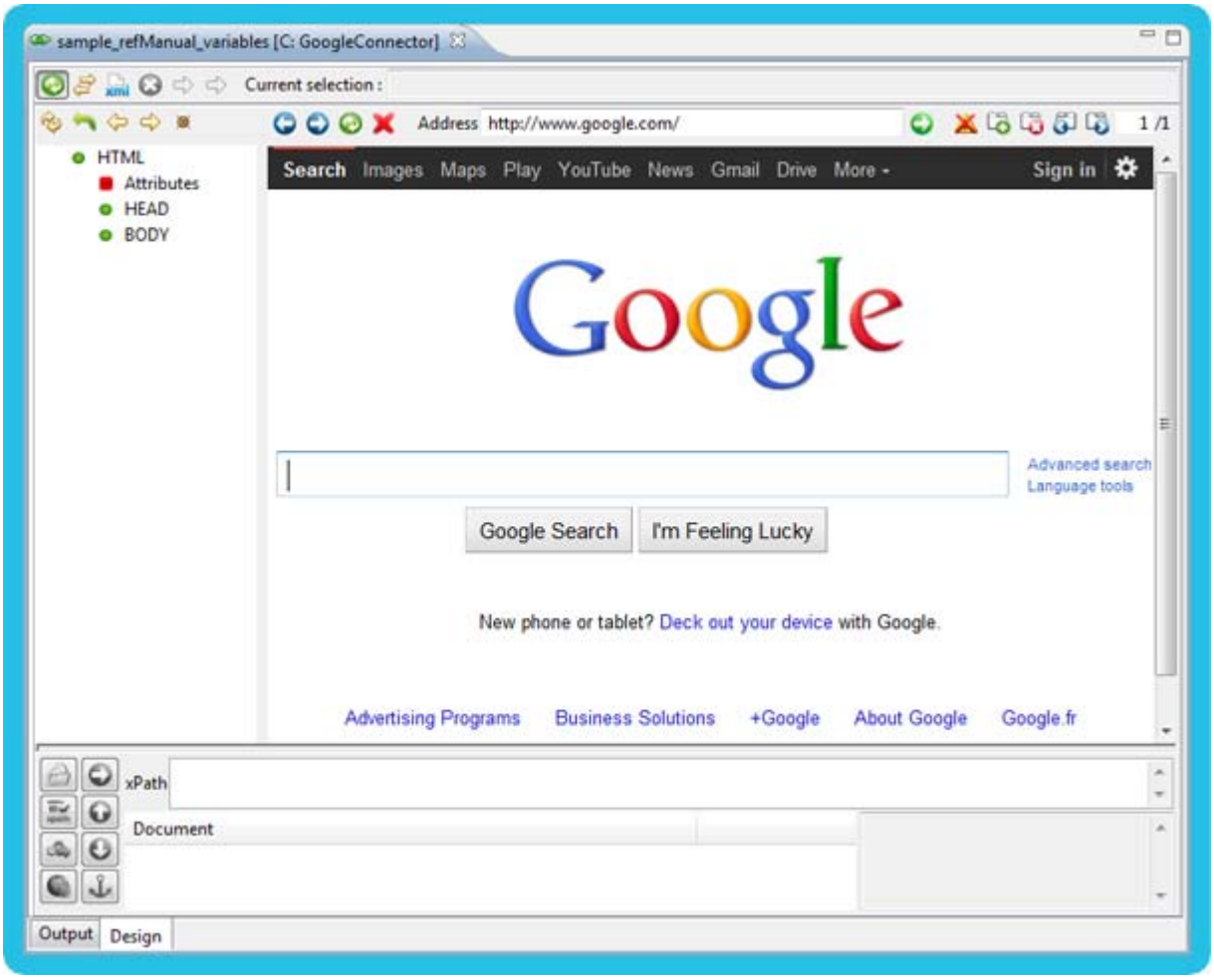


Figure 2 - 74: HTTP single-valued variable - Connector editor

Now execute the `searchYahoo` transaction by pressing F5 key on the transaction object in the **Projects** view. The internal browser connects to `http://fr.search.yahoo.com/search?p=Convertigo` as seen in **Design** tab of the editor:

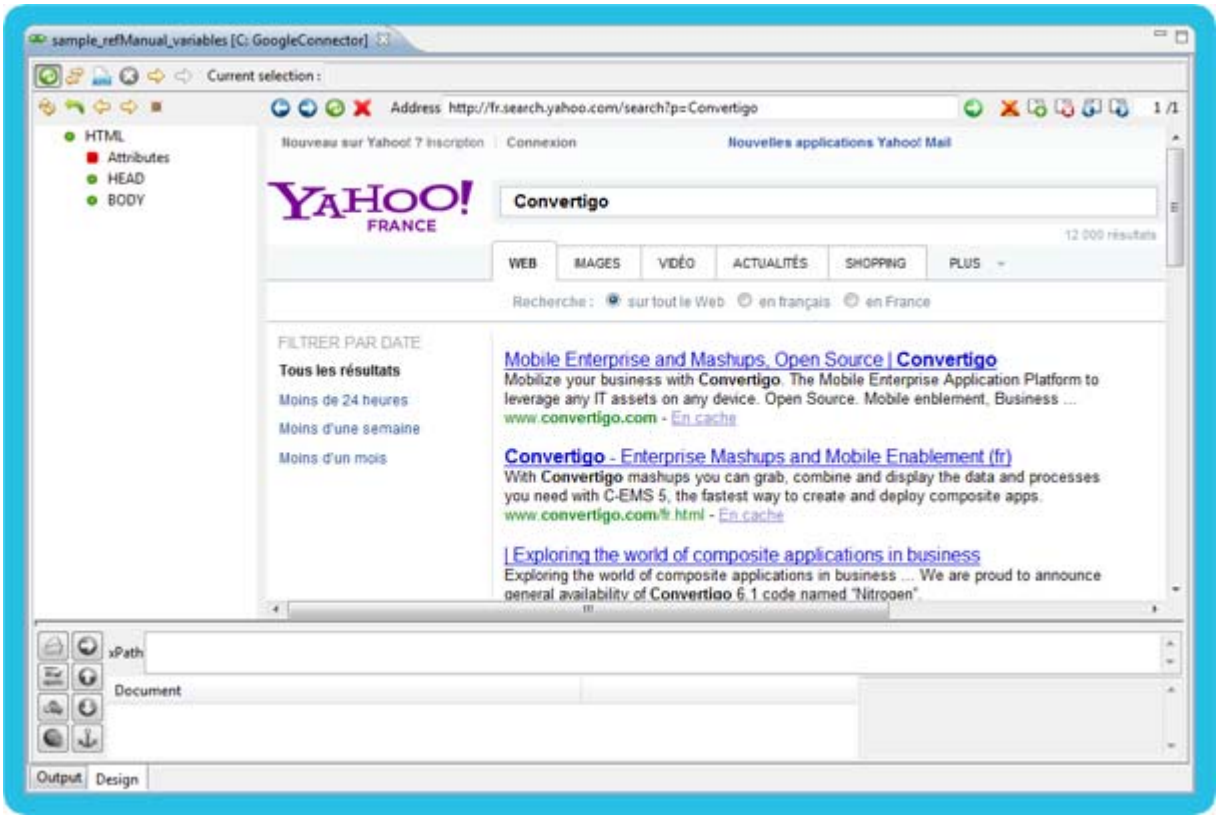


Figure 2 - 75: HTTP single-valued variable - Connector editor after transaction execution

WHAT HAPPENED?

At the start of the transaction, its `keyword` variable was added to the JavaScript scope.

The connector built the request URL by concatenating the server address, the transaction **Subpath** (empty) and an empty query string because the `keyword` variable hasn't specified any parameter name through its **HTTP name** property. Thus, it simply requested `http://www.google.com/`.

The Google search page screen class was detected thanks to its criteria and the corresponding entry handler was executed.

The *HTTP request* statement built the request URL by concatenating its **Host** value, **URI** value and a query string. This query string was formed from statement's variables details.

Each variable which **HTTP method** property is set to `GET` is added to the query string as a parameter:

- which name is set by the variable's **HTTP name** property,
- which value is the variable's value.

In this case, the parameter name is p and its value is the transaction's `keyword` default value "`Convertigo`". This value has overriden the statement's variable default value. If the transaction didn't declare any variable, the statement wouldn't have found a variable value in the JavaScript scope and it would have used its variable default value "`Orsay`".

Notes:

- If the **HTTP name** property is empty, the parameter is not added, thus not sent.

- If the **HTTP method** property is set to POST, the parameter is added to the body of request (POST data) rather than to the query string.

HTTP MULTI-VALUED VARIABLE (STATEMENT VARIABLES) 

OBJECT DESCRIPTION

Defines a multi-valued variable for an HTTP-based statement.

An *HTTP multi-valued variable* declares a variable which accepts one or more values to an HTTP-based statement.

This variable is dedicated to HTTP-based statements only:

- *HTTP request* statement.

This variable object allows defining a list of HTTP request parameters through the **HTTP name** and **HTTP method** properties.

It can define a default list of value(s), specified in the **Default value** property, that are used as HTTP parameters values if no value is found for this variable.

At runtime, the HTTP parameters values are calculated by Convertigo through the following steps:

- if a variable of the same name exists in the JavaScript scope of current context, the JavaScript value of the variable is chosen (this JavaScript variable should be an array of values),
- if no JavaScript variable is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives a value, the default values defined in **Default value** property are used,
- if no default value is specified, the variable is not sent in the HTTP request.

Note: In Convertigo Studio, when an *HTTP multi-valued variable* is created in an *HTTP request* statement, it can be easily replaced by an *HTTP single-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **SingleValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.
HTTP method	String	standard	Defines the HTTP method to use for this variable. This property allows choosing which HTTP method has to be used to send the variable in the HTTP request. The following values are available: • `GET`: the variable is added to the query string as follows: `?<initial_query_string>&<variable_name>=<variable_value>`. • `POST`: the variable is added to the post data sent in the HTTP request as a standard `POST FORM`. `GET` and `POST` method variables can be mixed in a same statement. If at least one `POST` variable is used, the statement's HTTP verb is overridden to `POST` by Convertigo.
HTTP name	String	standard	Defines the HTTP parameter name. This property allows defining the name of the HTTP parameter sent in the request by Convertigo executing the statement. The HTTP parameter named by this property is added to the query string or to post data, depending on the **HTTP method** property value. If the HTTP-based statement emulates a form submission, this property can match the `name` attribute of an HTML input field. If the HTTP-based statement emulates a resource access by URL, this property can match one of the variables names from the query string (between "&" and "=" characters).

Property	Type	Category	Description
Soap array	boolean	standard	Defines if the multi-valued variable should be seen as a Soap Array of a occurrence of variables. In the case of transaction or sequence defined as a public SOAP method, this property allows to specify of the current multi-valued variable has to be seen in SOAP envelope as a Soap Array with multiple values inside it or as an occurrence of identical variables.
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.

STEP VARIABLES

CALL SINGLE-VALUED VARIABLE 

OBJECT DESCRIPTION

Defines a single-valued variable for a step.

A *Call single-valued variable* is used to send a single-valued `input` variable to a transaction/sequence targeted by a *Call Transaction*/*Call Sequence* step.

It can define a default value, specified in the **Default value** property, that is used as parameter value if no value is found for this variable.

At runtime, the variable value is calculated by Convertigo through the following steps:

- if the **Source** property is set, the variable value is the source result (see **Source** property documentation),
- if no source is set, the JavaScript value of the variable is chosen, if a variable of the same name exists in the JavaScript scope of current context,
- if no JavaScript variable is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives a value, the default value set in the **Default value** property is used,
- if no default value is specified, the variable is not sent to the target transaction/sequence.

Note: In Convertigo Studio, when a *Call single-valued variable* is created in a *Call Transaction*/*Call Sequence* step, it can be easily replaced by a *Call multi-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **MultiValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.
Source	XMLVector	expert	Defines the source to use as variable value. This property allows defining the variable value as a source from a previous step. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and the variable takes for value the XML node value resulting from this execution (the variable value will be its text content). If the XPath doesn't match or if the source is left blank, the variable value is calculated as explained in the main description of this object.

Property	Type	Category	Description
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.

EXAMPLES

The *Call Transaction* and *Call Sequence* steps enable to request any transaction or sequence execution inside a given sequence. Those steps may provide input variables to the target transaction or sequence, matching variables defined for the target transaction or sequence. For more information about *Call Transaction* or *Call Sequence* steps, see "*Call Transaction*" and "*Call Sequence*" steps documentation and examples.

The first example shows *Call Transaction* steps variables.

Example 1

Let's consider the sequences developed in the `sample_documentation_CMS` project in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial. These sequences contain *Call Transaction* steps declaring *Call single-valued variables*.

*You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Documentation samples** > **Mashup sequencing** in the **New Project** wizard.*
*Associated projects can be opened by selecting **Convertigo Samples and Demos** > **Documentation samples** > **Legacy integration** and **Web integration** in the **New Project** wizard.*

The `GetXMLData` sequence includes two *Call Transaction* steps:

- `Call_Transaction_GetArticleData` step, launching `GetArticleData` *Javelin transaction* from `sample_documentation_CLI` project,
- `Call_Transaction_SearchGoogle` step, launching `SearchGoogleWithLimit`

HTML transaction from `sample_documentation_CWI` project.

Each of the target transactions declares variables. The steps have to send these variables to the call to the transaction.

To do so, they declare *Call variables*, imported from respective target transaction, that are used as input parameters by called transactions:

- a *Call single-valued variable*, named `article_no`, with the following parameters:

```
Call single-valued variable [
  Default value=""
  Source=[1255530858989,  .]  ==>  reference  to  the  preceding
<article_num> Element step
]
```

- a *Call single-valued variable*, named `keyword`, with the following parameters:

```
Call single-valued variable [
  Default value=""
  Source=[1255534601721,  ./name]  ==>  reference  to  the  preceding
IteratorOnEachRow step
]
```

- a *Call single-valued variable*, named `maxPages`, with the following parameters:

```
Call single-valued variable [
  Default value="2"
  Source=[]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio (here, `keyword` and `maxPages` variables):

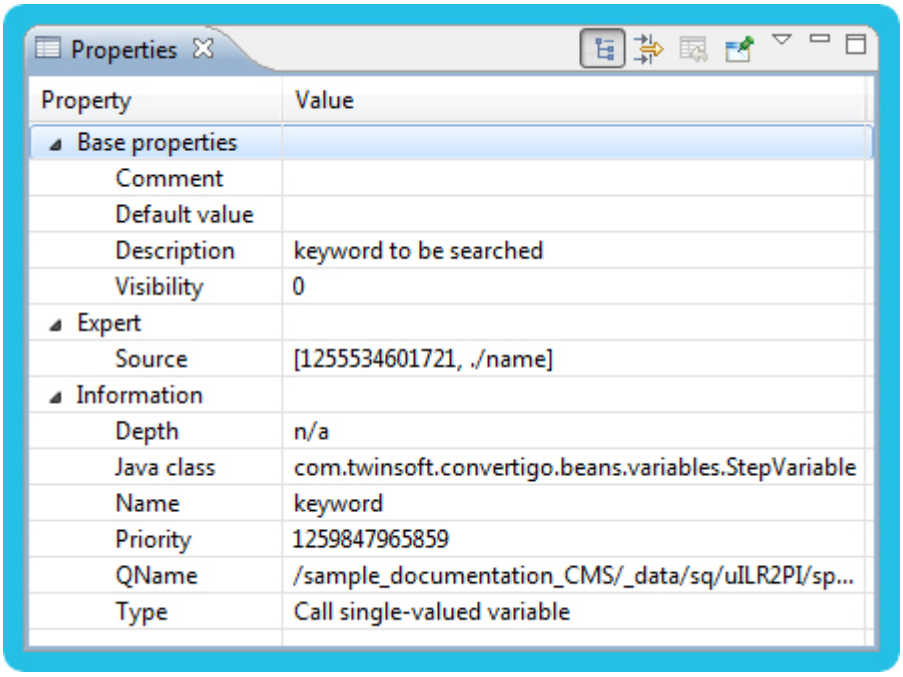


Figure 2 - 76: Call single-valued variable - Configuration example

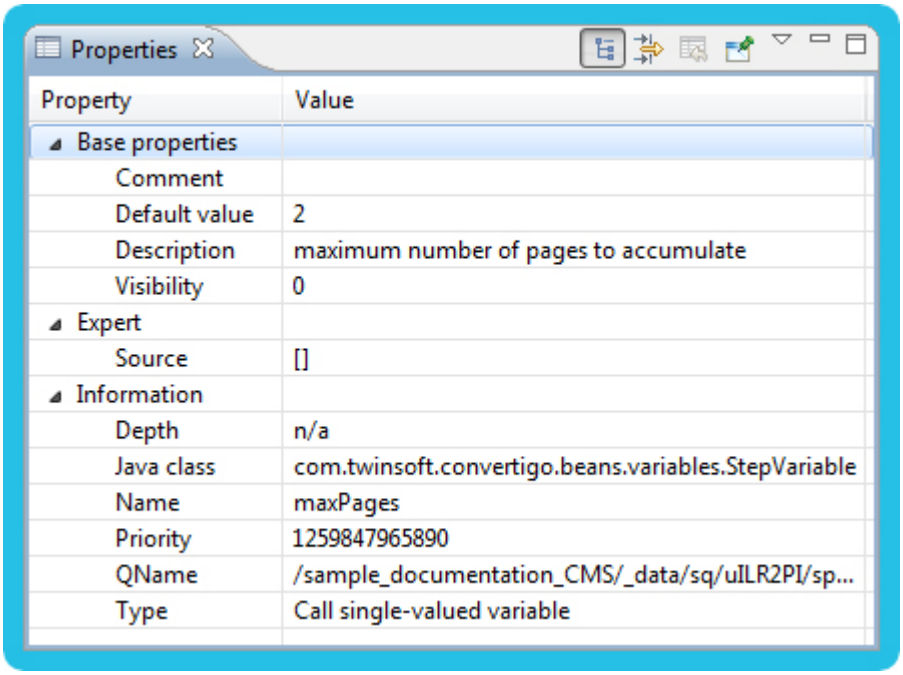


Figure 2 - 77: Call single-valued variable - Configuration example

The **Source** property of two of these variables is set so that the variables are sourced from the relevant previous steps. These properties are edited in the **Step Source** editor.

- The **Source** property of `article_no` variable is set from the previous `<article_num>` *jElement* step, with the following configuration:

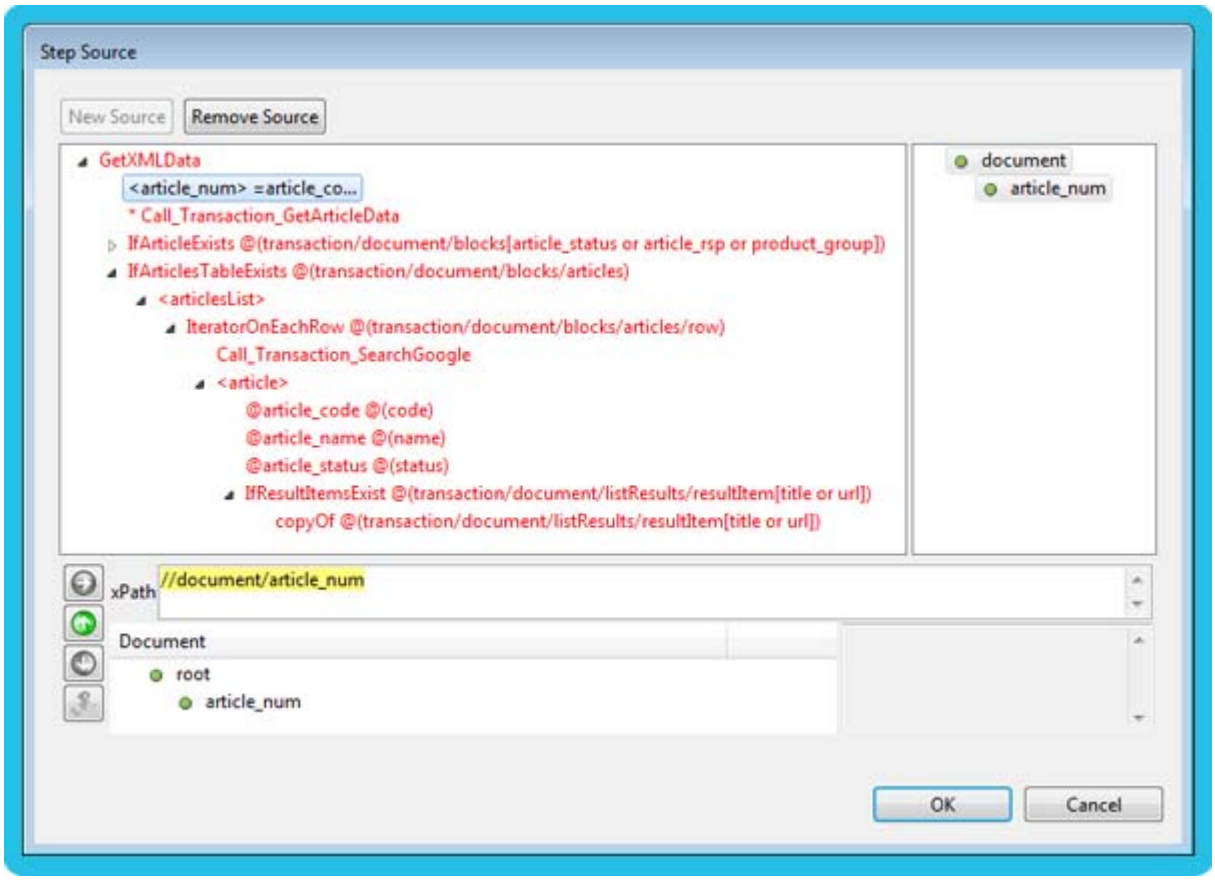


Figure 2 - 78: Call single-valued variable - Source of the article_no variable

- The **Source** property of `keyword` variable is set from the previous `IteratorOnEachRow` *Iterator* step, with the following configuration:

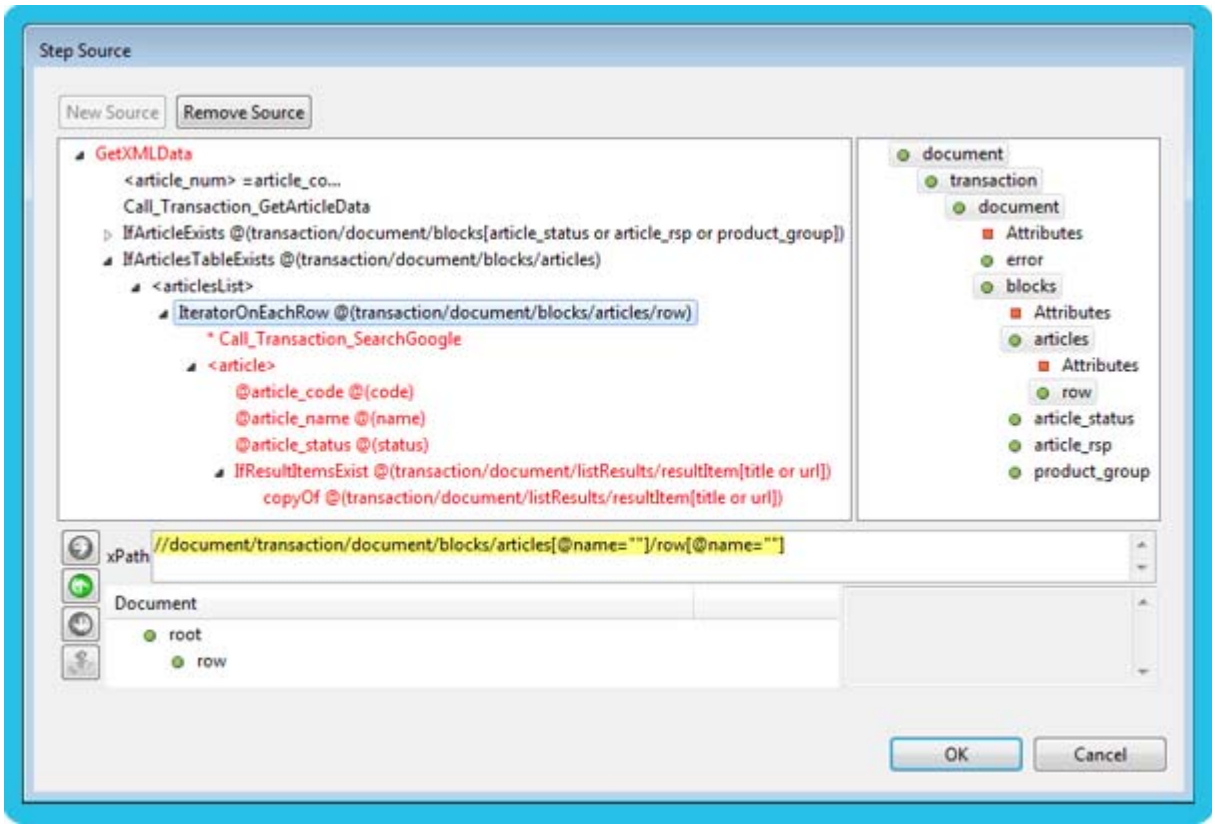


Figure 2 - 79: Call single-valued variable - Source of the keyword variable

These *Call single-valued variables* are created in the **Variables** folders of the *Call Transaction* steps, and appear as follows in the **Projects** view of the Convertigo Studio:

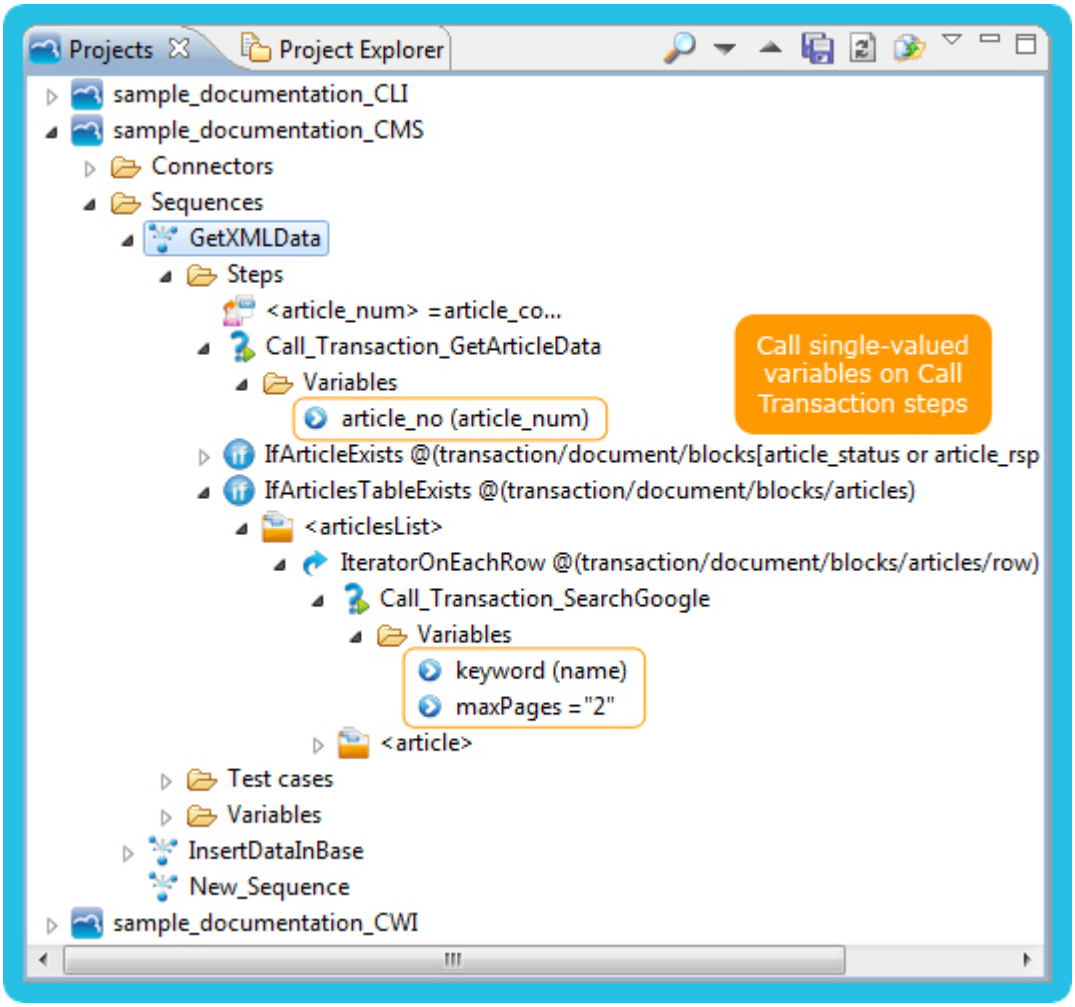


Figure 2 - 80: Call single-valued variable - Call Transaction steps with Call variables in Projects view

At runtime, variables are provided by the *Call Transaction* steps to the executed transactions. Values obtained from sources are sent throughout the execution depending on the previous steps current XML.

The second and third examples show *Call Sequence* steps variables. The following examples are in continuity with those given for *Requestable single-valued variable* and *Requestable multi-valued variable*. Please refer to these objects examples for more information before proceeding.

Example 2

We refer here to the `userLogin` and the `userLoginWithDefault` sequences of the `sample_refManual_variables` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

A *Generic sequence*, named `testLogin`, is created in order to call both sequences and retrieve their XML response. To do so, this sequence includes two *Call Sequence* steps:

- `Call_userLogin` step launches the `userLogin` sequence,
- `Call_userLoginWithDefault` step launches the `userLoginWithDefault` sequence.

This sequence is created in the **Sequences** folder of the project and appears, with the two steps, as follows in the **Projects** view:

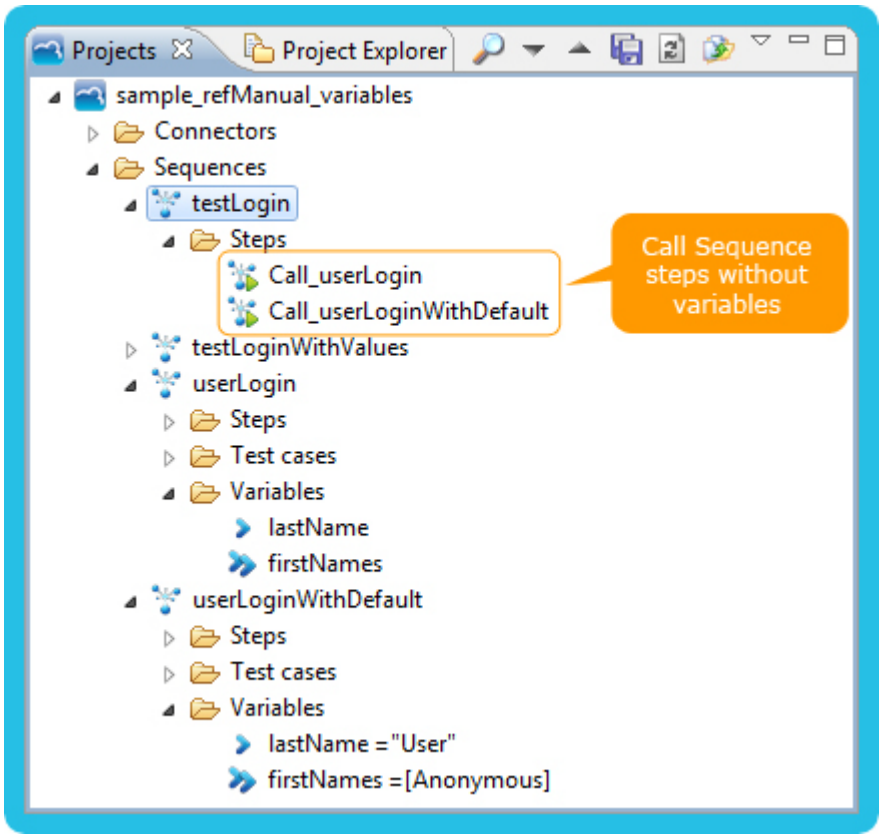


Figure 2 - 81: Call single-valued and multi-valued variables - Sequence and Call Sequence steps in Projects view

The `Call_userLogin` and the `Call_userLoginWithDefault` steps declare no variable, and their **Output** property is set to `true` in order to add the returned XML response to the `testLogin` sequence response.

At runtime, as no variable is provided by the *Call Sequence* steps to the called sequences, they are executed with no variable provided in entry.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

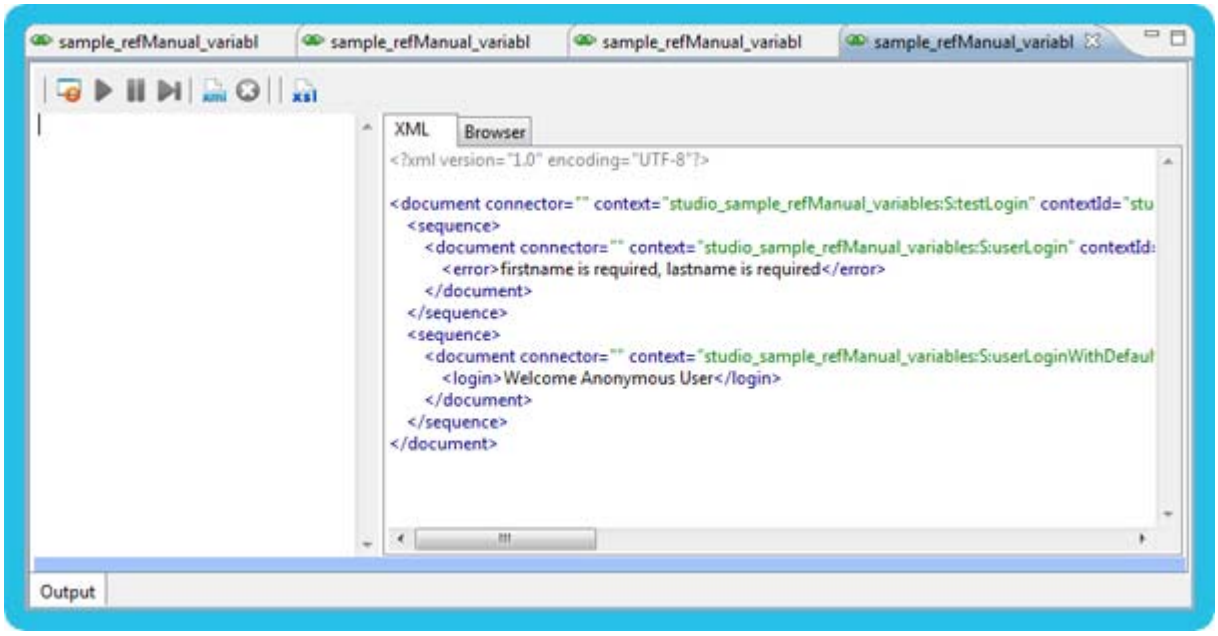


Figure 2 - 82: Call single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED ?

The `lastName` and `firstNames` variables were not provided by the *Call Sequence* steps, thus each target sequence used its variables default values.

The `loginUser` sequence returned an error message as it requires values to be provided.

The `loginUserWithDefault` returned a default `<login>` message as it handles a default case if values are not provided.

Example 3

This example is the same as the previous one except that the *Call Sequence* steps declare variables.

A *Generic sequence*, named `testLoginWithValues`, is created with two *Call Sequence* steps and two *Element* steps:

- `Call_userLogin` *Call Sequence* step launches the `userLogin` sequence,
- `<firstname>` *Element* step writes an XML element in result XML with value "`Node`",
- `<lastname>` *Element* step writes an XML element in result XML with value "`Value`",
- `Call_userLoginWithDefault` *Call Sequence* step launches the `userLoginWithDefault` sequence.

The `Call_userLogin` step declares two variables imported from the sequence and for which default values have been modified as follows:

- a *Call single-valued variable*, named `lastName`, with the following parameters:

```
Call single-valued variable [
   Default value="Value"
   Source=[]
]
```

- a *Call multi-valued variable*, named `firstNames`, with the following parameters:

```
Call multi-valued variable [
   Default value=["Fixed"]
   Source=[]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

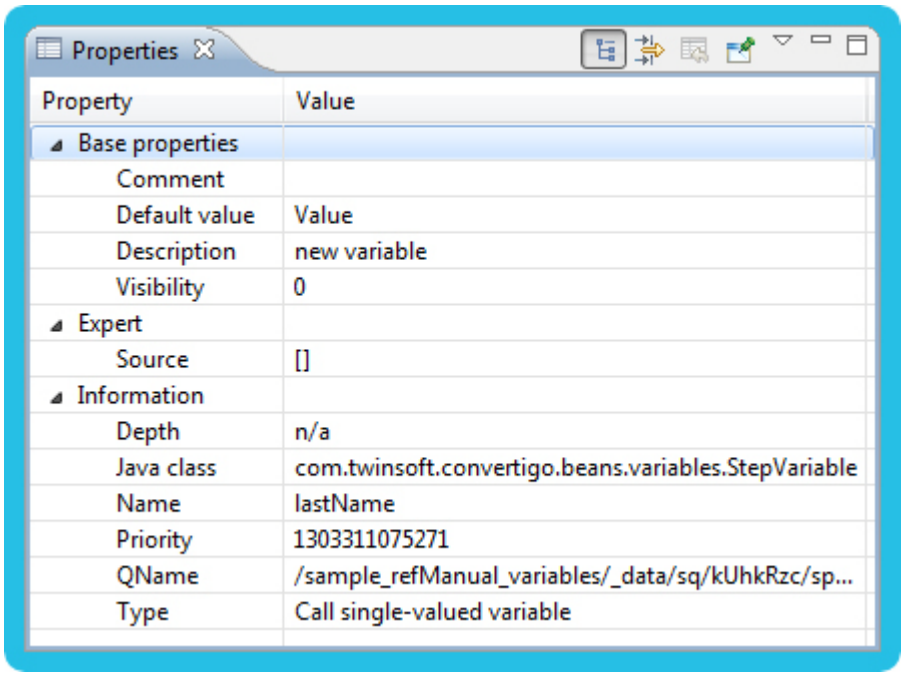


Figure 2 - 83: Call single-valued variable - Configuration example with default value

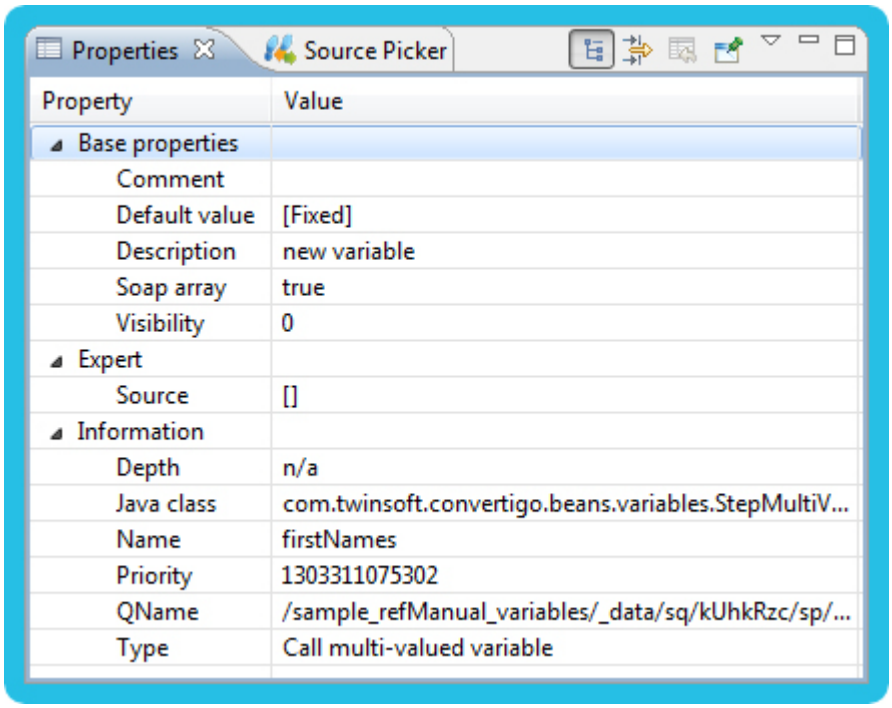


Figure 2 - 84: Call multi-valued variable - Configuration example with default value

The **Default value** property of the *Call multi-valued variable* is edited in the **Array** editor:

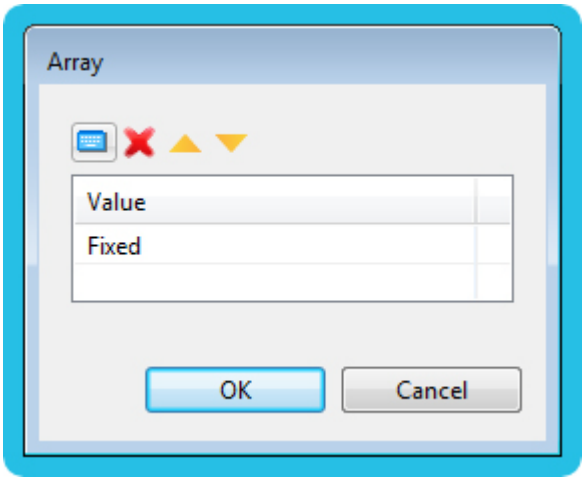


Figure 2 - 85: Call multi-valued variable - Default value property in Array editor

The `Call_userLoginWithDefault` step declares two variables imported from the sequence and for which **Source** property is set as follows:

- a *Call single-valued variable*, named `lastName`, with the following parameters:

```
Call single-valued variable [
   Default value=null
   Source=[1303311539328, .] ==> reference to the preceding <lastname>
step
]
```

- a *Call multi-valued variable*, named `firstNames`, with the following parameters:

```
Call multi-valued variable [
   Default value=null
   Source=[1303311478425,   .]   ==>   reference   to   the   preceding
<firstname> step
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

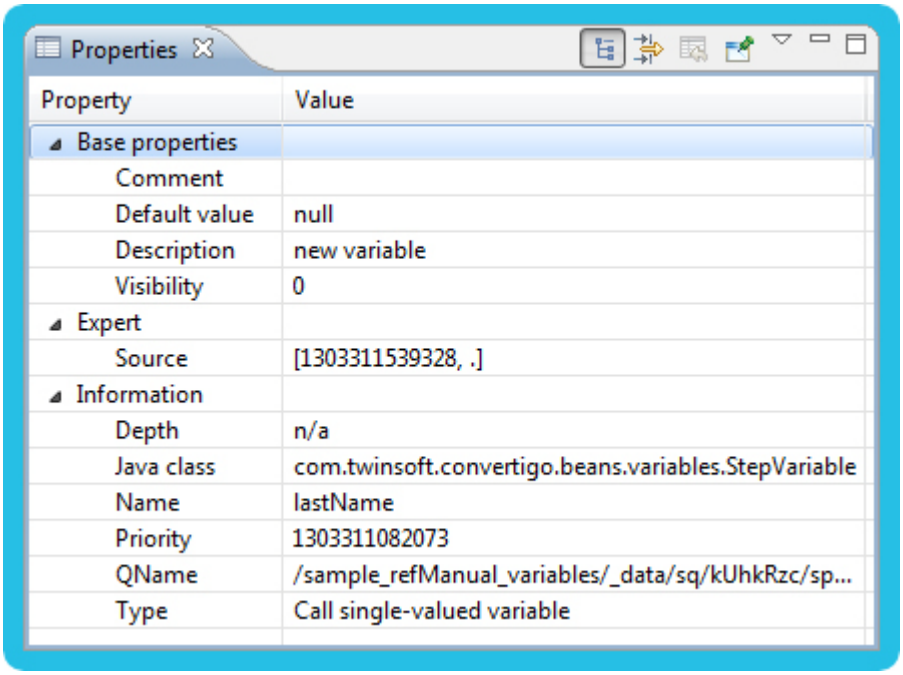


Figure 2 - 86: Call single-valued variable - Configuration example with Source

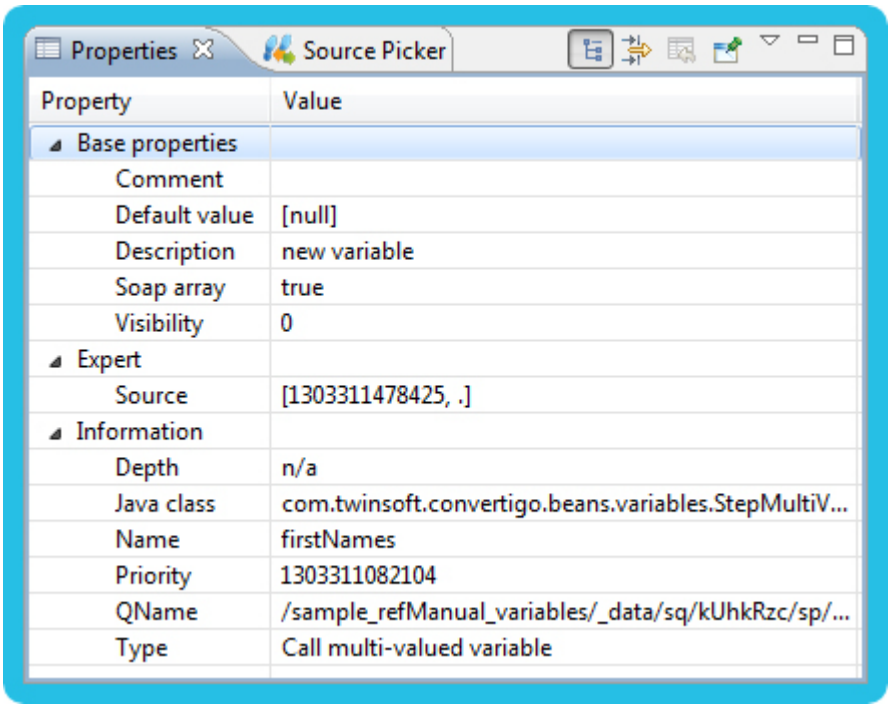


Figure 2 - 87: Call multi-valued variable - Configuration example with Source

The **Source** property of these two variables is set so that the variables are sourced from the relevant previous steps. These properties are edited in the **Step Source** editor with the following configuration (here, the `firstNames` variable **Source** property):

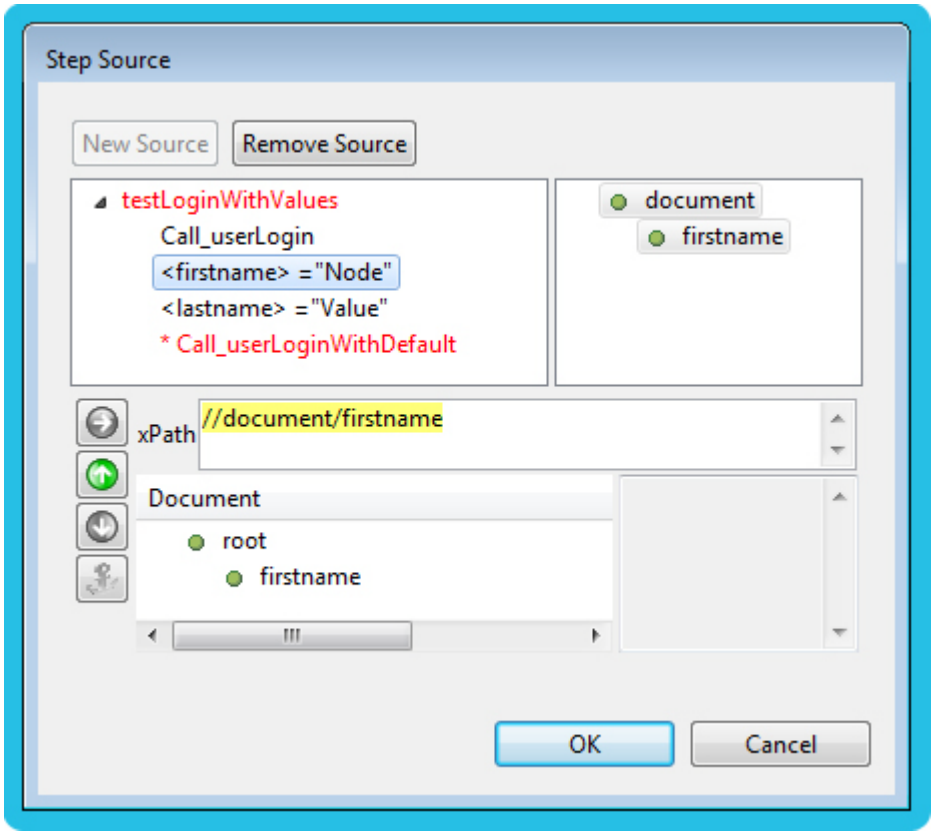


Figure 2 - 88: Call multi-valued variable - Source of the firstNames variable

The sequence is created in the **Sequences** folder of the project and appears, with the steps, as follows in the **Projects** view:

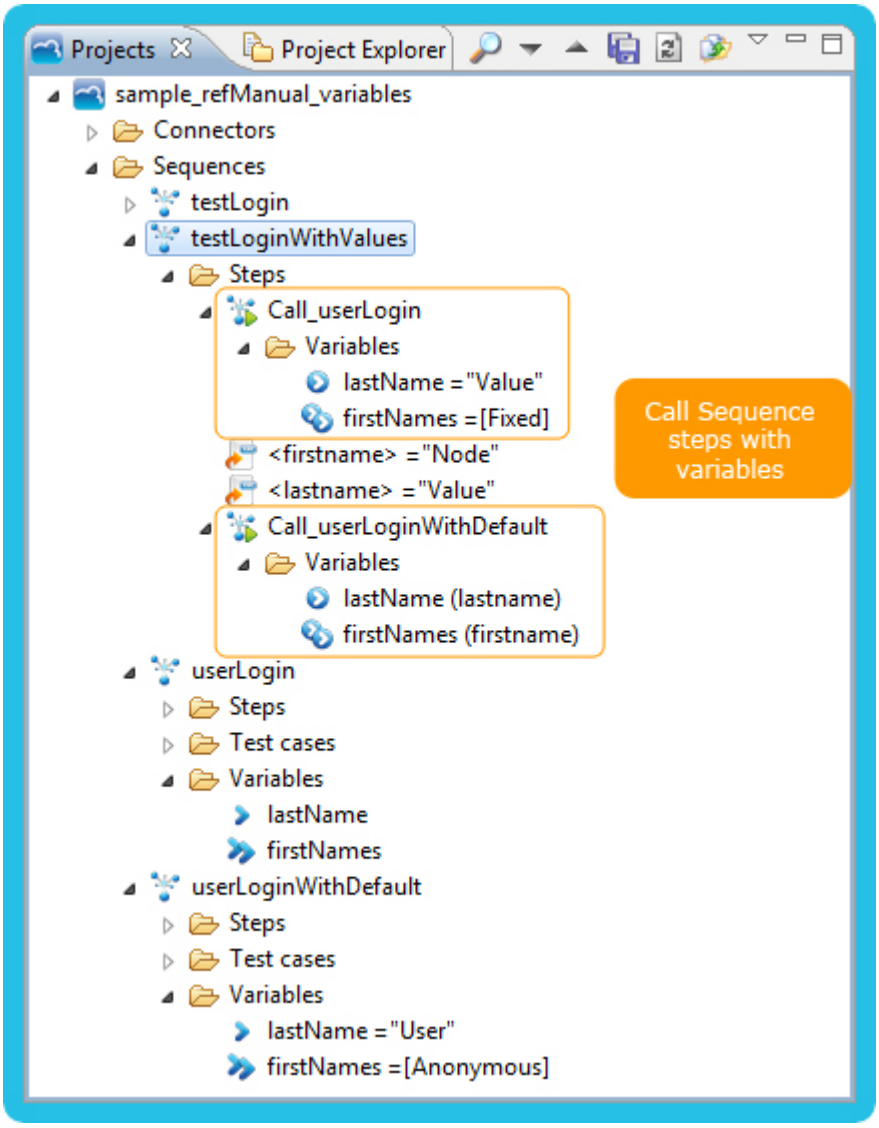


Figure 2 - 89: Call single-valued and multi-valued variables - Call Sequence steps with Call variables in Projects view

Both *Call Sequence* steps **Output** property is set to `true` in order to add the returned XML responses to the `testLogin` sequence XML result.

At runtime, as variables are provided by the *Call Sequence* steps to the executed sequences, they are executed with these variables values.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

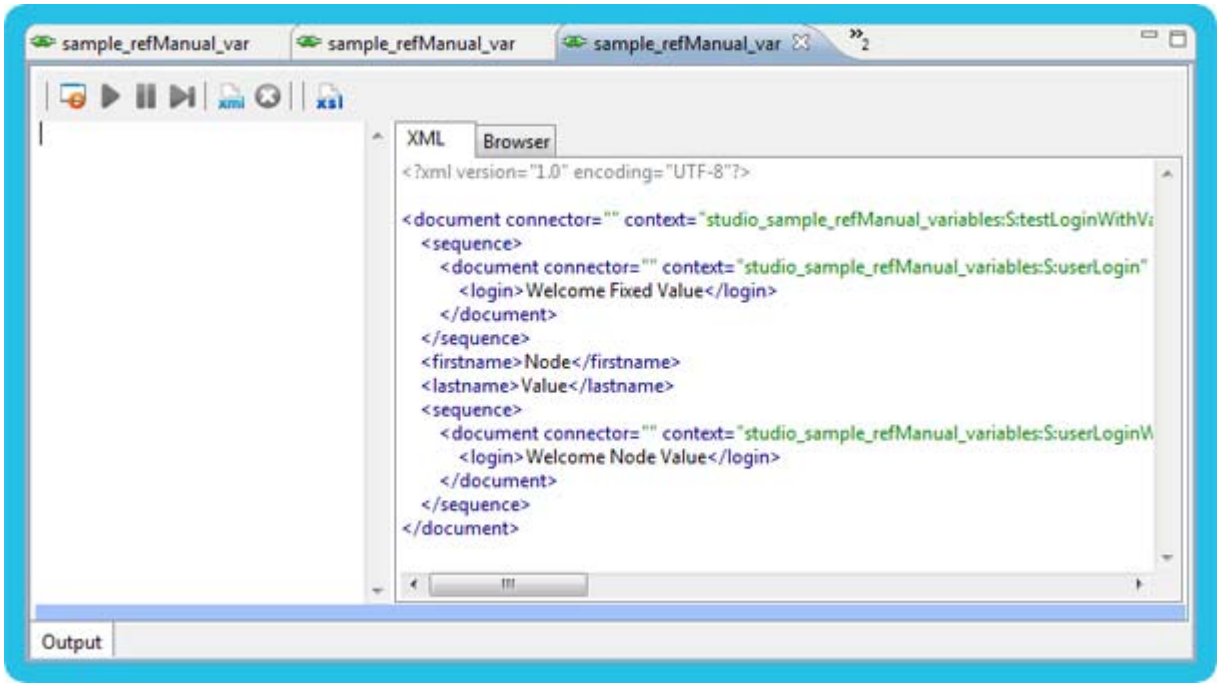


Figure 2 - 90: Call single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED?

The `lastName` and `firstNames` variables were provided by the `Call Sequence` steps, thus each target sequence variables was overridden by the values sent by the steps.

Both of the `loginUser` and `loginUserWithDefault` sequences returned a successful login message.

CALL MULTI-VALUED VARIABLE 

OBJECT DESCRIPTION

Defines a multi-valued variable for a step.

A *Call multi-valued variable* is used to send a multi-valued `input` variable to a transaction/sequence targeted by a *Call Transaction*/*Call Sequence* step.

It can define a list of default values, specified in the **Default value** property, that is used as parameter value if no value is found for this variable.

At runtime, the variable value is calculated by Convertigo through the following steps:

- if the **Source** property is set, the variable value is the source result (see **Source** property documentation),
- if no source is set, the JavaScript value of the variable is chosen, if a variable of the same name exists in the JavaScript scope of current context (this JavaScript variable should be an array of values),
- if no JavaScript variable is defined, the context value of the variable is chosen, if a variable of the same name is stored in current context,
- if none of the previous methods gives a value, the default list of values set in the **Default value** property is used,
- if no default value is specified, the variable is not sent to the target transaction/sequence.

Note: In Convertigo Studio, when a *Call multi-valued variable* is created in a *Call Transaction*/*Call Sequence* step, it can be easily replaced by a *Call single-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **SingleValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.
Soap array	boolean	standard	Defines if the multi-valued variable should be seen as a Soap Array of a occurrence of variables. In the case of transaction or sequence defined as a public SOAP method, this property allows to specify of the current multi-valued variable has to be seen in SOAP envelope as a Soap Array with multiple values inside it or as an occurrence of identical variables.
Source	XMLVector	expert	Defines the source to use as variable value. This property allows defining the variable value as a source from a previous step. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and the variable takes for value the XML NodeList resulting from this execution. The variable value will be an array of the text contents. In the target sequence/transaction, it is the standard Convertigo `Array of Strings` variable. If the XPath doesn't match or if the source is left blank, the variable value is calculated as explained in the main description of this object.

Property	Type	Category	Description
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.

EXAMPLES

The *Call Transaction* and *Call Sequence* steps enable to request any transaction or sequence execution inside a given sequence. Those steps may provide input variables to the target transaction or sequence, matching variables defined for the target transaction or sequence. For more information about *Call Transaction* or *Call Sequence* steps, see "*Call Transaction*" and "*Call Sequence*" steps documention and examples.

These examples show *Call Sequence* steps variables. The following examples are in continuity with those given for *Requestable single-valued variable* and *Requestable multi-valued variable*. Please refer to these objects examples for more information before proceeding.

Example 1

We refer here to the `userLogin` and the `userLoginWithDefault` sequences of the `sample_refManual_variables` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

A *Generic sequence*, named `testLogin`, is created in order to call both sequences and retrieve their XML response. To do so, this sequence includes two *Call Sequence* steps:

- `Call_userLogin` step launches the `userLogin` sequence,
- `Call_userLoginWithDefault` step launches the `userLoginWithDefault` sequence.

This sequence is created in the **Sequences** folder of the project and appears, with the two steps, as follows in the **Projects** view:

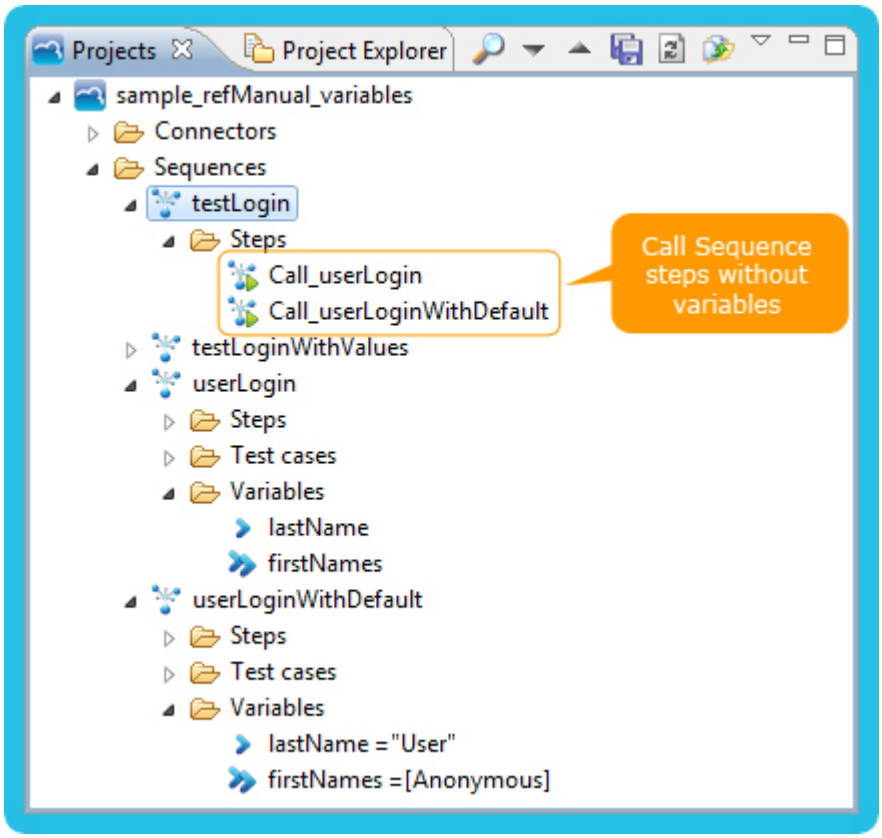


Figure 2 - 91: Call single-valued and multi-valued variables - Sequence and Call Sequence steps in Projects view

The `Call_userLogin` and the `Call_userLoginWithDefault` steps declare no variable, and their **Output** property is set to `true` in order to add the returned XML response to the `testLogin` sequence response.

At runtime, as no variable is provided by the *Call Sequence* steps to the called sequences, they are executed with no variable provided in entry.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

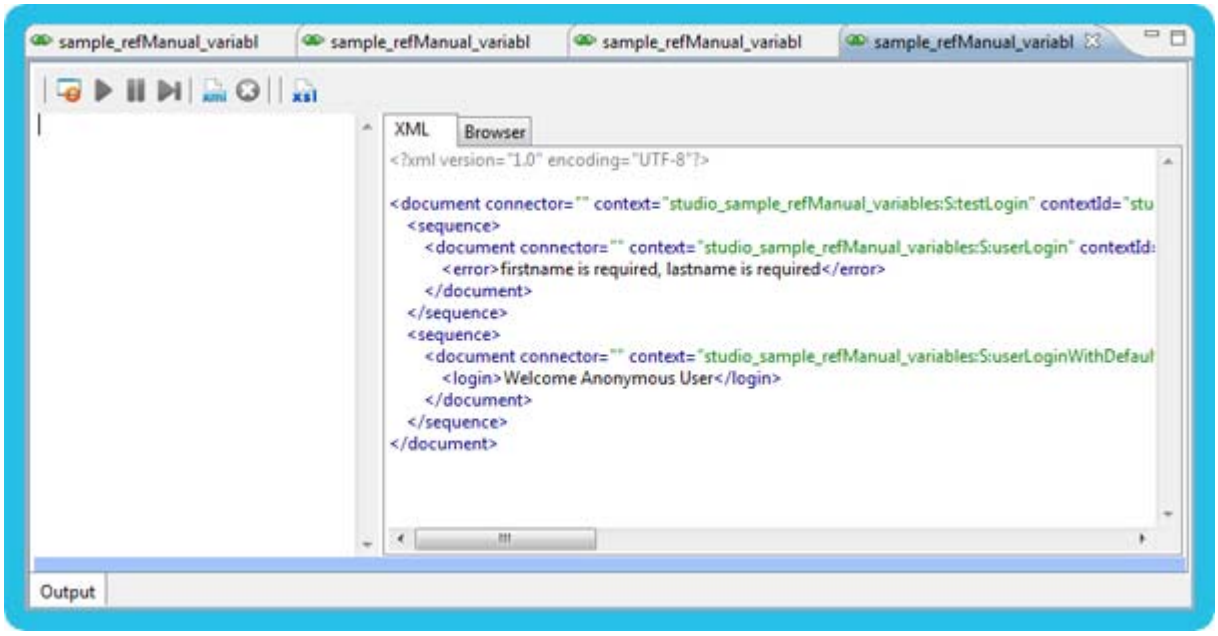


Figure 2 - 92: Call single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED ?

The `lastName` and `firstNames` variables were not provided by the *Call Sequence* steps, thus each target sequence used its variables default values.

The `loginUser` sequence returned an error message as it requires values to be provided.

The `loginUserWithDefault` returned a default `<login>` message as it handles a default case if values are not provided.

Example 2

This example is the same as the previous one except that the *Call Sequence* steps declare variables.

A *Generic sequence*, named `testLoginWithDefault`, is created with two *Call Sequence* steps and two *Element* steps:

- `Call_userLogin` *Call Sequence* step launches the `userLogin` sequence,
- `<firstname>` *Element* step writes an XML element in result XML with value "`Node`",
- `<lastname>` *Element* step writes an XML element in result XML with value "`Value`",
- `Call_userLoginWithDefault` *Call Sequence* step launches the `userLoginWithDefault` sequence.

The `Call_userLogin` step declares two variables imported from the sequence and for which default values have been modified as follows:

- a *Call single-valued variable*, named `lastName`, with the following parameters:

```
Call single-valued variable [
   Default value="Value"
   Source=[]
]
```

- a *Call multi-valued variable*, named `firstNames`, with the following parameters:

```
Call multi-valued variable [
   Default value=["Fixed"]
   Source=[]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

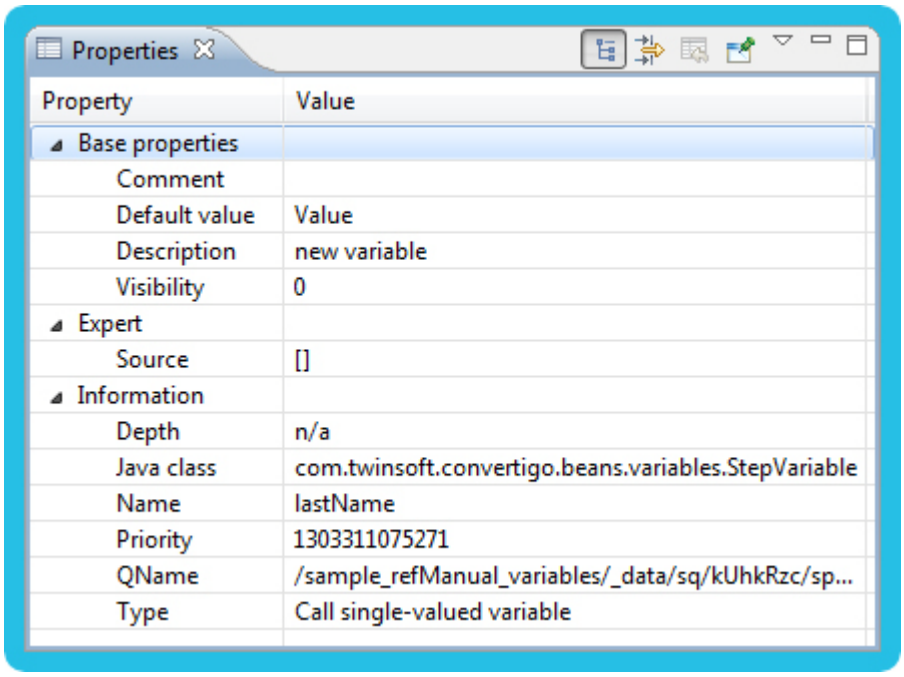


Figure 2 - 93: Call single-valued variable - Configuration example with default value

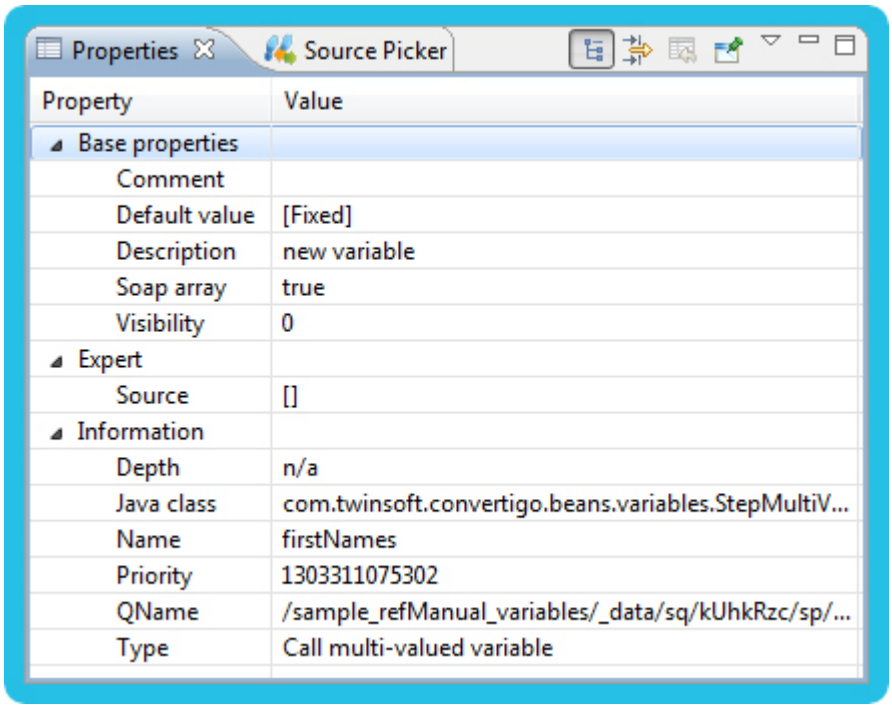


Figure 2 - 94: Call multi-valued variable - Configuration example with default value

The **Default value** property of the *Call multi-valued variable* is edited in the **Array** editor:

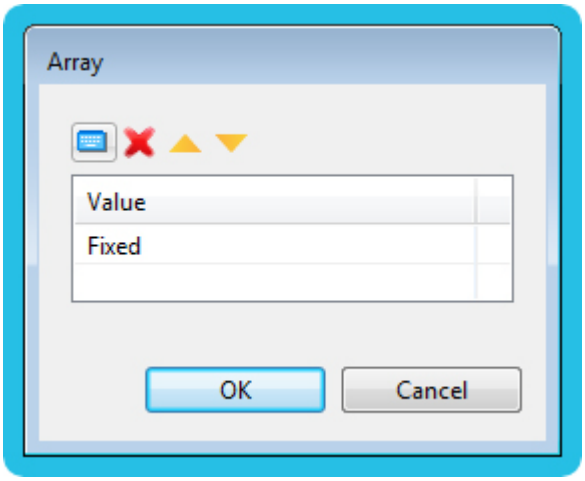


Figure 2 - 95: Call multi-valued variable - Default value property in Array editor

The `Call_userLoginWithDefault` step declares two variables imported from the sequence and for which **Source** property is set as follows:

- a *Call single-valued variable*, named `lastName`, with the following parameters:

```
Call single-valued variable [
   Default value=null
   Source=[1303311539328, .] ==> reference to the preceding <lastname>
step
]
```

- a *Call multi-valued variable*, named `firstNames`, with the following parameters:

```
Call multi-valued variable [
   Default value=null
   Source=[1303311478425,   .]   ==>   reference   to   the   preceding
<firstname> step
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

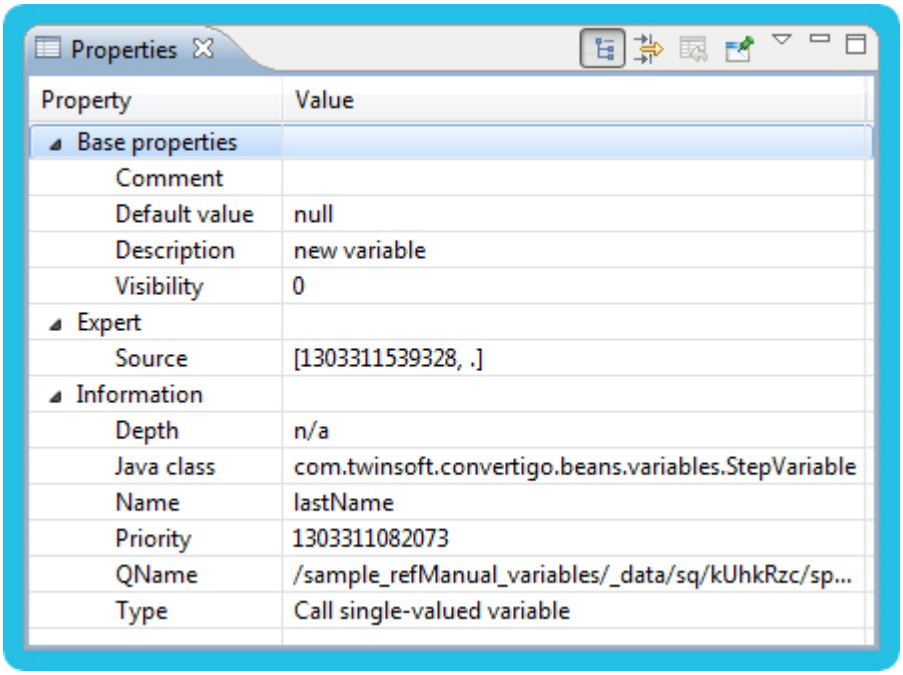


Figure 2 - 96: Call single-valued variable - Configuration example with Source

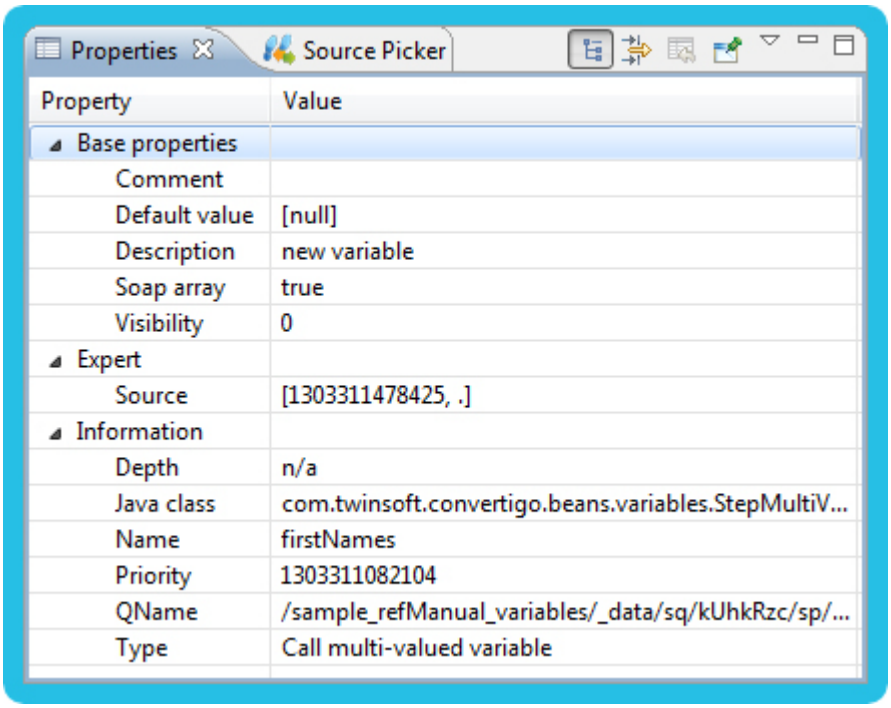


Figure 2 - 97: Call multi-valued variable - Configuration example with Source

The **Source** property of these two variables is set so that the variables are sourced from the relevant previous steps. These properties are edited in the **Step Source** editor with the following configuration (here, the `firstNames` variable **Source** property):

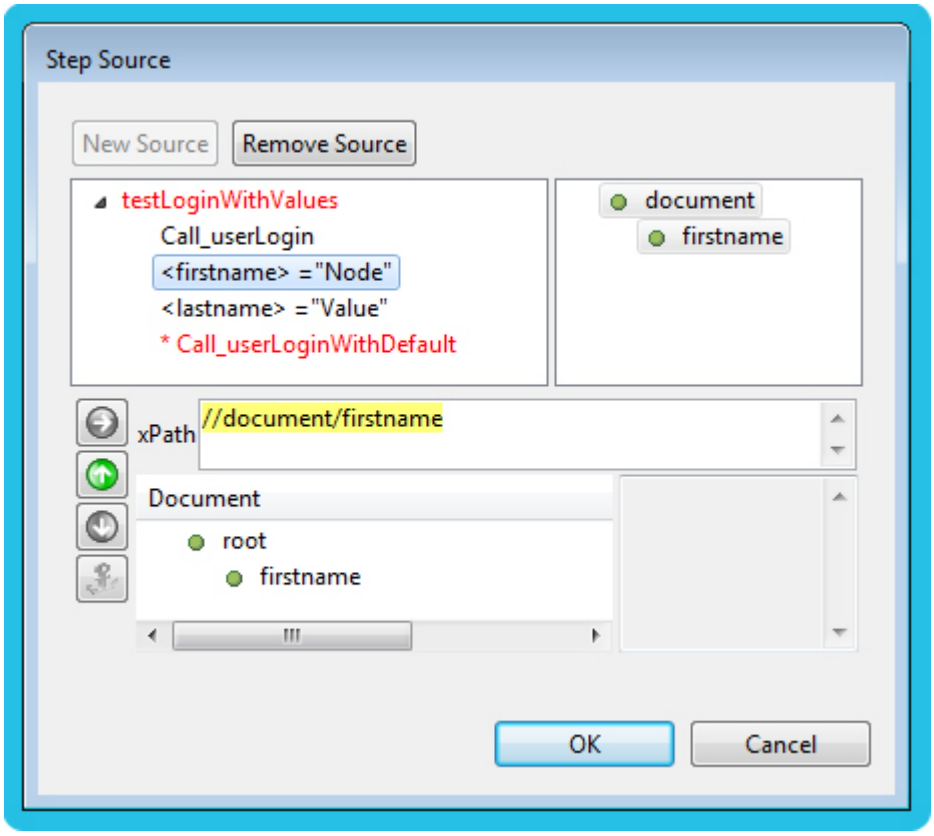


Figure 2 - 98: Call multi-valued variable - Source of the firstNames variable

The sequence is created in the **Sequences** folder of the project and appears, with the steps, as follows in the **Projects** view:

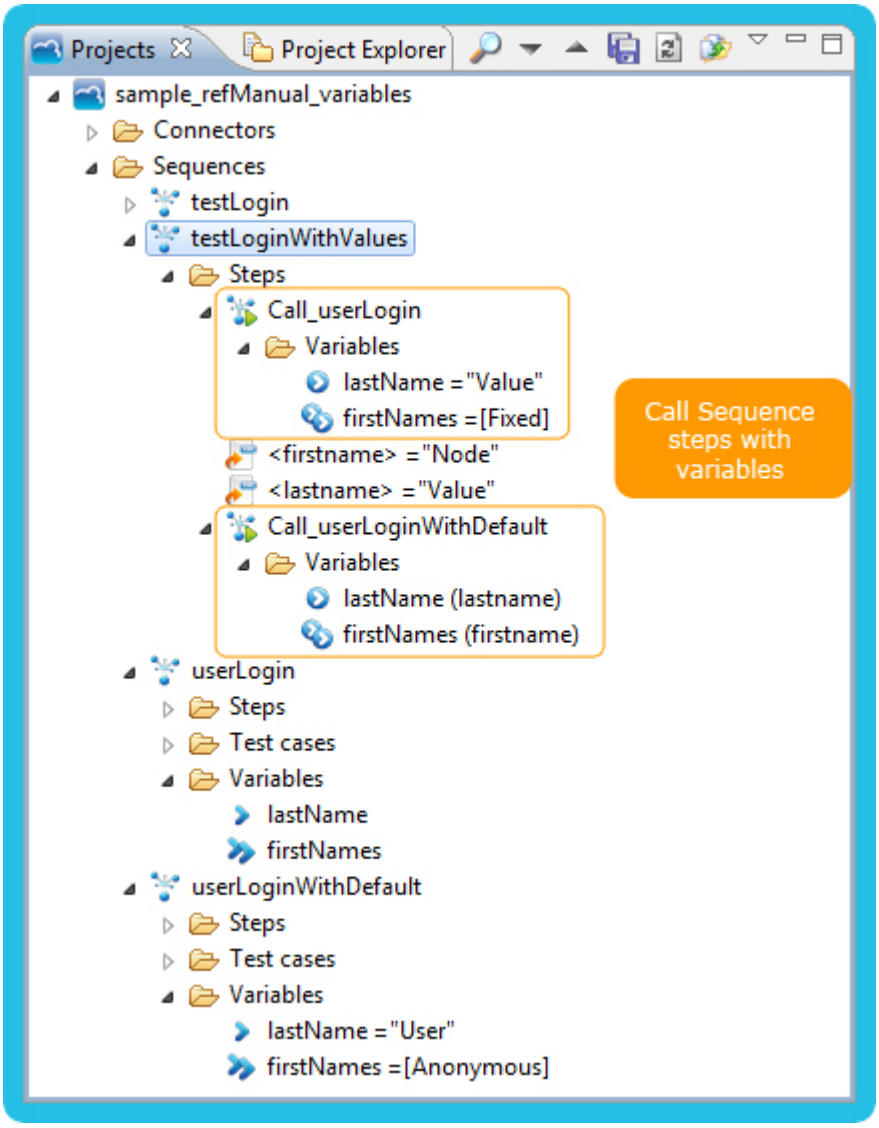


Figure 2 - 99: Call single-valued and multi-valued variables - Call Sequence steps with Call variables in Projects view

Both *Call Sequence* steps **Output** property is set to `true` in order to add the returned XML responses to the `testLogin` sequence XML result.

At runtime, as variables are provided by the *Call Sequence* steps to the executed sequences, they are executed with these variables values.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

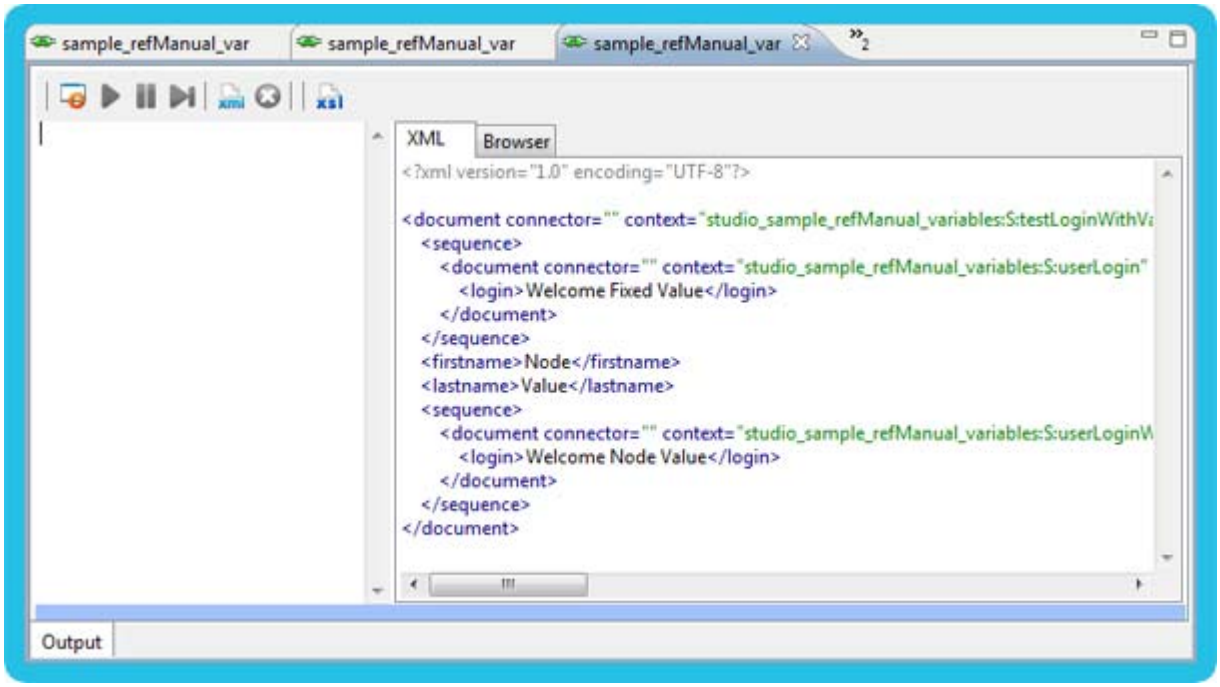


Figure 2 - 100: Call single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED?

The `lastName` and `firstNames` variables were provided by the `Call Sequence` steps, thus each target sequence variables was overridden by the values sent by the steps.

Both of the `loginUser` and `loginUserWithDefault` sequences returned a successful login message.

TEST CASE VARIABLES

TEST SINGLE-VALUED VARIABLE 

OBJECT DESCRIPTION

Defines a single-valued variable for a *Test Case*.

A *Test single-valued variable* is used as a single-valued `input` variable for the transaction or sequence targeted by the *Test Case*.

The variable value to use when executing the *Test Case* is specified in the **Default value** property.

Note: In Convertigo Studio, when a *Test single-valued variable* is created in a *Test Case*, it can be easily replaced by a *Test multi-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **MultiValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.

Property	Type	Category	Description
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.

EXAMPLES

Test Cases are really useful to test any particular case on a transaction or a sequence which declares and uses variables.

The following examples are in continuity with those given for *Request single-valued variable* and *Request multi-valued variable*. Please refer to these objects examples for more information before proceeding.

Example 1

We refers here to the `userLoginWithDefault` sequence of the `sample_refManual_variables` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

A *Test Case*, named `testJohnMarshall`, has been created in order to test the sequence for a given user. This user has two first names, John and Mike, his last name is Marshall.

For this given user, this *Test Case* declares two *Test variables* imported from the sequence and for which default values have been modified as follows:

- a *Test single-valued variable* `lastName` with the following parameters:

```
Test single-valued variable [
   default value="Marshall"
]
```

- a *Test multi-valued variable* `firstNames` with the following parameters:

```
Test multi-valued variable [
  Default value=["John", "Mike"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

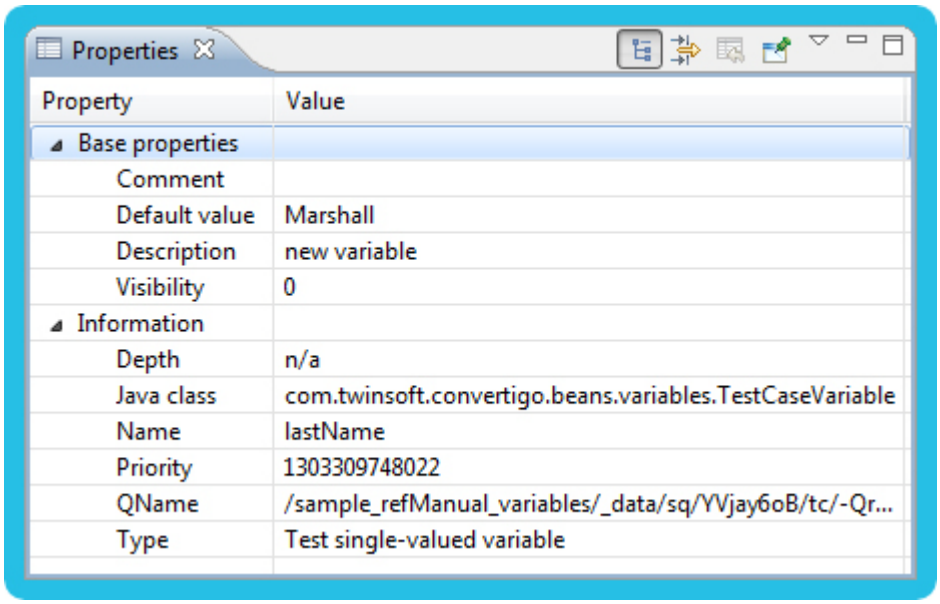


Figure 2 - 101: Test single-valued variable - Configuration example

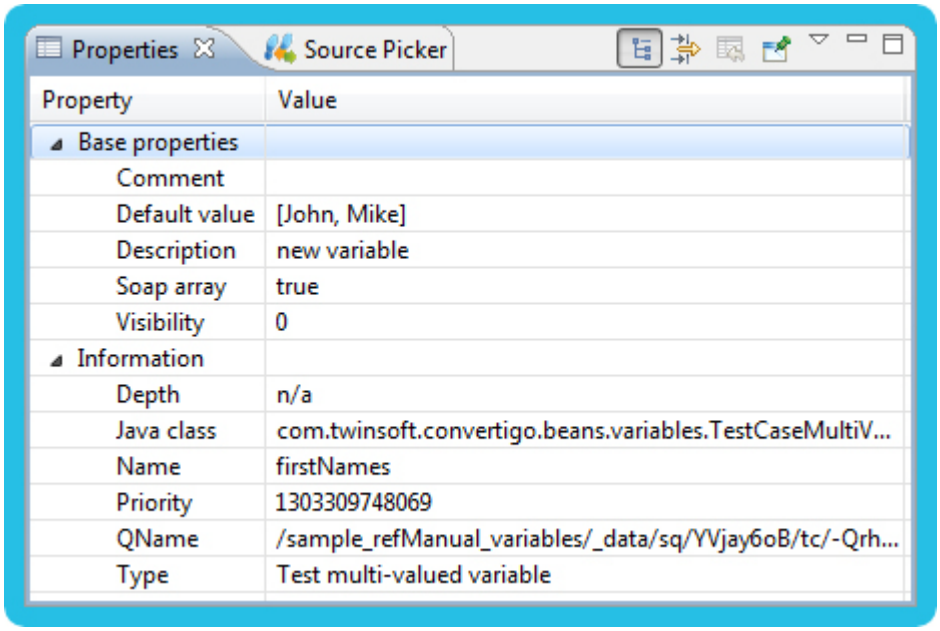


Figure 2 - 102: Test multi-valued variable - Configuration example

The **Default value** properties have been set to Marshall user's specific details. For the *Test multi-valued variable*, it is edited in the **Array** editor:

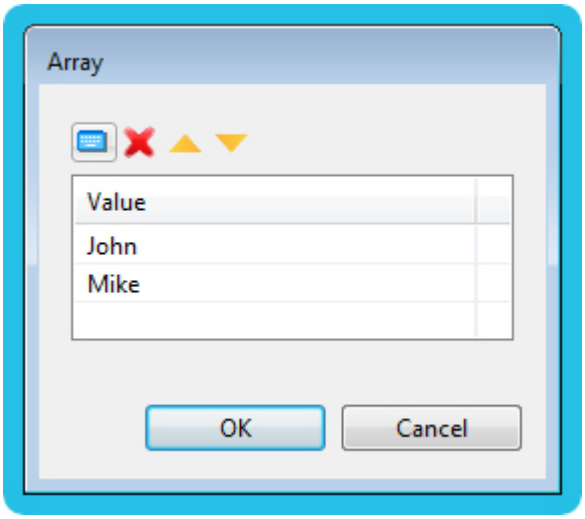


Figure 2 - 103: Test multi-valued variable - Default value property in Array editor

The *Test Case* object is created in the **Test cases** folder of the sequence and appears as follows in the **Projects** view with both *Test single-valued variable* and *Test multi-valued variable*:

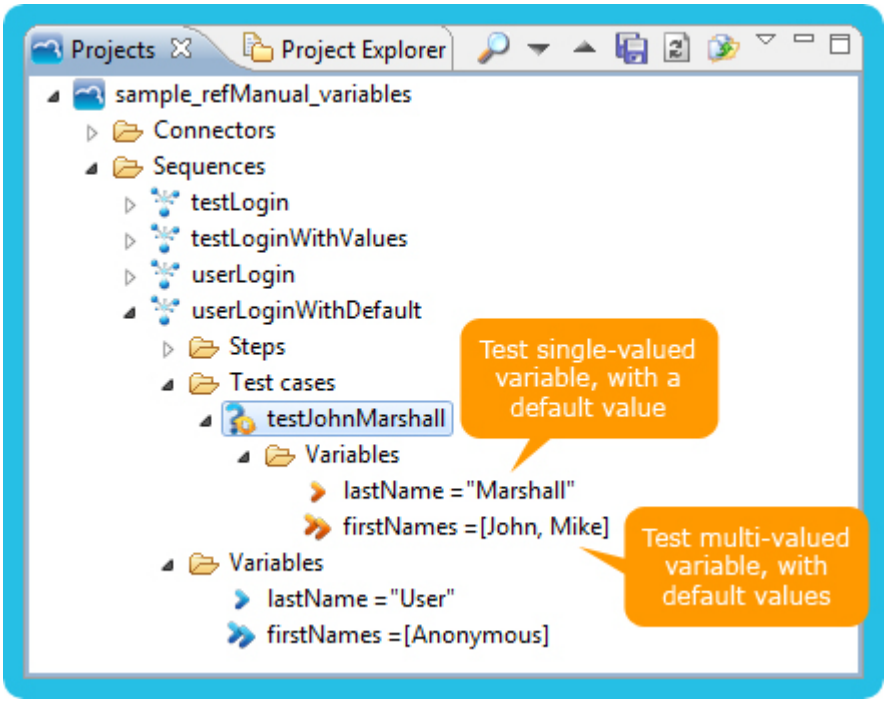


Figure 2 - 104: Test single-valued and multi-valued variables - Test Case and Test variables in Projects view

At runtime, *Test variables* (with their default values) are inserted into the JavaScript scope of the sequence. Thus, when running the *Test Case* (in the Studio or in the test platform), the sequence is executed with the variables using the *Test variables* values.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

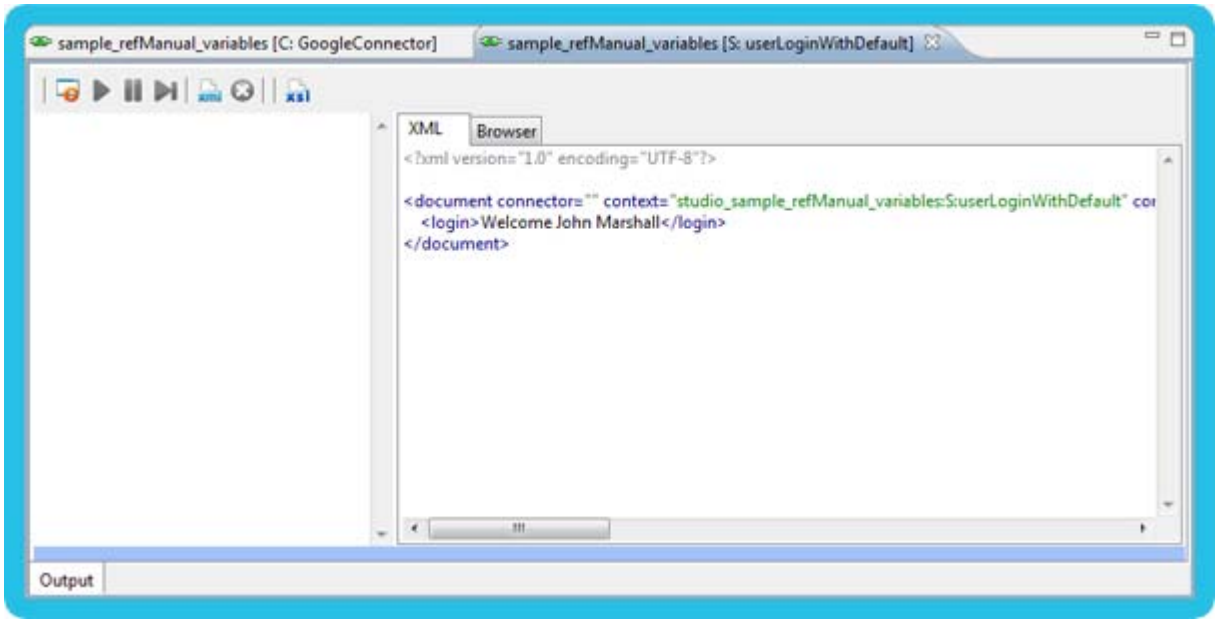


Figure 2 - 105: Test single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED?

Both of sequence's variable default values have been overridden with the *Test Case variables* default values.

Example 2

We refer here to the `userLogin` sequence of the `sample_refManual_variables` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

A *Test Case*, named `testJohn`, has been created in order to test the sequence for a given user without specifying it's last name. We want to verify that the sequence will return an error message as expected.

For this given user, this *Test Case* declares two variables imported from the sequence and for which default values have been modified as follows:

- a *Test single-valued variable* `lastName` with the following parameters:

```
Test single-valued variable [
   Default value=null
]
```

- a *Test multi-valued variable* `firstNames` with the following parameters:

```
Test multi-valued variable [
   Default value=["John"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

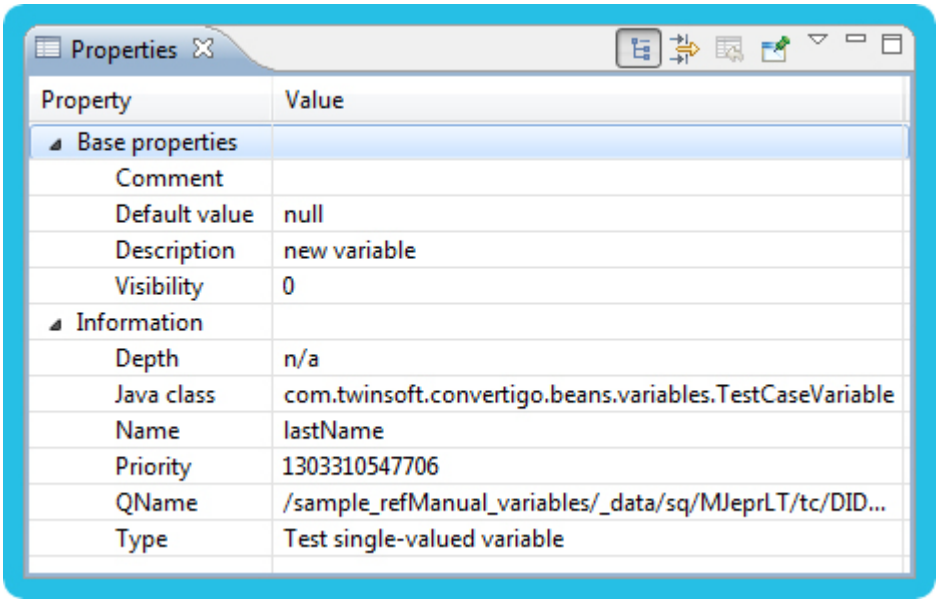


Figure 2 - 106: Test single-valued variable - Configuration example

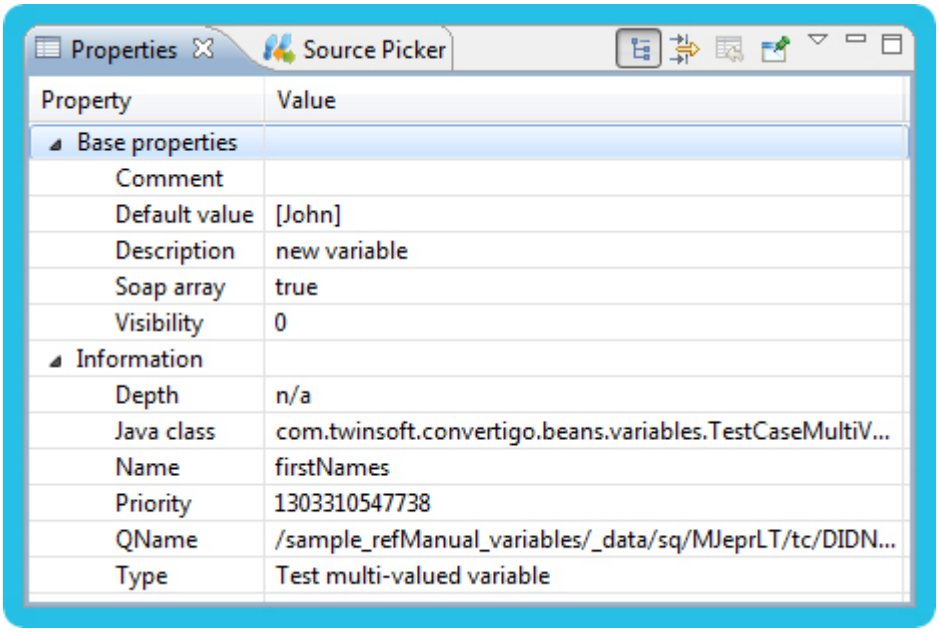


Figure 2 - 107: Test multi-valued variable - Configuration example

The `lastName` variable **Default value** property is set to `null` value in order to be ignored, i.e. not to be sent to the sequence.

The `firstNames` variable **Default value** property is edited in the **Array** editor:

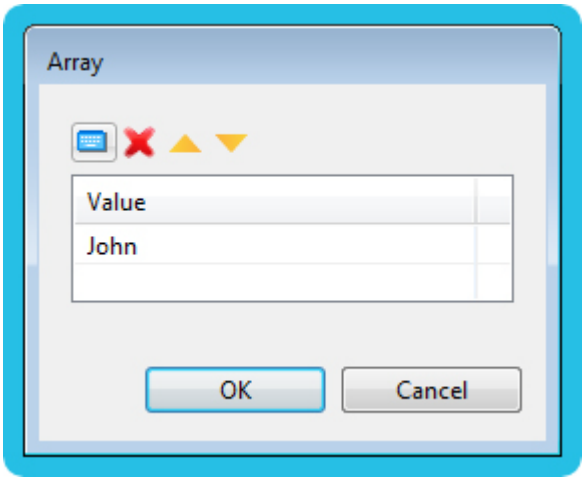


Figure 2 - 108: Test multi-valued variable - Default value property in Array editor

The *Test Case* is created in the **Test cases** folder of the sequence and appears as follows in the **Projects** view with both *Test single-valued variable* and *Test multi-valued variable*:

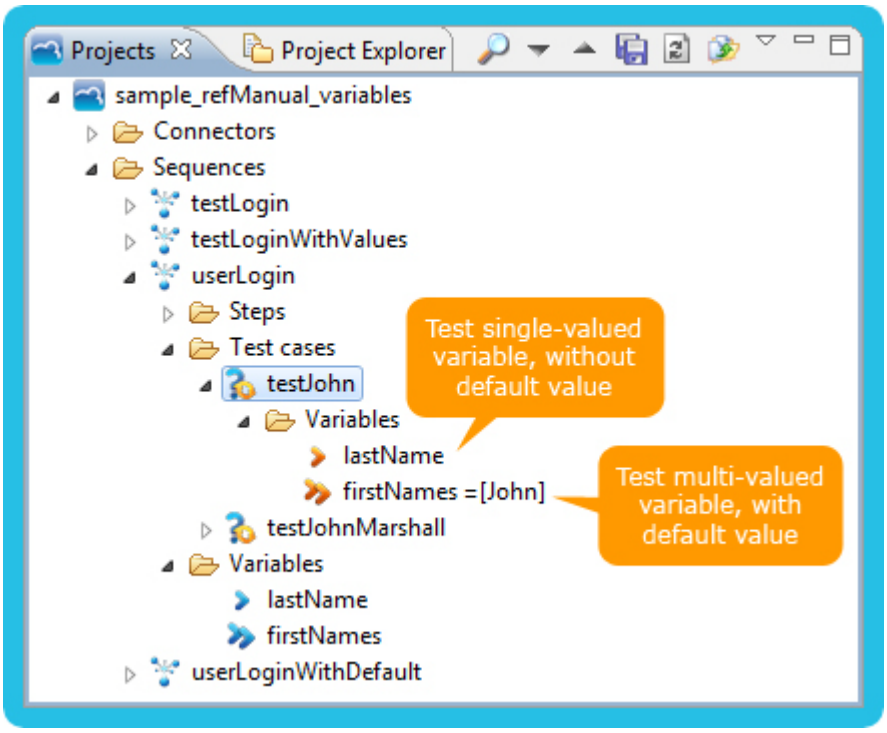


Figure 2 - 109: Test single-valued and multi-valued variables - Test Case and Test variables in Projects view

At runtime, only *firstNames* variable (with its default values) is inserted into the JavaScript scope of the sequence. Thus, when running the *Test Case* (in the Studio or in the test platform), the sequence is executed with the *firstNames* variable using the *Test variable* values.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

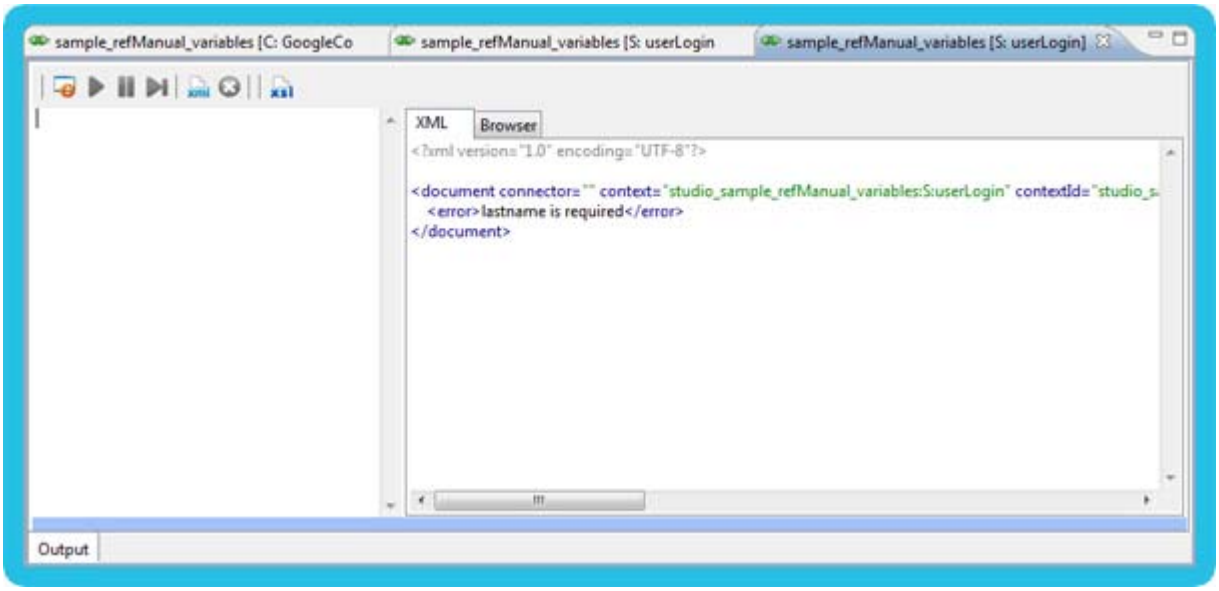


Figure 2 - 110: Test single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED?

The `firstNames` variable has been provided by the *Test Case*, thus sequence's `firstNames` variable default value "`null`" has been overridden by the *Test variable* value "`[John]`".

The `lastName` variable has not been provided by the *Test Case*, thus sequence's `lastName` variable default value "`null`" has been used.

The test fails as expected (following the sequence's implementation) and the error message is returned.

TEST MULTI-VALUED VARIABLE 

OBJECT DESCRIPTION

Defines a multi-valued variable for a *Test Case*.

A *Test multi-valued variable* is used as a multi-valued `input` variable for the transaction or sequence targeted by the *Test Case*.

The variable values to use when executing the *Test Case* are specified in the **Default value** property.

Note: In Convertigo Studio, when a *Test multi-valued variable* is created in a *Test Case*, it can be easily replaced by a *Test single-valued variable*, using the right-click menu on the variable and choosing the option **Change to** > **SingleValued variable**.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default value	Object	standard	Defines the variable's default value(s). This property allows defining a default value or default list of values to use when no variable value is provided to the parent transaction (or sequence). A variable is always created with a default value set to `null`, which means that the variable is only declared and has no default value. At run time, Convertigo looks for the variable among the query parameters, the JavaScript scope or the objects in the context to retrieve its value. If the variable is found, its value is used, if not found, the default value specified by this property is used. In this last case, and if the default value of the variable is not set (**Default value** property set to `null`), an exception can be thrown by any object or JavaScript code trying to use the undefined variable. It is up to the Convertigo developer to unset the variable's `null` value, i.e. to set a default value to the variable. He should prefer using a *Test Case* to test specific values for the variable or pass a variable value directly when invoking the transaction (or sequence). Note: To unset the `null` value of the property, click on the cross-shaped button in the field. Then, the default value is an empty string. You can use it as is or add a value.
Description	String	standard	Describes the variable. This property is used to describe the variable in the widget generated from its parent transaction (or sequence) in Convertigo Mashup Composer.

Property	Type	Category	Description
Soap array	boolean	standard	Defines if the multi-valued variable should be seen as a Soap Array of a occurrence of variables. In the case of transaction or sequence defined as a public SOAP method, this property allows to specify of the current multi-valued variable has to be seen in SOAP envelope as a Soap Array with multiple values inside it or as an occurrence of identical variables.
Visibility	int	standard	Defines the variable's visibility. This property allows defining whether the variable's value is masked or not in: • `log files`: selecting this option will mask the variable's value that may be printed in all loggers, • `studio user interface`: selecting this option will mask the variable's value in the **Properties** view from the Studio, as well as in the tree of the **Projects** view, • `platform user interface`: selecting this option will mask the variable's value in the test platform of the project and when editing the project in Convertigo web administration, • `project's XML files`: selecting this option will mask the variable's value in the project's XML files generated on the file system when saving the objects from the project. Any combination of these options can be chosen, it allows to customize precisely the variable's value display. A last option is available: `Mask value in all`. Selecting this option will mask the variable's value in all previously described cases.

EXAMPLES

Test Cases are really useful to test any particular case on a transaction or a sequence which declares and uses variables.

The following examples are in continuity with those given for *Request single-valued variable* and *Request multi-valued variable*. Please refer to these objects examples for more information before proceeding.

Example 1

We refers here to the `userLoginWithDefault` sequence of the `sample_refManual_variables` project.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **Variables examples** in the **New Project** wizard.*

A *Test Case*, named `testJohnMarshall`, has been created in order to test the sequence for a given user. This user has two first names, John and Mike, his last name is Marshall.

For this given user, this *Test Case* declares two *Test variables* imported from the sequence

and for which default values have been modified as follows:

- a *Test single-valued variable* `lastName` with the following parameters:

```
Test single-valued variable [
  default value="Marshall"
]
```

- a *Test multi-valued variable* `firstNames` with the following parameters:

```
Test multi-valued variable [
  Default value=["John", "Mike"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

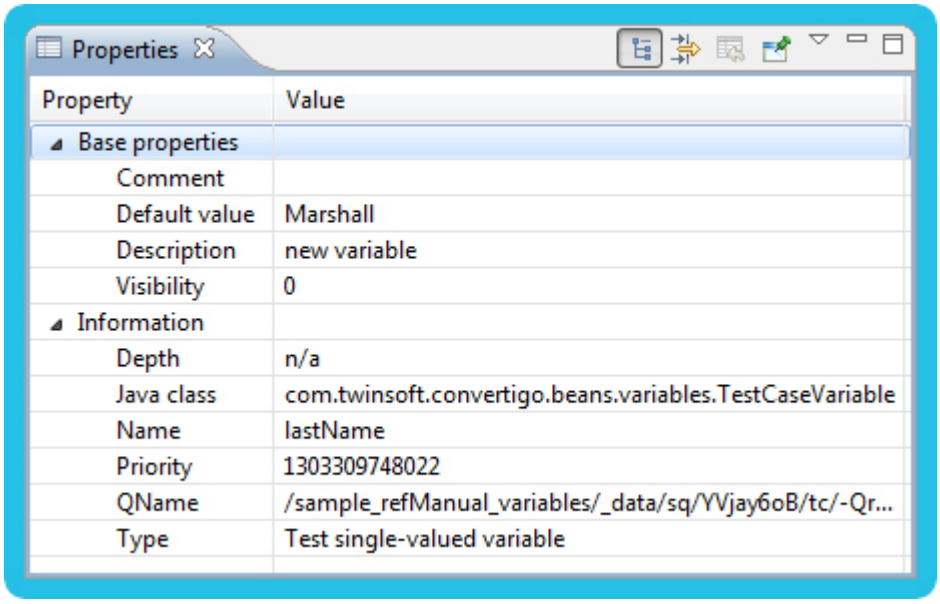


Figure 2 - 111: Test single-valued variable - Configuration example

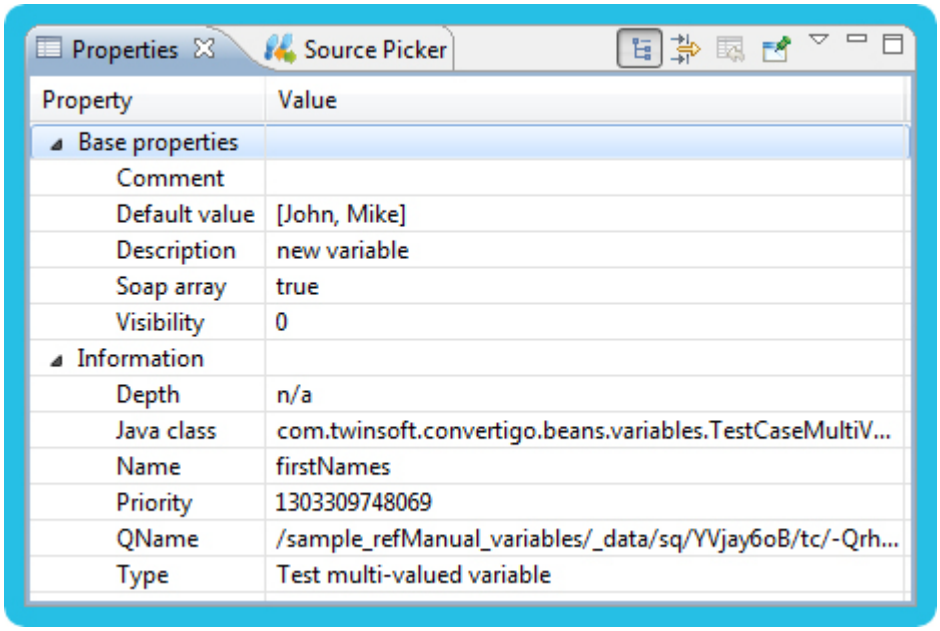


Figure 2 - 112: Test multi-valued variable - Configuration example

The **Default value** properties have been set to Marshall user's specific details. For the *Test multi-valued variable*, it is edited in the **Array** editor:

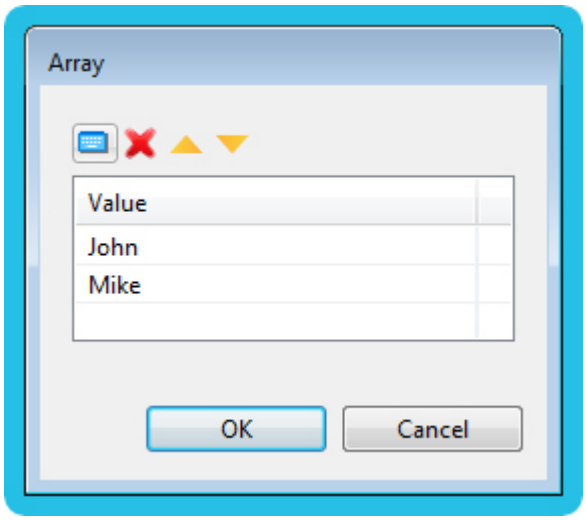


Figure 2 - 113: Test multi-valued variable - Default value property in Array editor

The *Test Case* object is created in the **Test cases** folder of the sequence and appears as follows in the **Projects** view with both *Test single-valued variable* and *Test multi-valued variable*:

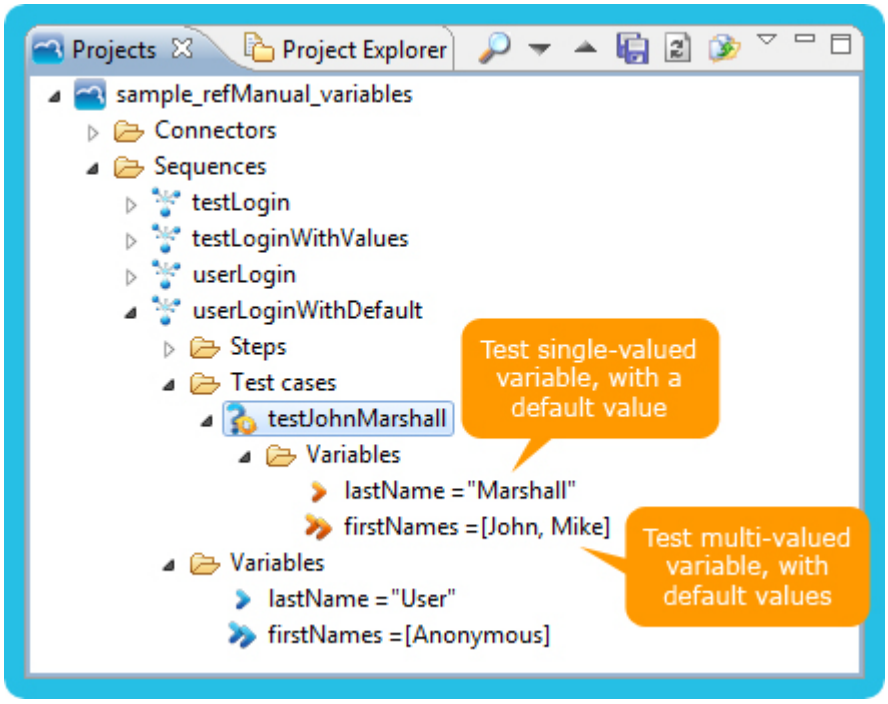


Figure 2 - 114: Test single-valued and multi-valued variables - Test Case and Test variables in Projects view

At runtime, *Test variables* (with their default values) are inserted into the JavaScript scope of the sequence. Thus, when running the *Test Case* (in the Studio or in the test platform), the sequence is executed with the variables using the *Test variables* values.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

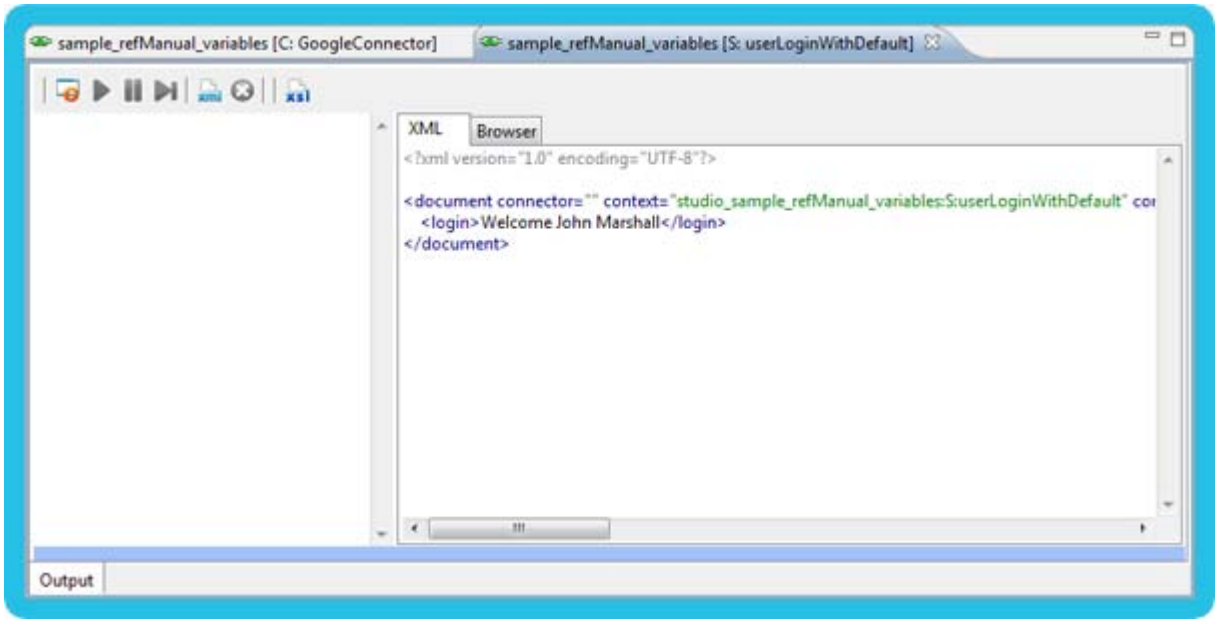


Figure 2 - 115: Test single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED?

Both of sequence's variable default values have been overridden with the *Test Case variables* default values.

Example 2

We refer here to the `userLogin` sequence of the `sample_refManual_variables` project.

> *You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open* ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Variables examples*** *in the* ***New Project*** *wizard.*

A *Test Case*, named `testJohn`, has been created in order to test the sequence for a given user without specifying it's last name. We want to verify that the sequence will return an error message as expected.

For this given user, this *Test Case* declares two variables imported from the sequence and for which default values have been modified as follows:

- a *Test single-valued variable* `lastName` with the following parameters:

```
Test single-valued variable [
   Default value=null
]
```

- a *Test multi-valued variable* `firstNames` with the following parameters:

```
Test multi-valued variable [
   Default value=["John"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

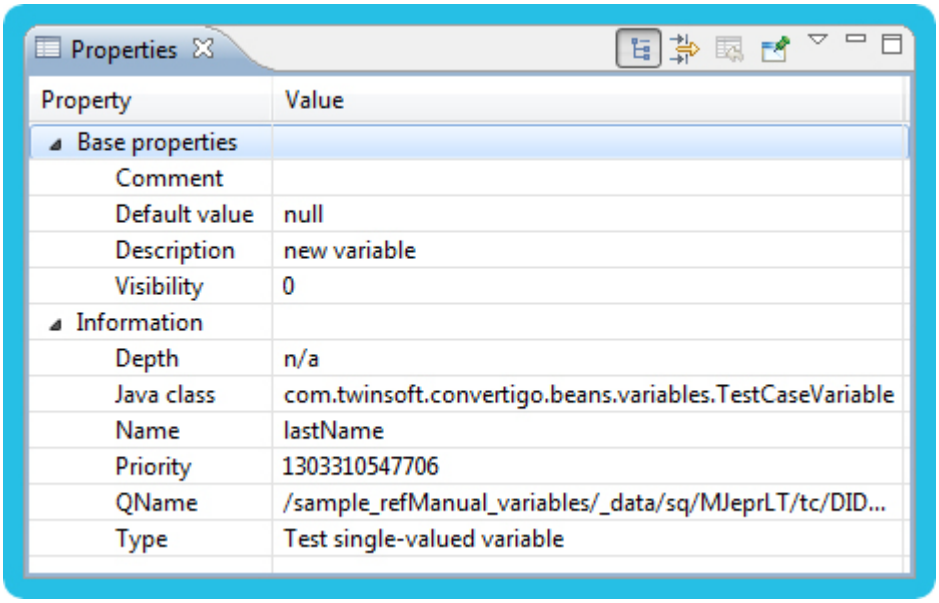


Figure 2 - 116: Test single-valued variable - Configuration example

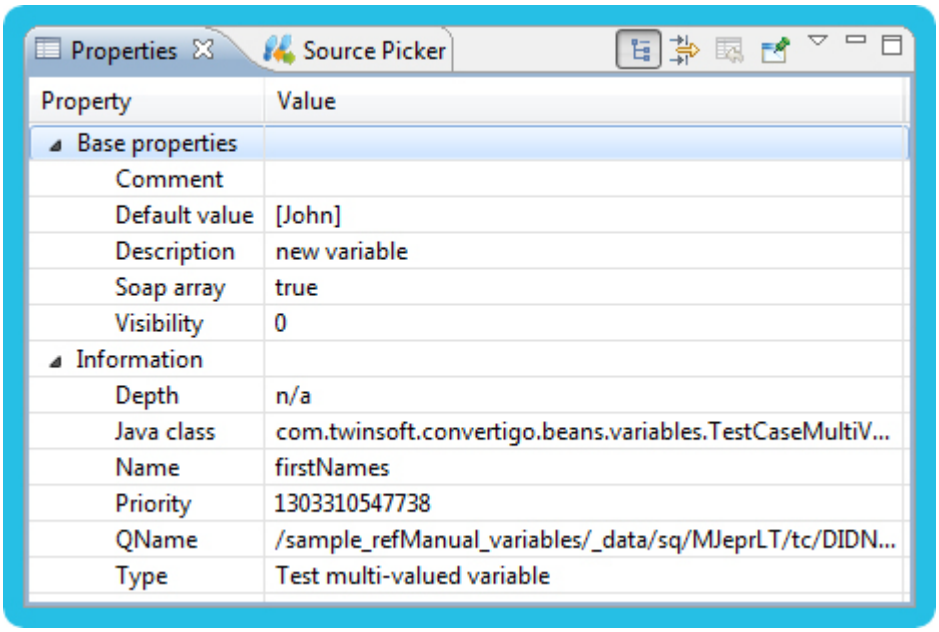


Figure 2 - 117: Test multi-valued variable - Configuration example

The `lastName` variable **Default value** property is set to `null` value in order to be ignored, i.e. not to be sent to the sequence.

The `firstNames` variable **Default value** property is edited in the **Array** editor:

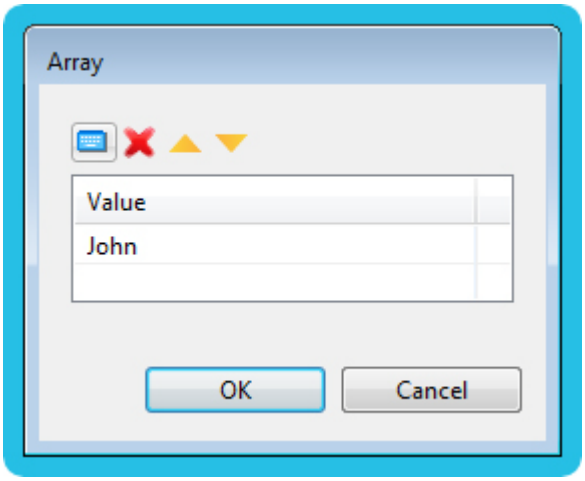


Figure 2 - 118: Test multi-valued variable - Default value property in Array editor

The *Test Case* is created in the **Test cases** folder of the sequence and appears as follows in the **Projects** view with both *Test single-valued variable* and *Test multi-valued variable*:

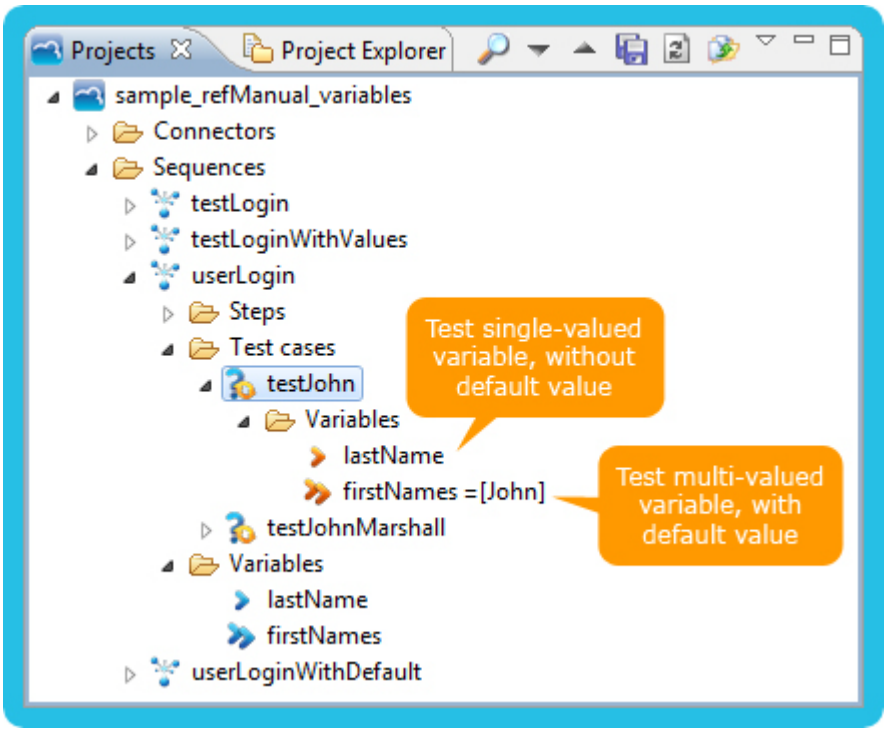


Figure 2 - 119: Test single-valued and multi-valued variables - Test Case and Test variables in Projects view

At runtime, only *firstNames* variable (with its default values) is inserted into the JavaScript scope of the sequence. Thus, when running the *Test Case* (in the Studio or in the test platform), the sequence is executed with the *firstNames* variable using the *Test variable* values.

In the Studio, the resulting XML is displayed in the **XML** tab of the sequence editor:

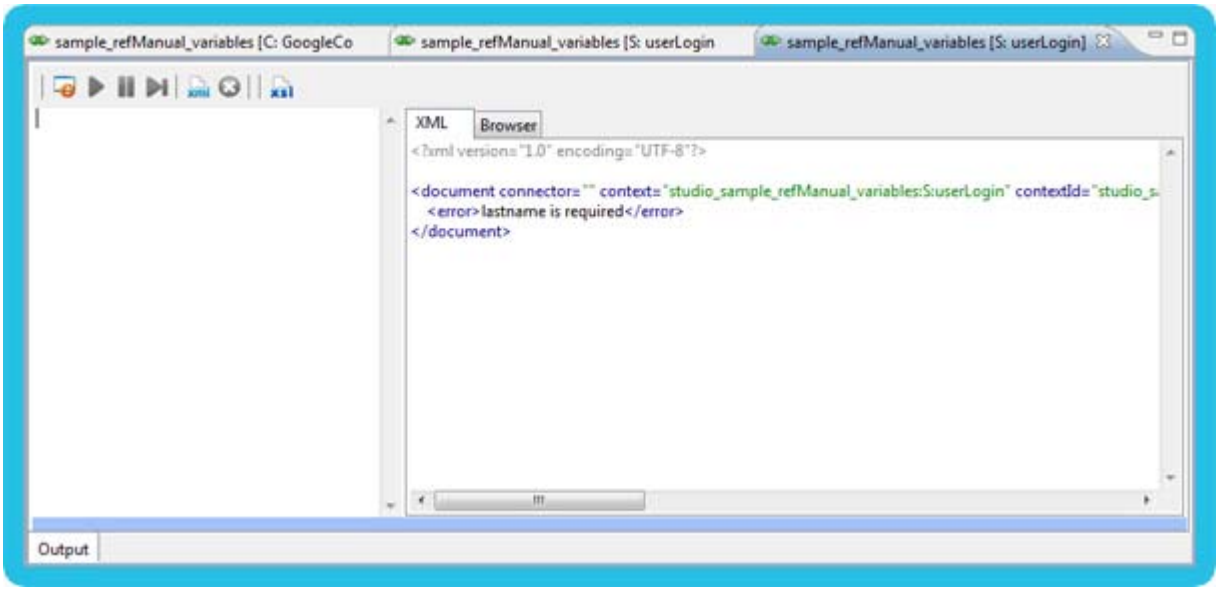


Figure 2 - 120: Test single-valued and multi-valued variables - Sequence execution result in sequence editor

WHAT HAPPENED?

The `firstNames` variable has been provided by the *Test Case*, thus sequence's `firstNames` variable default value "`null`" has been overridden by the *Test variable* value "`[John]`".

The `lastName` variable has not been provided by the *Test Case*, thus sequence's `lastName` variable default value "`null`" has been used.

The test fails as expected (following the sequence's implementation) and the error message is returned.

2.1.3 References

SCHEMA REFERENCES

IMPORT XSD SCHEMA 

OBJECT DESCRIPTION

References an XSD file and imports its schemas in this project.

The *Import XSD schema* reference enhances the current project's schema by importing the referenced XSD file.

The imported XSD objects (types, elements, groups, ...) can be used anywhere in current project sequences, using the **Assigned XSD Complex type QName** and **Assigned XSD Element ref QName** properties.

Note: The imported XSD file should declare a target namespace different from the target namespace of the current project. It is mandatory for an XSD to be imported in another.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
XSD URL	String	standard	Defines the URL of the XSD file to import. If the XSD file to import is outside of the current project (either a remote file or a file on the developer's computer but external to the project), this **XSD URL** property is used to define the file's URL. Notes: • If the file to import is located in the file system (local or network drives), use the "Browse" button of the wizard: it will automatically create the correct file URL depending on your operating system (`file://[host]/path` or `file:[//host]/path`). • Only one of both **XSD local path** or **XSD URL** properties can be used. If both are filled, only **XSD local path** property is used.
XSD local path	String	standard	Defines the Convertigo local path of the imported XSD file. If the XSD file to import is a local file in the current project or in the current workspace, this **XSD local path** property is used to define the local file path. This path is relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. Note: Only one of both **XSD local path** or **XSD URL** properties can be used. If both are filled, only **XSD local path** property is used.

IMPORT WSDL SCHEMA 

OBJECT DESCRIPTION

References a WSDL file and imports its schemas in this project.

The *Import WSDL schema* reference enhances the current project's schema by importing the referenced WSDL file's schemas.

The imported XSD objects (types, elements, groups, ...) can be used anywhere in current project sequences, using the **Assigned XSD Complex type QName** and **Assigned XSD Element ref QName** properties.

Note: The imported WSDL file should declare a target namespace different from the target namespace of the current project. It is mandatory for an XSD to be imported in another.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
WSDL URL	String	standard	Defines the URL of the WSDL file to import. If the WSDL file to import is outside of the current project (either a remote file or a file on the developer's computer but external to the project), this **WSDL URL** property is used to define the file's original URL. Notes: • If the file to import is located in the file system (local or network drives), use the "Browse" button of the wizard: it will automatically create the correct file URL depending on your operating system (`file://[host]/path` or `file:[//host]/path`). • Once imported, the WSDL file will be copied locally in the current project's resources. This will then fill the **WSDL local path** property. If both are filled, only **WSDL local path** property is used.
WSDL local path	String	standard	Defines the Convertigo local path of the imported WSDL file. If the WSDL file to import is a local file in the current project or in the current workspace, this **WSDL local path** property is used to define the local file path. This path is relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. Note: Only one of both **WSDL local path** or **WSDL URL** properties can be used. If both are filled, only **WSDL local path** property is used.

INCLUDE XSD SCHEMA 

OBJECT DESCRIPTION

References an XSD file and includes its schemas in this project.

The *Include XSD schema* reference enhances the current project's schema by including the referenced XSD file.

The included XSD objects (types, elements, groups, ...) can be used anywhere in current project sequences, using the **Assigned XSD Complex type QName** and **Assigned XSD Element ref QName** properties.

Note: The included XSD file should declare the same target namespace as the current project. It is mandatory for an XSD to be included in another.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
XSD URL	String	standard	Defines the URL of the XSD file to import. If the XSD file to import is outside of the current project (either a remote file or a file on the developer's computer but external to the project), this **XSD URL** property is used to define the file's URL. Notes: • If the file to import is located in the file system (local or network drives), use the "Browse" button of the wizard: it will automatically create the correct file URL depending on your operating system (`file://[host]/path` or `file:[//host]/path`). • Only one of both **XSD local path** or **XSD URL** properties can be used. If both are filled, only **XSD local path** property is used.
XSD local path	String	standard	Defines the Convertigo local path of the imported XSD file. If the XSD file to import is a local file in the current project or in the current workspace, this **XSD local path** property is used to define the local file path. This path is relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. Note: Only one of both **XSD local path** or **XSD URL** properties can be used. If both are filled, only **XSD local path** property is used.

IMPORT PROJECT SCHEMA 

OBJECT DESCRIPTION

References a Convertigo project and imports its schema in this project.

The *Import Project schema* reference enhances the current project's schema by importing the referenced project's XSD.

The imported XSD objects are used when *Call Sequence*/*Call Transaction* steps are used in current project's sequences. In this case, when creating a *Call Sequence*/*Call Transaction* step, the *Import Project schema* reference is automatically created.

The imported XSD objects can also be used anywhere else in current project sequences, using the **Assigned XSD Complex type QName** and **Assigned XSD Element ref QName** properties.

Note: The referenced project must be present in the same Convertigo as current project.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Project name	String	standard	The name of the Convertigo project which XSD is referenced in this project. This property allows to choose the project name to reference from all projects existing in the Convertigo.

Web Service references

IMPORT WEB SERVICE 

OBJECT DESCRIPTION

References a web service by creating the *HTTP connector* with all transactions matching the web service methods, referencing its WSDL file and importing its schemas in this project.

The *Import web service* reference creates an *HTTP connector* configured to target the web service. It automatically creates *XML HTTP transactions* for each method described by the web service, including their variables.

The *Import web service* reference also enhances the current project's schema by importing the referenced WSDL file's schemas (such as an *Import WSDL schema* reference).

The imported XSD objects (types, elements, groups, ...) are automatically used to define the transactions output schemas. They can also be used anywhere in current project sequences, using the **Assigned XSD Complex type QName** and **Assigned XSD Element ref QName** properties.

Notes:

- The referenced WSDL file is copied locally in current project's resources, in `wsdl` folder.
- The imported WSDL file should declare a target namespace different from the target namespace of the current project. It is mandatory for an XSD to be imported in another.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
WSDL URL	String	standard	Defines the URL of the WSDL file to import. If the WSDL file to import is outside of the current project (either a remote file or a file on the developer's computer but external to the project), this **WSDL URL** property is used to define the file's original URL. Notes: • If the file to import is located in the file system (local or network drives), use the "Browse" button of the wizard: it will automatically create the correct file URL depending on your operating system (`file://[host]/path` or `file:[//host]/path`). • Once imported, the WSDL file will be copied locally in the current project's resources. This will then fill the **WSDL local path** property. If both are filled, only **WSDL local path** property is used.

Property	Type	Category	Description
WSDL local path	String	standard	Defines the Convertigo local path of the imported WSDL file. If the WSDL file to import is a local file in the current project or in the current workspace, this **WSDL local path** property is used to define the local file path. This path is relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. Note: Only one of both **WSDL local path** or **WSDL URL** properties can be used. If both are filled, only **WSDL local path** property is used.

2.2 Mobile Application

2.2.1 Main objects

MOBILE APPLICATION 

OBJECT DESCRIPTION

Defines the mobile application implemented in this project.

The *Mobile application* object allows the developer to define a mobile application in a Convertigo Mobile project. Only one *Mobile application* object can be associated with a Convertigo *Project*.

The *Mobile application* object represents the mobile application developed in project's resources. It takes place for the web mobile application as well as for the native device applications that are built using Convertigo Mobile Builder server, leading to the generation of mobile applications that can be installed on devices.

The *Mobile application* object includes the Flash Update functionality. When mobile application pages and resources are changed on the Convertigo project, the Flash Update will automatically update them in the mobile applications installed on devices. This feature can be enabled or not.

Most properties of *Mobile application* are taken into account at application build. They cannot be updated thanks to the Flash Update: the app needs to be built again and updated on stores when these properties are changed. It is also the case for the `config.xml` configuration file: if it is changed, it is taken into account only at application re-build.

Only the following *Mobile application* properties are always directly updated at Flash Update:

- **Enable Flash Update** property,
- **Flash Update requires user confirmation** property,
- **Application version** property (only updated in JavaScript variable, accessible using `C8O.getCordovaEnv("currentVersion");` method),
- **Splashscreen hiding mode** property,
- and **Accessibility** property.

Note: If no file content has been modified, the Flash Update does not detect that some updates were made. The following cases are not managed for the moment:

- When updating the **Application version** property: if no other file content is updated, the Flash Update does not detect this change and the new value is not updated. The value available in JavaScript expression `C8O.getCordovaEnv("currentVersion");` remains the old one.
- When renaming or deleting file(s): if no other file content is updated, the file renaming or deletion is not detected by the Flash Update. These changes are not updated on mobile devices and the old files remain.

These limitations are due to the fact that the Flash Update relies for now on the update date of files. If you want one of these changes to be taken into account, update one of the project resource files and save this update. It will lead the Flash Update to detect that an update was made.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	expert	Defines the mobile application's accessibility. This property can take the following values: • `Public`: The mobile application is runnable from everyone, it is visible in the Test Platform and it can be built. This is the default value. • `Hidden`: The mobile application is runnable but only from people who know the execution URL, it is not visible in the Test Platform so cannot be built, excepted for people who are identified in the Test Platform as administrator. • `Private`: The mobile application is only runnable from people who are identified in the Test Platform as administrator, it is not visible in the Test Platform and cannot be built, excepted for people who are identified in the Test Platform as administrator. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see, run and build all mobile applications, no matter what their **Accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see, run and build public mobile applications and run hidden ones if he knows their execution URL.
Application ID	String	standard	Defines the mobile application ID. If empty, the mobile application ID is set by default to `com.convertigo.mobile.<project_name>`, with `<project_name>` the name of the Convertigo project. The mobile application ID is used to build the mobile applications on the Convertigo Mobile builder platform. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("applicationId");` method.
Application author email	String	standard	Defines the author email of the mobile application. When the mobile application is built, the **Application author email** property defines the built application author email in the build server. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("applicationAuthorEmail");` method.

Property	Type	Category	Description
Application author name	String	standard	Defines the author name of the mobile application. When the mobile application is built, the **Application author name** property defines the built application author name in the build server. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("applicationAuthorName");` method.
Application author website	String	standard	Defines the website URL of the mobile application's author. When the mobile application is built, the **Application author website** property defines the built application author's website URL in the build server. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("applicationAuthorWebsite");` method.
Application description	String	standard	Defines the short description of the mobile application. When the mobile application is built, the **Application description** property defines the built application short description in the build server. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("applicationDescription");` method.
Application name	String	standard	Defines the name used by the mobile application. When the mobile application is built, the **Application name** property defines the built application name, used to identify the application in the build server. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("applicationName");` method.
Application version	String	standard	Defines the mobile application's version. This property allows the project's developer to set a version to the mobile application. It is used for the built mobile application. The version syntax should be of the following form: `x.y.z`, with x, y and z being numbers. If not, the value is automatically transformed to an `x.y.z` version. For example, "2" will be transformed to "2.0.0", "3.1" to "3.1.0", "3.1.4_beta" to "3.1.4". If left empty, the version of the parent *Project* is used. Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("builtVersion");` method. If this value is updated thanks to the Flash Update, the current version (not from build) is available in client JavaScript code using the `C8O.getCordovaEnv("currentVersion");` method.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Convertigo server endpoint	String	standard	Defines the URL of the Convertigo server used by the mobile application. The **Convertigo server endpoint** property defines the accessible URL of the Convertigo server where the project is deployed. This Convertigo server needs to be accessed by the mobile application after installation on devices. If this property is left empty, the default value is set to `http://<current_Convertigo_host:port>/<Convertigo_webapp_name>`, with: • `<current_Convertigo_host:port>` being the host name or IP address, plus port number, of the Convertigo server currently accessed, • `<Convertigo_webapp_name>` being the current Convertigo webapp name (for example `convertigo` for Studio or on-premises Server, `cems` for Cloud Server). Note: After the application is built, this value is available in client JavaScript code using the `C8O.getCordovaEnv("endPoint");` method.
Enable Flash Update	boolean	standard	Defines whether the Flash Update feature is enabled for this mobile application. The Flash Update feature allows the mobile application to be notified when updates have been deployed on the server. The installed applications on mobile devices are then be automatically updated. The **Enable Flash Update** property can be set to `true`, enabling the Flash Update, or to `false`, disabling the Flash Update. Default value is `true`. Note: When this property is changed, the built applications already installed on devices will automatically take the new value into account, allowing the developer to switch between Flash Update modes without building again the app and re-deploying it on the stores.
Flash Update build mode	FlashUpdateBuildMode	expert	Defines the mobile application build mode. The mobile application can be built empty for installation on devices or already containing the all user interface. The **Build mode** property can take one of the following values: • `full`: the built mobile application will contain all the user interface and resources, • `light`: the built mobile application is a shell that will download the user interface and resources at first launch. In this case, the Flash Update feature should be enabled thanks to the **Enable Flash Update** property, otherwise, the UI and resources will not be downloaded and the app will remain empty.

Property	Type	Category	Description
Flash Update requires user confirmation	boolean	expert	Defines if the Flash Update feature requires the user confirmation. When the Flash Update feature is enabled thanks to the **Enable Flash Update** property, the **Flash Update requires user confirmation** property allows to define whether the application update is done automatically at application startup (property set to `false`), or after a confirmation from the mobile application's user (property set to `true`). Default value is `false`, which sets the Flash Update to be automatic. Note: When this property is changed, the built applications already installed on devices will automatically take the new value into account, allowing the developer to switch between Flash Update modes without building again the app and re-deploying it on the stores.
Flash Update timeout	long	expert	Defines the maximum time (in ms) for Flash Update to check for application resource updates. If the timeout is reached, the Flash Update automatically redirects to the local application without update. An infinite timeout can be defined by setting this property to `0`: the Flash Update will wait endlessly for updates. In case of network failure (no connection to network from the device, HTTP error from server, etc.), the timeout is not used and the Flash Update automatically redirects to the local application.
Mobile builder authentication token	String	expert	Defines the authentication token of the Mobile builder account to use to build the mobile application. When building a mobile application, a Mobile builder account (which is nothing more than a PhoneGap build account) is mandatory. Convertigo provides one by default, used by default in Convertigo engine. This default Mobile builder authentication token can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level authentication token will be used by default for all mobile applications built by the Convertigo. The **Mobile builder authentication token** property allows to override the Mobile builder authentication token for this mobile application's build. If left empty, the common Mobile builder authentication token defined at Convertigo engine level is used. Note: Once a PhoneGap build account is configured thanks to the Mobile builder authentication token, do not forget to configure all mobile platforms certificates and keys in accordance. Refer to the documentation of each platform object for more information.

Property	Type	Category	Description
Splashscreen hiding mode	SplashRemove Mode	standard	Defines the hiding mode of the mobile application's splashscreen. This property allows the project's developer to configure how and when the application splashscreen has to be hidden. This property can take the following values: • `Before Flash Update`: The application's splashscreen is automatically hidden by the C8O JavaScript library, before the Flash Update starts. The Flash Update page is visible in the mobile application and then redirects to the application pages. • `After Flash Update`: The application's splashscreen is automatically hidden by the C8O JavaScript library, after the Flash Update starts. The Flash Update page is not visible in the mobile application, masked by the splashscreen. The splashscreen is hidden when the library redirects to the application pages. • `Manual`: The application's splashscreen is not automatically hidden by the C8O JavaScript library. The splashscreen must be explicitely hidden by the mobile application JavaScript code, using: `C8O.splashscreenHide();` method. For *Android* platform, the `config.xml` file can declare a timeout for splashscreen hiding: `SplashScreenDelay`. Note: The C8O JavaScript API includes two methods to manipulate the splashscreen directly in application JavaScript code: • `C8O.splashscreenShow();` allows to show the splashscreen, • `C8O.splashscreenHide();` allows to hide the splashscreen.

2.2.2 Platforms

MOBILE PLATFORMS

ANDROID 

OBJECT DESCRIPTION

Android mobile platform

Android mobile platform allows creating an Android application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Android certificate password	String	expert	Defines Android certificate password to use for building the Android application from this application. When building a mobile application for Android platform, an Android certificate (including title, password and keystore password) is mandatory. Convertigo provides one by default, this Android certificate is used by default in Convertigo engine. This default Android certificate can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level Android certificate will be used by default for all Android dedicated applications built by the Convertigo. The **Android certificate password** property allows to override the Android certificate password for this mobile application's build. If left empty, the common Android certificate password defined at Convertigo engine level is used. Note: The Android certificate is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the Android certificate (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

Property	Type	Category	Description
Android certificate title	String	expert	Defines Android certificate title for building the Android application from this application. When building a mobile application for Android platform, an Android certificate (including title, password and keystore password) is mandatory. Convertigo provides one by default, the Convertigo Android certificate is used by default in Convertigo engine, . This default Android certificate can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level Android certificate will be used by default for all Android dedicated applications built by the Convertigo. The **Android certificate title** property allows to override the Android certificate title for this mobile application's build. If left empty, the common Android certificate title defined at Convertigo engine level is used. Note: The Android certificate is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the Android certificate (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.
Android keystore password	String	expert	Defines Android keystore password to use for building the Android application from this application. When building a mobile application for Android platform, an Android certificate (including title, password and keystore password) is mandatory. Convertigo provides one by default, this Android certificate is used by default in Convertigo engine. This default Android certificate can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level Android certificate will be used by default for all Android dedicated applications built by the Convertigo. The **Android keystore password** property allows to override the Android keystore password for this mobile application's build. If left empty, the common Android keystore password defined at Convertigo engine level is used. Note: The Android certificate is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the Android certificate (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

BLACKBERRY 

OBJECT DESCRIPTION

BlackBerry mobile platform

BlackBerry mobile platform allows creating a BlackBerry application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
BlackBerry key password	String	expert	Defines BlackBerry key password to use for building the BlackBerry application from this application. When building a mobile application for BlackBerry platform, a BlackBerry key (including title and password) is mandatory. Convertigo provides one by default, this BlackBerry key is used by default in Convertigo engine. This default BlackBerry key can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level BlackBerry key will be used by default for all Android dedicated applications built by the Convertigo. The **BlackBerry key password** property allows to override the BlackBerry key password for this mobile application's build. If left empty, the common BlackBerry key password defined at Convertigo engine level is used. Note: The BlackBerry key is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the BlackBerry key (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

Property	Type	Category	Description
BlackBerry key title	String	expert	Defines BlackBerry key title to use for building the BlackBerry application from this application. When building a mobile application for BlackBerry platform, a BlackBerry key (including title and password) is mandatory. Convertigo provides one by default, this BlackBerry key is used by default in Convertigo engine. This default BlackBerry key can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level BlackBerry key will be used by default for all BlackBerry dedicated applications built by the Convertigo. The **BlackBerry key title** property allows to override the BlackBerry key title for this mobile application's build. If left empty, the common BlackBerry key title defined at Convertigo engine level is used. Note: The BlackBerry key is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the BlackBerry key (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

BLACKBERRY 10 

OBJECT DESCRIPTION

BlackBerry 10 mobile platform

BlackBerry 10 mobile platform allows creating a BlackBerry 10 application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
BlackBerry key password	String	expert	Defines BlackBerry key password to use for building the BlackBerry application from this application. When building a mobile application for BlackBerry platform, a BlackBerry key (including title and password) is mandatory. Convertigo provides one by default, this BlackBerry key is used by default in Convertigo engine. This default BlackBerry key can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level BlackBerry key will be used by default for all Android dedicated applications built by the Convertigo. The **BlackBerry key password** property allows to override the BlackBerry key password for this mobile application's build. If left empty, the common BlackBerry key password defined at Convertigo engine level is used. Note: The BlackBerry key is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the BlackBerry key (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

Property	Type	Category	Description
BlackBerry key title	String	expert	Defines BlackBerry key title to use for building the BlackBerry application from this application. When building a mobile application for BlackBerry platform, a BlackBerry key (including title and password) is mandatory. Convertigo provides one by default, this BlackBerry key is used by default in Convertigo engine. This default BlackBerry key can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level BlackBerry key will be used by default for all BlackBerry dedicated applications built by the Convertigo. The **BlackBerry key title** property allows to override the BlackBerry key title for this mobile application's build. If left empty, the common BlackBerry key title defined at Convertigo engine level is used. Note: The BlackBerry key is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the BlackBerry key (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

iOS 

Object Description

iOS mobile platform

iOS mobile platform allows creating an iOS application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
iOS certificate password	String	expert	Defines iOS certificate password to use for building the iOS application from this application. When building a mobile application for iOS platform, an iOS certificate (including title and password) is mandatory. Convertigo provides one by default, this iOS certificate is used by default in Convertigo engine. This default iOS certificate can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level iOS certificate will be used by default for all iOS dedicated applications built by the Convertigo. The **iOS certificate password** property allows to override the iOS certificate password for this mobile application's build. If left empty, the common iOS certificate password defined at Convertigo engine level is used. Note: The iOS certificate is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the iOS certificate (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

Property	Type	Category	Description
iOS certificate title	String	expert	Defines iOS certificate title to use for building the iOS application from this application. When building a mobile application for iOS platform, an iOS certificate (including title and password) is mandatory. Convertigo provides one by default, this iOS certificate is used by default in Convertigo engine. This default iOS certificate can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level iOS certificate will be used by default for all iOS dedicated applications built by the Convertigo. The **iOS certificate title** property allows to override the iOS certificate title for this mobile application's build. If left empty, the common iOS certificate title defined at Convertigo engine level is used. Note: The iOS certificate is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the iOS certificate (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

WINDOWS 8 

OBJECT DESCRIPTION

Windows 8 platform

Windows 8 platform allows creating a Windows 8 application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

WINDOWS PHONE 7 

OBJECT DESCRIPTION

Windows Phone 7 mobile platform

Windows Phone 7 mobile platform allows creating a Windows Phone 7 application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Windows Phone publisher ID title	String	expert	Defines Windows Phone publisher ID title to use for building the Windows Phone 7 application from this application. When building a mobile application for Windows Phone platform, a Window Phone publisher ID (including its title) is mandatory. Convertigo provides one by default, this Window Phone publisher ID is used by default in Convertigo engine. This default Window Phone publisher ID can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level Window Phone publisher ID will be used by default for all Windows Phone dedicated applications built by the Convertigo. The **Window Phone publisher ID title** property allows to override the Window Phone publisher ID title for this mobile application's build. If left empty, the common Window Phone publisher ID title defined at Convertigo engine level is used. Note: The Windows Phone publisher ID is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the Windows Phone publisher ID (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

Windows Phone 8 

Object Description

Windows Phone 8 mobile platform

Windows Phone 8 mobile platform allows creating a Windows Phone 8 application from the *Mobile application* below which it is added.

The mobile application dedicated to the platform is built from:

- the common resources of the *Mobile application*, located in `<project_folder>/DisplayObjects/mobile` (with `<project_folder>` the root folder of your mobile project resources),
- and are possibly completed by the resources dedicated to the platform, located in `<project_folder>/DisplayObjects/platforms/<platform_object_name>` (with `<project_folder>` the root folder of your mobile project resources, and with `<platform_object_name>` the resources folder named after your mobile platform object name).

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Windows Phone publisher ID title	String	expert	Defines Windows Phone publisher ID title to use for building the Windows Phone 8 application from this application. When building a mobile application for Windows Phone platform, a Window Phone publisher ID (including its title) is mandatory. Convertigo provides one by default, this Window Phone publisher ID is used by default in Convertigo engine. This default Window Phone publisher ID can be configured at engine level, in the *Mobile builder* tab of the *Administration Console*'s *Configuration* page. This engine level Window Phone publisher ID will be used by default for all Windows Phone dedicated applications built by the Convertigo. The **Window Phone publisher ID title** property allows to override the Window Phone publisher ID title for this mobile application's build. If left empty, the common Window Phone publisher ID title defined at Convertigo engine level is used. Note: The Windows Phone publisher ID is linked to a PhoneGap build account. If a **Mobile builder authentication token** is configured in the *Mobile application* or at Convertigo engine level, the Windows Phone publisher ID (defined here or at Convertigo engine level) must be one of the "Signing keys" declared in this PhoneGap build account.

2.3 Sequencer

2.3.1 Main objects

Generic Sequence 

Object Description

Defines and orchestrates a series of actions.

A *Sequence* defines actions (called *Steps*), the order in which they are executed and conditions of execution. It follows a logical process meant at achieving a specific goal.

A *Sequence* can be set as part of the project containing transactions to be orchestrated or as part of any other project. For example, a sequencer project can contain *Sequences* orchestrating exclusively transactions from other projects.

Any *Sequence* can be triggered as a web service in SOAP or REST protocol. It can return XML data combined from the orchestrated transactions and other steps to the web service caller. Any XML output structure can be defined using appropriate steps.

"Blind" *Sequences* are also possible - they do not return any data to the caller, but can for example insert data in databases using a defined *SQL connector*, or insert data into forms (thanks to *HTML transaction* for example).

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/ sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/ sequences and run hidden ones if he knows their execution URL.

Property	Type	Category	Description
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Include response element	boolean	expert	Defines whether the response element should be included in web service result or whether the data should be directly under the public method response node (i.e. allows to remove the encompassing 'response' node or not)
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the sequence has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From sequence`: Use the XSL style sheet attached to the sequence. When used, make sure a style sheet is added to the sequence.

EXAMPLES

Example 1

Let's consider the `sample_documentation_CWI` project set in the context of the "*Starting with Convertigo Web Integrator*" tutorial. It includes a transaction called `searchGoogleWithLimit`:

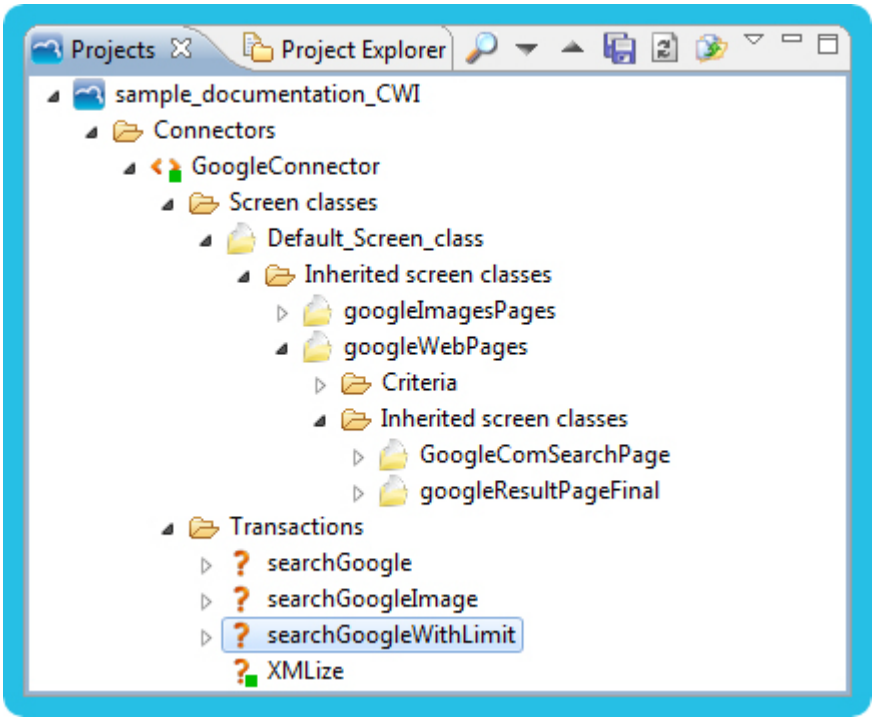


Figure 2 - 121: Generic Sequence - Project with existing transaction

In this example, we want to define a *Generic Sequence*, in a new project named `sample_refManual_steps`, orchestrating several calls to the `searchGoogleWithLimit` transaction.

*You can find the complete example projects in the Studio. To open these projects, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **Sequencer objects examples** in the **New Project** wizard.*
*Associated project can be opened by selecting **Convertigo Samples and Demos** > **Documentation samples** > **Web integration** in the **New Project** wizard.*

A *Generic Sequence* object is created with the following properties:

```
Generic Sequence [
   accessibility=hidden
   response timeout=360
   style sheet=none
]
```

The *Generic Sequence* object is created in the **Sequences** folder of a new project and appears as follows in the **Projects** view:

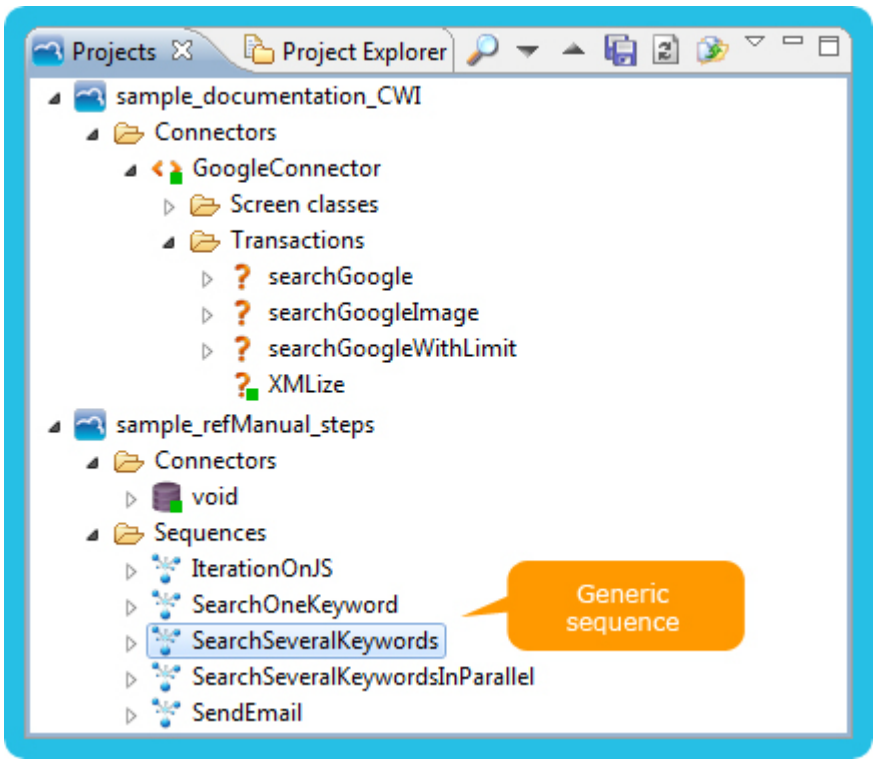


Figure 2 - 122: Generic Sequence - Sequence object created in Projects view

Its parameters are edited in the **Properties** view of the Convertigo Studio:

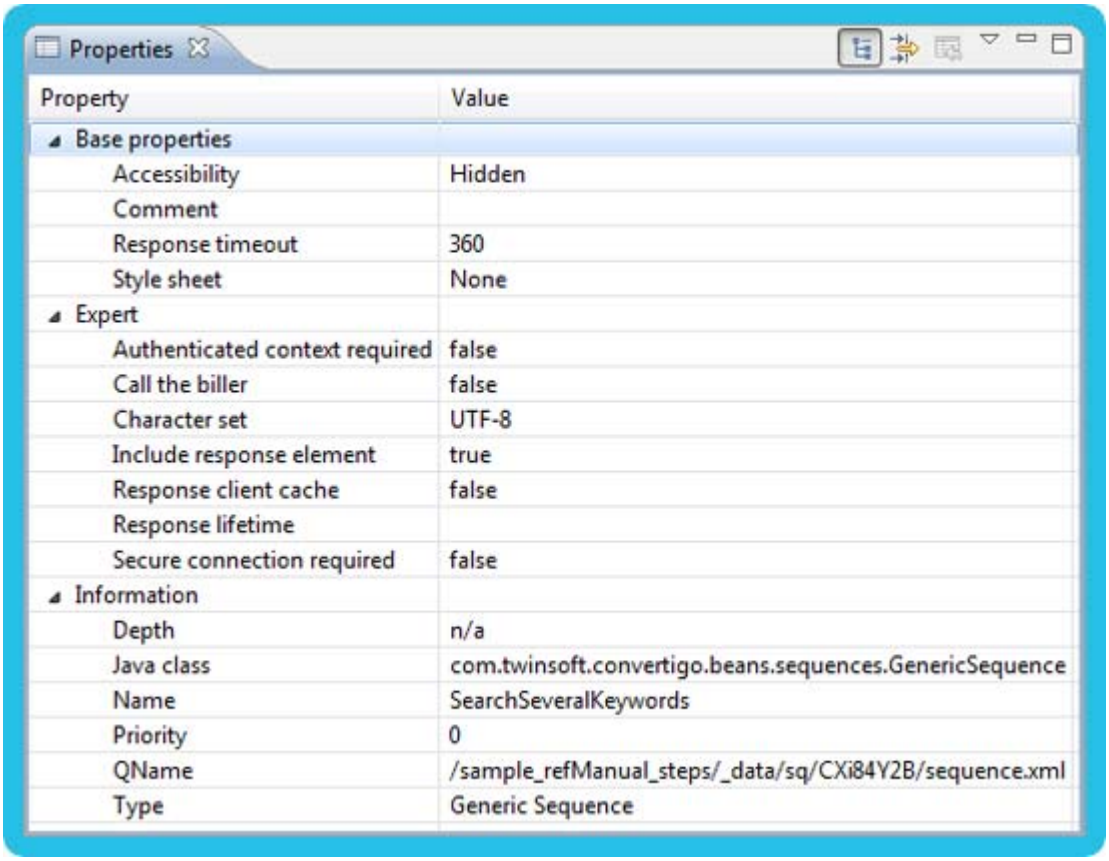


Figure 2 - 123: Generic Sequence - Configuration example

The **Response timeout** property is set to "360" for the *Generic Sequence* to have the time to orchestrate several calls to the `searchGoogle` transaction, which **Reponse timeout** property is set to "60".

To complete the *Generic Sequence*, *Request variables* have to be added as well as *Steps* defining the actions to orchestrate. See "*Request single-valued variable*" and "*Request multi-valued variable*" documentations and example as well as "*Iterator step*" and "*Transaction step*" documentations and examples.

Example 2

The `sample_documentation_CMS` project set in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial includes two sequences, each of them illustrating one of the two main purposes of CMS sequences:

- the first sequence, `GetXMLData`, is meant at orchestrating transactions so as to collect data, arrange ("mix" as required by the user) collected data and produce an arranged XML output,
- the second sequence, `InsertDataInBase`, is meant at orchestrating transactions so as to collect data, arrange ("mix" as required by the user) collected data, insert them into database tables and produce a *check* XML output, that is to say XML data informing the user about possible errors returned by SQL transactions.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

Both sequences are made up of a *Generic Sequence*, of *steps* and are associated with a *sequence variable* (in this example, the sequence variable matches the variable of the first called transaction).

The `GetXMLData` sequence appears as follows in the **Projects** view of the Convertigo Studio:

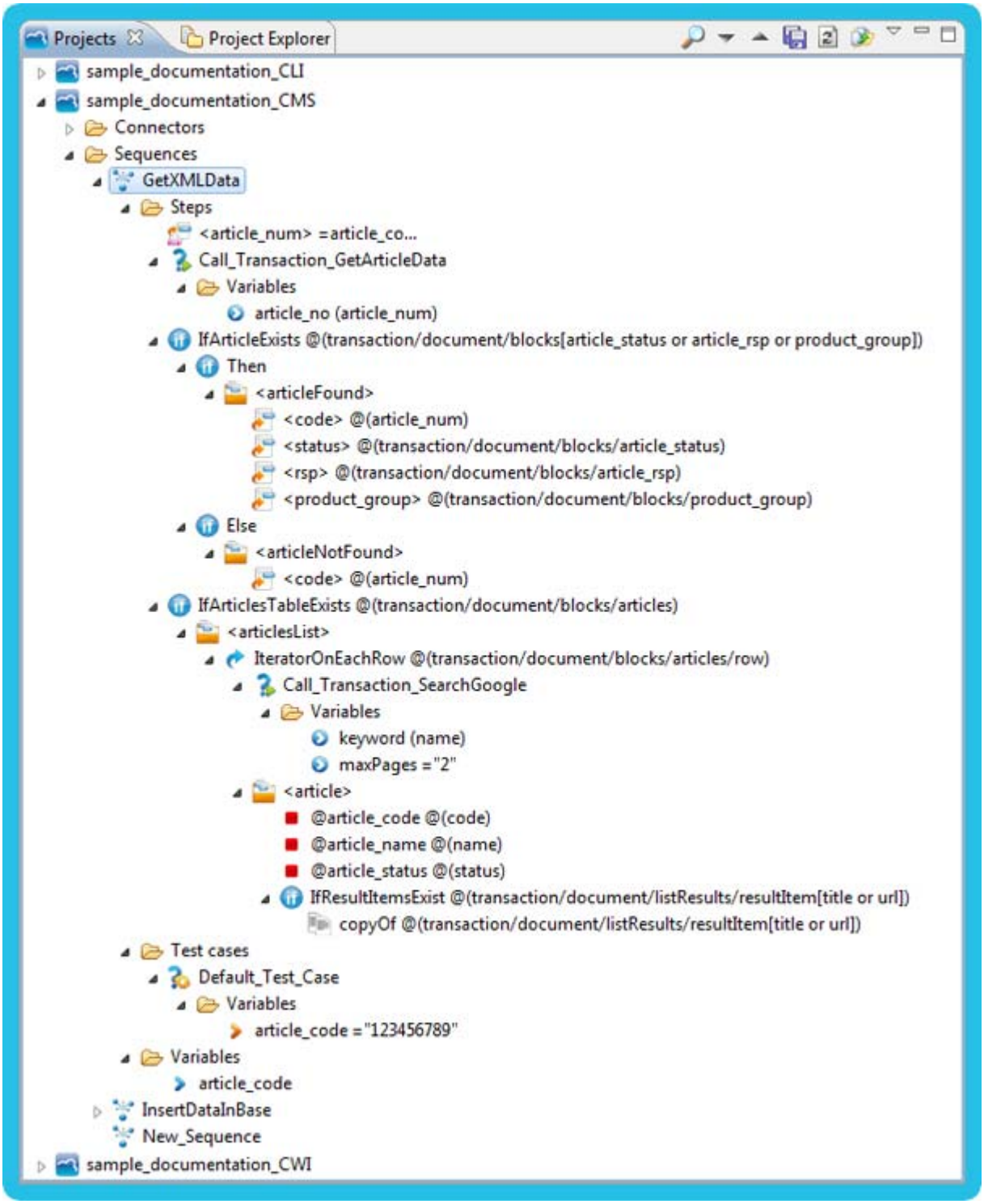


Figure 2 - 124: Generic Sequence - GetXMLData sequence in Projects view

The `InsertDataInBase` sequence appears as follows in the **Projects** view of the Convertigo Studio:

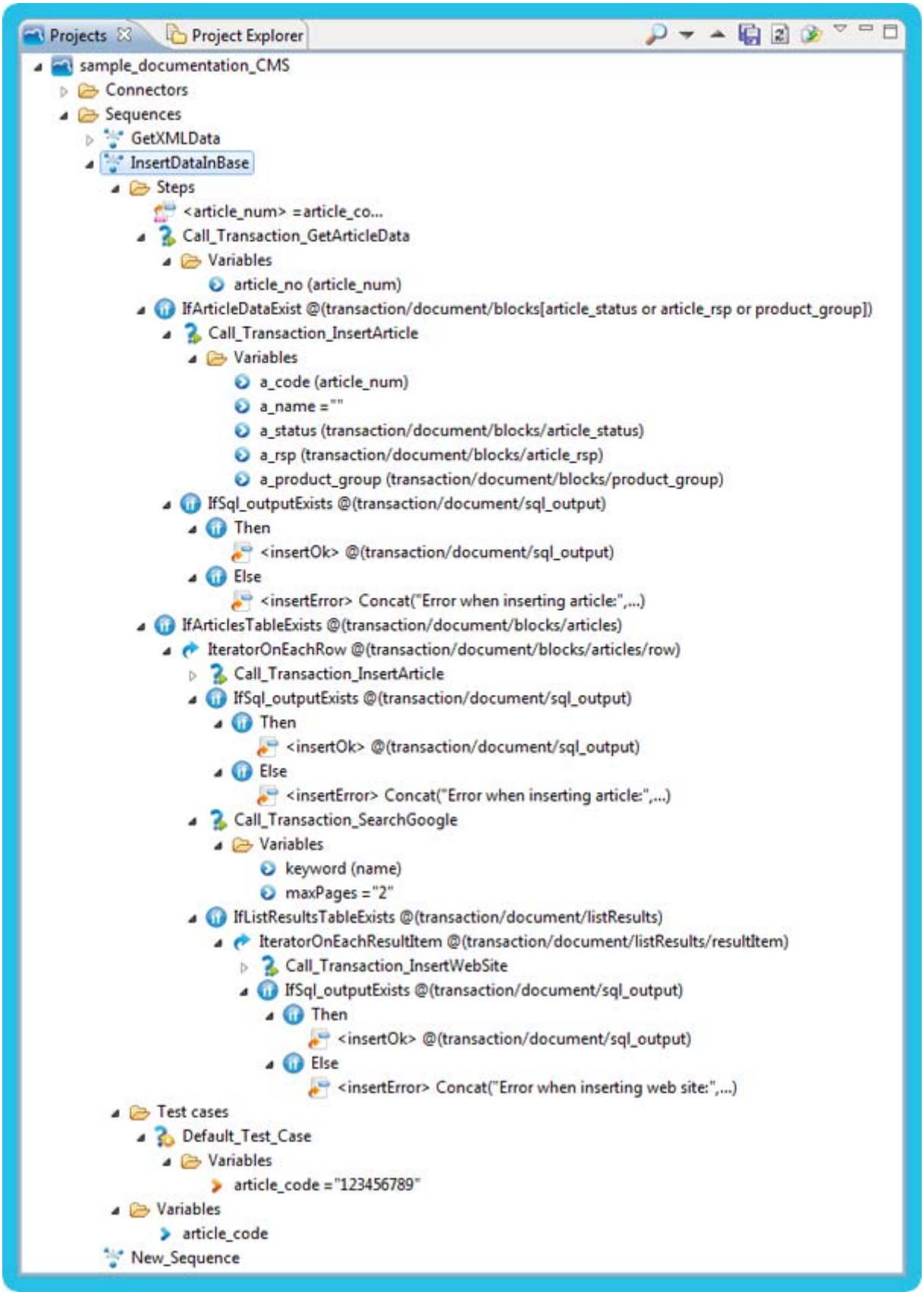


Figure 2 - 125: Generic Sequence - InsertDataInBase sequence in Projects view

Both sequences are set with similar property values. These appear as follows in the **Properties** view of the Convertigo Studio:

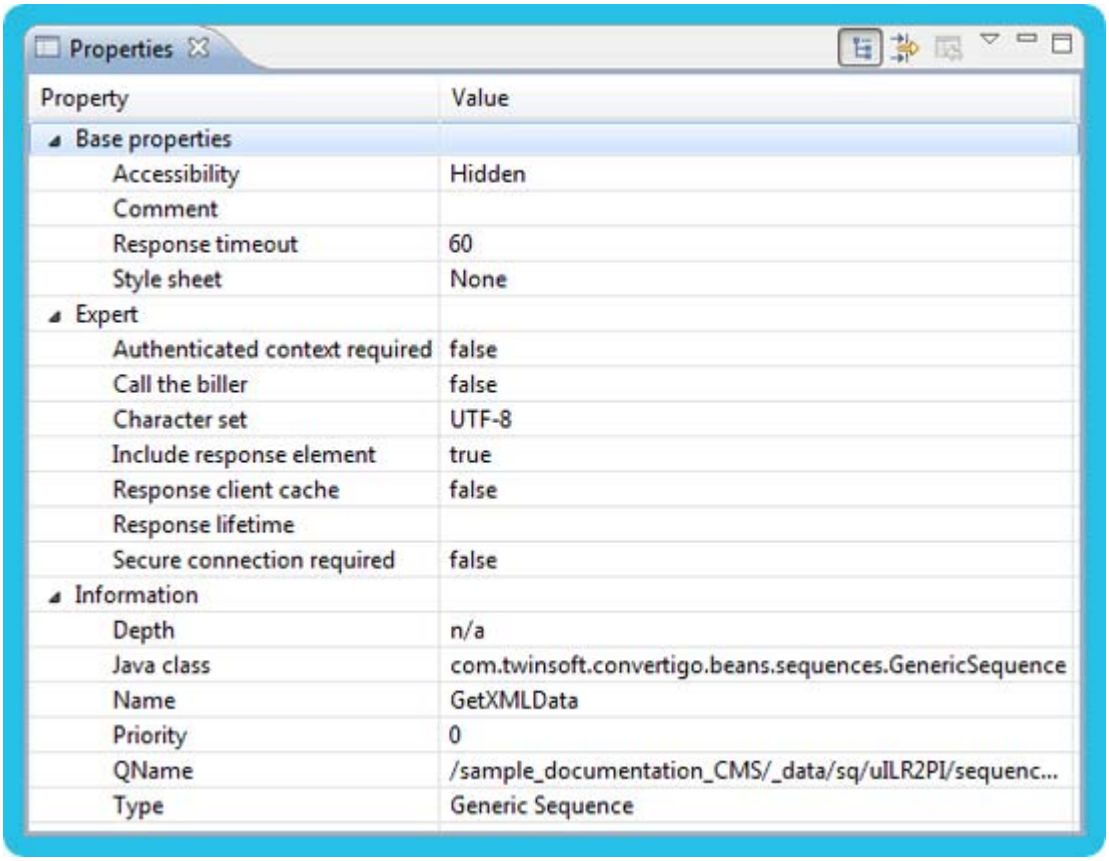


Figure 2 - 126: Generic Sequence - GetXMLData sequence configuration example

2.3.2 Steps

FLOW CONTROL STEPS

Object Description

Defines an `IF` conditional step based on a JavaScript condition.

The *jIf* step is one of Convertigo Sequencer conditional steps. This step is based on a JavaScript condition and contains other steps executed only if the condition is fulfilled.

The condition, defined in the **Condition** property, is a JavaScript expression that is evaluated during the sequence execution as `true` or `false`. If the condition is considered `true`, then steps under the parent *jIf* step are executed. If the condition is considered `false`, the steps under the parent *jIf* step are not executed.

Note: In Convertigo Studio, when an *jIf* step is created in a sequence, it can be easily replaced by an *jIfThenElse*, using the right-click menu on the step and choosing the option **Change to** > **jIfThenElse**. The **Condition** property remains the same and the steps present in the *jIf* are moved to the *Then* sub-step.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child steps. JavaScript variables and code are supported in this property.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

JIFTHENELSE 

OBJECT DESCRIPTION

Defines an `IF...THEN...ELSE...` conditional step based on a JavaScript condition.

The *jIfThenElse* step is one of the Convertigo Sequencer conditional steps. This step is based on a JavaScript condition and contains two child steps (*Then* and *Else*) which are executed depending on the condition fulfillment:

- `Then` step and child steps are executed when the condition is verified,
- `Else` step and child steps are executed when the condition is not verified.

The condition, defined in the **Condition** property, is a JavaScript expression that is evaluated during the sequence execution as `true` or `false`.

Note: In Convertigo Studio, when an *jIfThenElse* step is created in a sequence, it can be easily replaced by an *jIf*, using the right-click menu on the step and choosing the option **Change to** > **jIf**. The **Condition** property remains the same and the steps present in the sub-steps are:

- steps present in the *Then* step are moved to the *jIf*,
- steps present in the *Else* step are deleted.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child steps. JavaScript variables and code are supported in this property.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

IfExist 

Object Description

Defines an `IF` conditional step looking for node(s) on a source.

The *IfExist* step is one of Convertigo Sequencer conditional steps. This step contains other steps executed only if the source defined through the **Source** property exists.

Note: In Convertigo Studio, when an *IfExist* step is created in a sequence, it can be easily replaced by an *IfExistThenElse*, using the right-click menu on the step and choosing the option **Change to** > **IfExistThenElse**. The **Source** property remains the same and the steps present in the *IfExist* are moved to the *Then* sub-step.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to work on. This property allows defining a list of nodes from a previous step on which current step performs tests. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: the test fails.

Examples

Let's consider the `GetXMLData` sequence set in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains an *IfExist* step called `IfArticlesTableExists` which purpose is to check if the `articles` XML table exists in the XML output of the previously called `GetArticleData` transaction. If so, a *Complex* step is executed and generates an `articlesList` XML element serving as source for further steps. If not, the sequence ends because no other step is defined after the *IfExist* step.

The `IfArticlesTableExists` step appears as follows:

- in the **Projects** view of the Convertigo Studio:

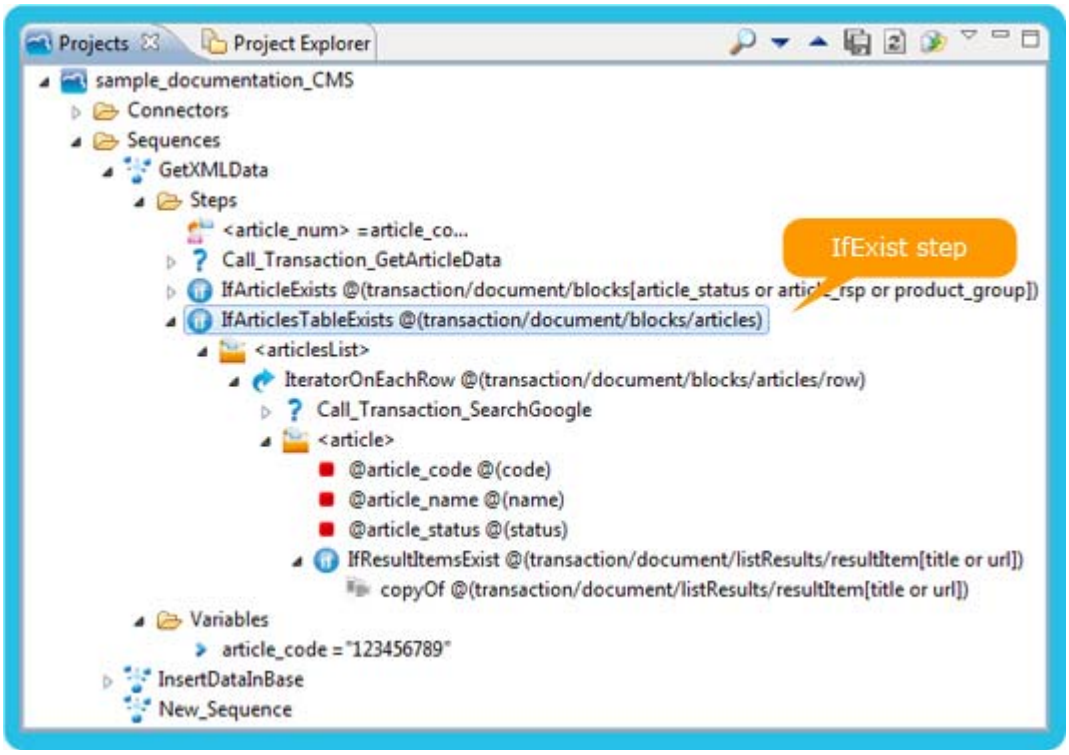


Figure 2 - 127: IfExist step - IfArticlesTableExists step in GetXMLData sequence

- in the **Properties** view of the Convertigo Studio:

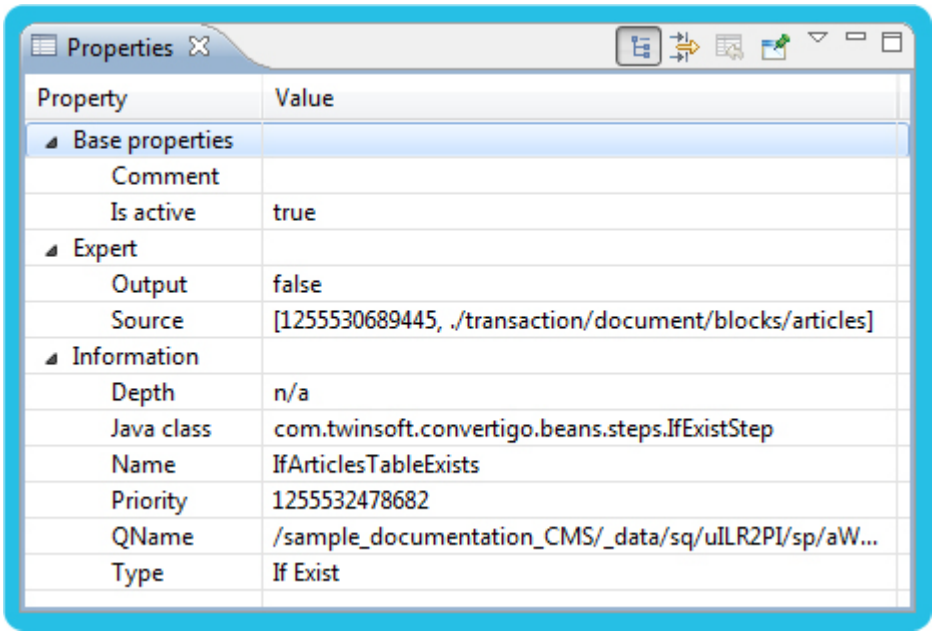


Figure 2 - 128: IfExist step - IfArticlesTableExists step properties

The **Ouput** property is set to `false` because no XML output is needed from the *IfExist* step. Only XML elements generated by the child steps (*Complex* step and others) are needed in the sequence XML output. The **Source** property points towards the `articles` node in the `GetArticleData` transaction XML schema retrieved by the `Call_Transaction_GetArticleData` step:

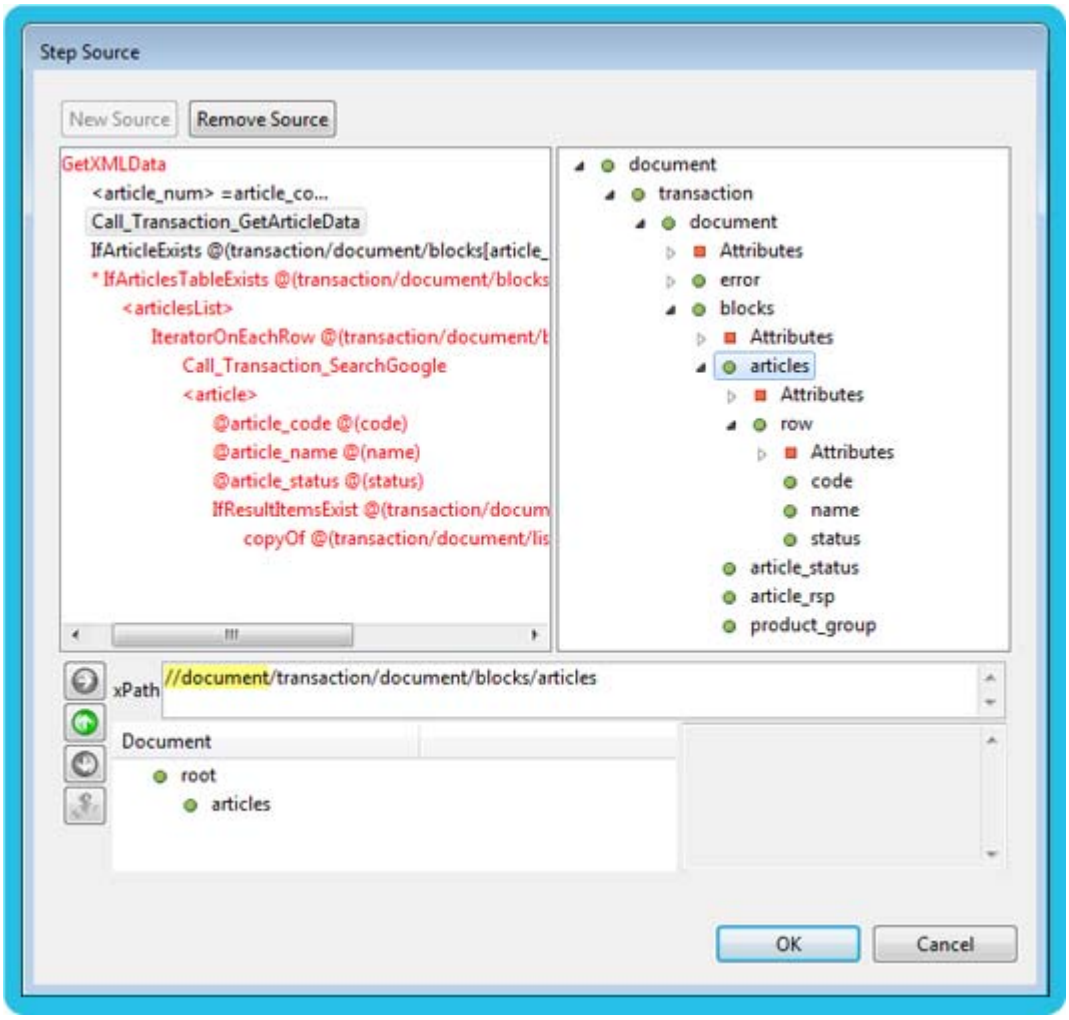


Figure 2 - 129: IfExist step - IfArticlesTableExists step source

IfExistThenElse 

Object Description

Defines an `IF...THEN...ELSE...` conditional step looking for node(s) on a source.

The *IfExistThenElse* step is one of Convertigo Sequencer conditional steps. This step contains two child steps (*Then* and *Else*) which are executed depending on whether the source defined through the **Source** property exists or not:

- *Then* step and child steps are executed when the specified source exists.
- *Else* step and child steps are executed when the specified source does not exist.

Note: In Convertigo Studio, when an *IfExistThenElse* step is created in a sequence, it can be easily replaced by an *IfExist*, using the right-click menu on the step and choosing the option **Change to** > **IfExist**. The **Source** property remains the same and the steps present in the sub-steps are:

- steps present in the *Then* step are moved to the *IfExist*,
- steps present in the *Else* step are deleted.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to work on. This property allows defining a list of nodes from a previous step on which current step performs tests. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: the test fails.

EXAMPLES

Let's consider the `GetXMLData` sequence set in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains an *IfExistThenElse* step called `IfArticleExists` which purpose is to check if a number of XML elements (`article_status`, `article_rsp`, `product_group`) exist in the XML output of the previously called `GetArticleData` transaction. Then, if at least one of the sourced elements exists, a *Complex* step is executed and an `articleFound` XML element is generated. Else, a *Complex* step is executed and an `articleNotFound` XML element is generated.

The `IfArticleExists` step appears as follows:

- in the **Projects** view of the Convertigo Studio:

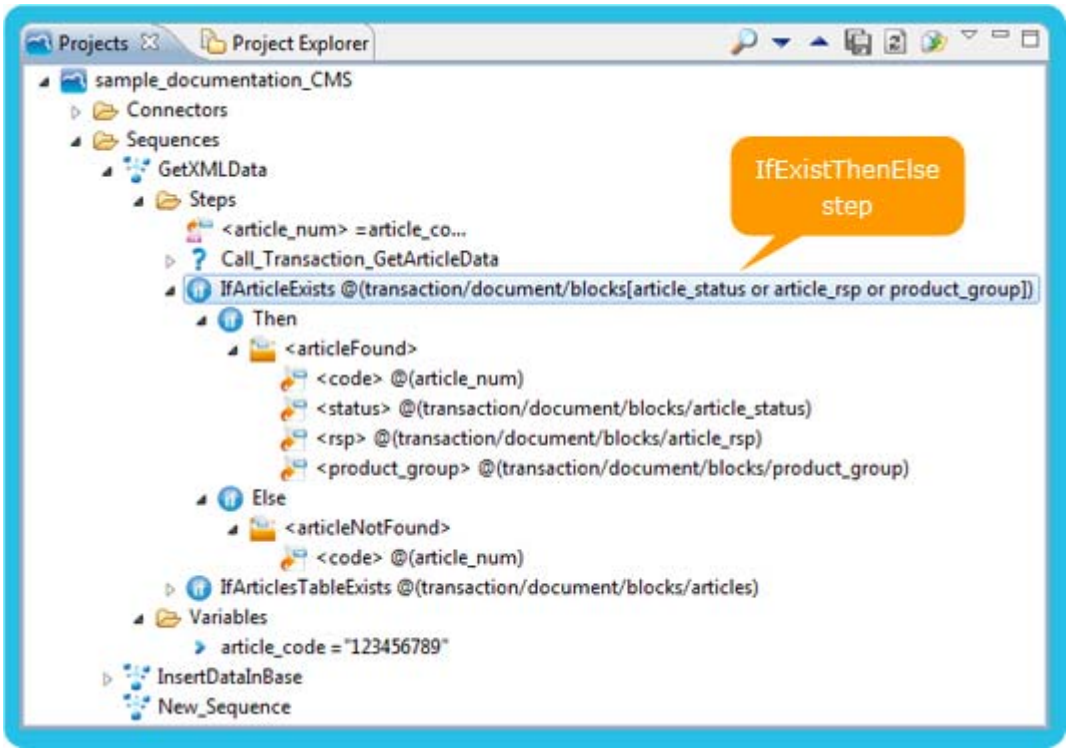


Figure 2 - 130: IfExistThenElse step - IfArticleExists step in GetXMLData sequence

- in the **Properties** view of the Convertigo Studio:

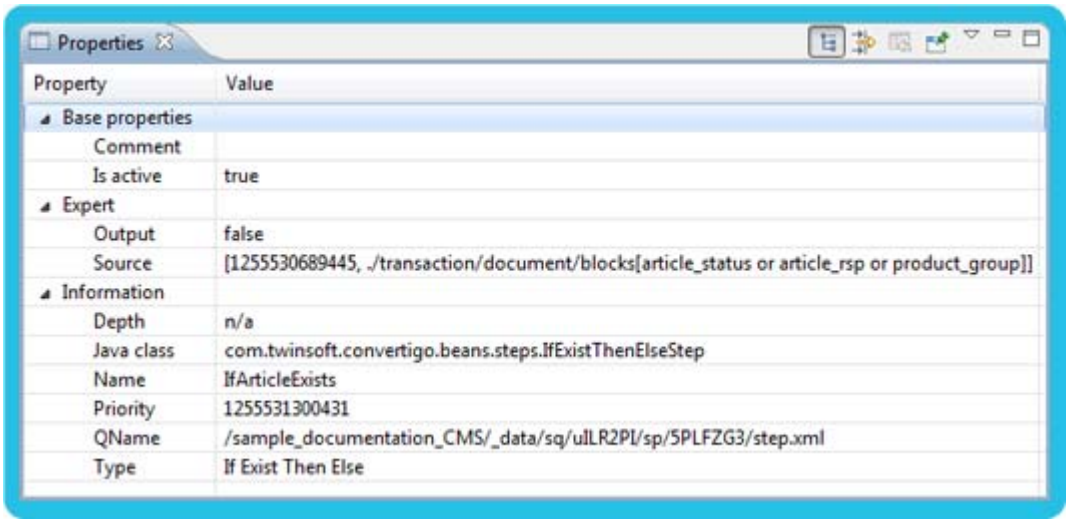


Figure 2 - 131: IfExistThenElse step - IfArticleExists step properties

The **Ouput** property is set to `false` because no XML output is needed from the *IfExistThenElse* step. Only XML elements generated by the child steps (*Then* and *Else*) are needed in the sequence XML output. The **Source** property points towards the nodes, in the `GetArticleData` transaction XML schema retrieved by the `Call_Transaction_GetArticleData` step, serving as basis for the `IfArticleExists` step "if" condition (i.e, "If the `article_status` tag OR the `article_rsp` tag OR `product_group` tag exist"):

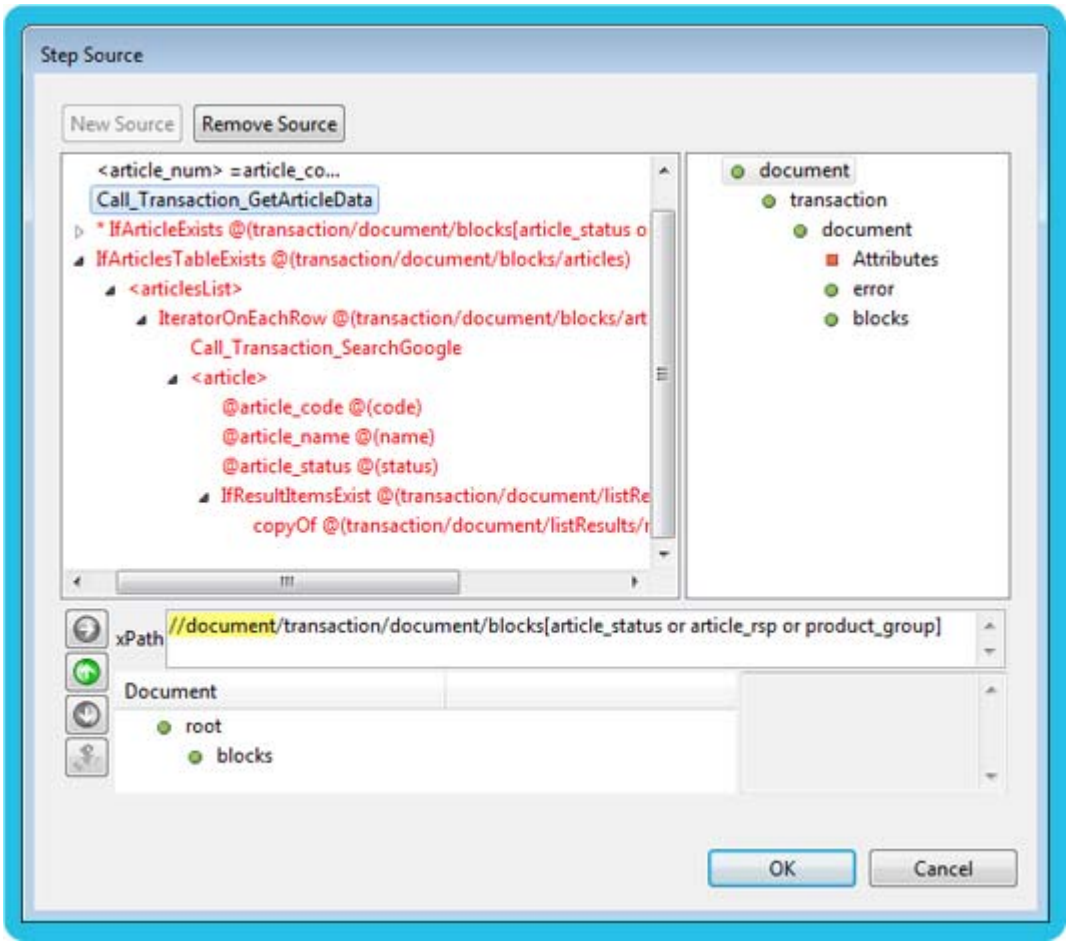


Figure 2 - 132: IfExistThenElse step - IfArticleExists step sources

IfIsIn 

Object Description

Defines an `IF` conditional step looking for matches on a source.

The *IfIsIn* step is one of Convertigo Sequencer conditional steps. This step is based on a source and one or more regular expression(s) called "Tests". Child steps are executed only if the specified source exists and if tests match on that specified source.

Note: In Convertigo Studio, when an *IfIsIn* step is created in a sequence, it can be easily replaced by an *IfIsInThenElse*, using the right-click menu on the step and choosing the option **Change to** > **IfIsInThenElse**. The **Source** and **Tests** properties remain the same and the steps present in the *IfIsIn* are moved to the *Then* sub-step.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to work on. This property allows defining a list of nodes from a previous step on which current step performs tests. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: the test fails.

Property	Type	Category	Description
Tests	XMLVector	expert	Defines match tests as regular expressions. This property allows to define a list of tests that are applied on the source result. For each test, two elements have to be set: • **Operator**: value to choose between `AND` and `NOT`, the operator value is applied on the regular expression result to keep it (`AND`) or to inverse it (`NOT`). • **Regular exp**: defines a regular expression to apply (inverted or not thanks to operator value) on the source result. Notes: • A new test can be added to the list using the blue keyboard icon. The tests defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • In order to be able to test the regular expressions on the source result, the defined source has to select a text node. • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

IfIsInThenElse 

Object Description

Defines an `IF...THEN...ELSE...` conditional step looking for matches on a source.

The *IfIsInThenElse* step is one of the Convertigo Sequencer conditional steps. This step is based on a source and one or more regular expression(s) called "Tests". This step contains two child steps (*Then* and *Else*) which are executed depending on whether the specified source exists and if tests match on that specified source or not:

- *Then* step and child steps are executed when specified source exists and tests match on source,
- *Else* step and child steps are executed when specified source exists and tests do not match on source or when specified source does not exist.

Note: In Convertigo Studio, when an *IfIsInThenElse* step is created in a sequence, it can be easily replaced by an *IfIsIn*, using the right-click menu on the step and choosing the option **Change to** > **IfIsIn**. The **Source** and **Tests** properties remain the same and the steps present in the sub-steps are:

- steps present in the *Then* step are moved to the *IfIsIn*,
- steps present in the *Else* step are deleted.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Property	Type	Category	Description
Source	XMLVector	expert	Defines the source to work on. This property allows defining a list of nodes from a previous step on which current step performs tests. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: the test fails.
Tests	XMLVector	expert	Defines match tests as regular expressions. This property allows to define a list of tests that are applied on the source result. For each test, two elements have to be set: • **Operator**: value to choose between `AND` and `NOT`, the operator value is applied on the regular expression result to keep it (`AND`) or to inverse it (`NOT`). • **Regular exp**: defines a regular expression to apply (inverted or not thanks to operator value) on the source result. Notes: • A new test can be added to the list using the blue keyboard icon. The tests defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • In order to be able to test the regular expressions on the source result, the defined source has to select a text node. • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

JWHILE 

OBJECT DESCRIPTION

Defines a `WHILE` loop step based on a JavaScript condition.

This step executes a group of child steps as the condition expression set in the **Condition** property remains true.

Note: You can add other steps to this step: these are the steps executed in the loop.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child steps. JavaScript variables and code are supported in this property.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

JDoWhile 

Object Description

Defines a `DO...WHILE` loop step based on a JavaScript condition.

This step executes a group of child steps once, then repeats execution of the loop until the condition expression set in the **Condition** property is found to be false.

Note: You can add other steps to this step: these are the steps executed in the loop.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child steps. JavaScript variables and code are supported in this property.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

ITERATOR 

OBJECT DESCRIPTION

Defines a loop step iterating on XML nodes result from a source.

Also called *For Each* step, the *Iterator* step:

- defines a source as input list to work on, i.e. a list of nodes from a previous step, used as a recurring element (for example table rows),
- iterates on each element of the specified source,
- contains child steps that are executed on each iteration, as other loop steps (for example, see "*jIterator*", "*jWhile*" and "*jDoWhile*" steps documentation and examples).

In the iteration, child steps can access and use the current iterated element:

- using a source pointing on the *Iterator* step itself,
- using the JavaScript variable named `item`, which is a Java Node object (item of the NodeList resulting from the input source).

They also can access the current iteration index using the JavaScript variable named `index` updated on each iteration, which is an integer.

Note: The current item value can be accessed using the following code statement:

- `item.getTextContent()` if the Node is of Text or Attribute type,
- `item.getNodeValue()` if the Node is of Element type.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Max. iterations	JS expression	standard	Defines the maximum number of iterations. Intended mostly for testing purposes, this (optional) parameter limits the number of times the iterator loops. This property is a JavaScript expression that is evaluated during the sequence execution. By default, it is not filled, so the *Iterator* loops on each node from the source.

Property	Type	Category	Description
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source list to iterate on. This property allows defining a list of nodes from a previous step on which current step works. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: the loop does not execute its child steps and the parent sequence execution continues.
Starting index	JS expression	standard	Defines the index from which the *Iterator* should start to iterate. In the case you do not want to start an iteration at the first item (index `0`), you can specify a starting index in this property. This property is a JavaScript expression that is evaluated during the sequence execution. By default, it is set to `0` for starting at the first item of the source list. If the defined starting index does not exist in the source list, the loop does not execute its child steps and the parent sequence execution continues.

EXAMPLES

Let's consider the `GetXMLData` sequence set in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains an *Iterator* step called `IteratorOnEachRow` which purpose is to:

- *iterate* on each row of the `articles` table generated by the `GetArticleData` transaction,
- *execute* child steps on each of these rows.

The `IteratorOnEachRow` step appears as follows:

- in the **Projects** view of the Convertigo Studio:

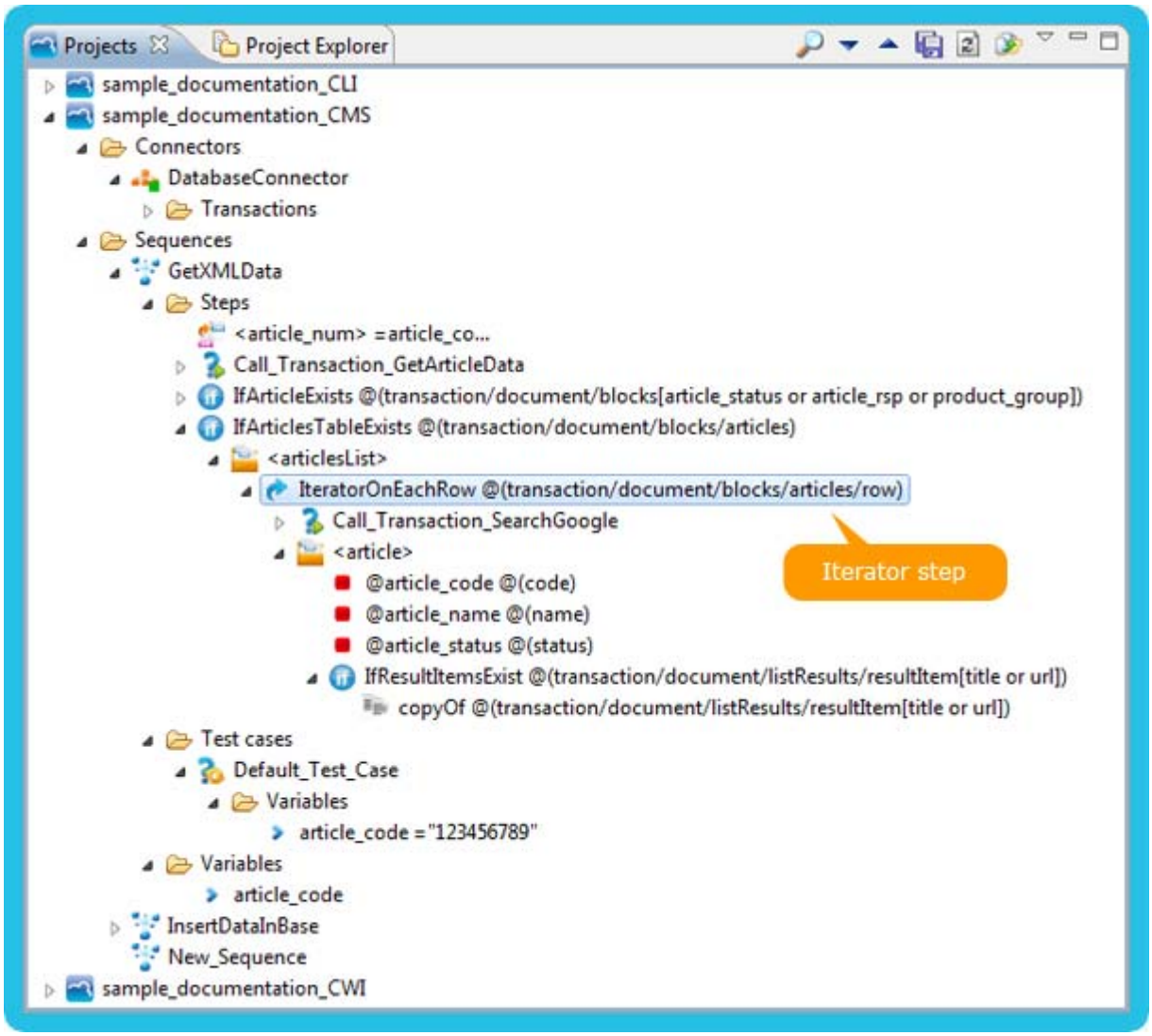


Figure 2 - 133: Iterator step - IteratorOnEachRow step in GetXMLData sequence

- in the **Properties** view of the Convertigo Studio:

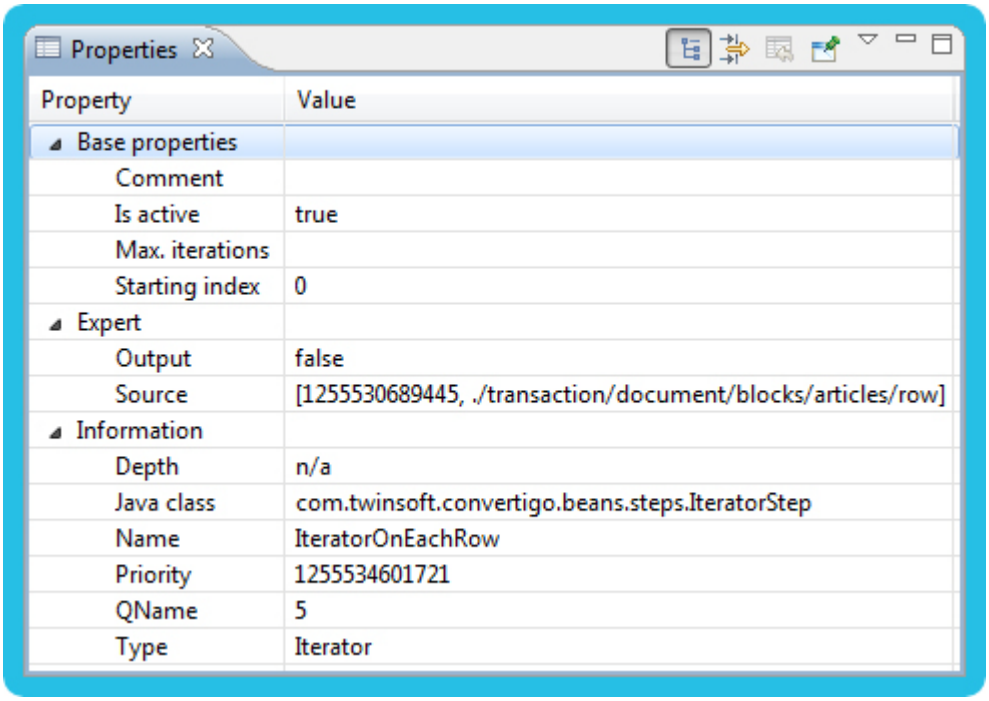


Figure 2 - 134: Iterator step - Configuration example

The **Ouput** property is set to `false` because no XML output is needed from the *Iterator* step. Only XML elements generated for each iteration, i.e. by *Iterator* step's child steps (*Call Transation, Complex, Attribute steps,* see Figure 2 - 133), are needed in the sequence XML output.

The **Source** property points towards the node on which the step is to iterate (an `articles` child element called `row`). This node belongs to the `GetArticleData` transaction XML schema retrieved by the `Call_Transaction_GetArticleData` step:

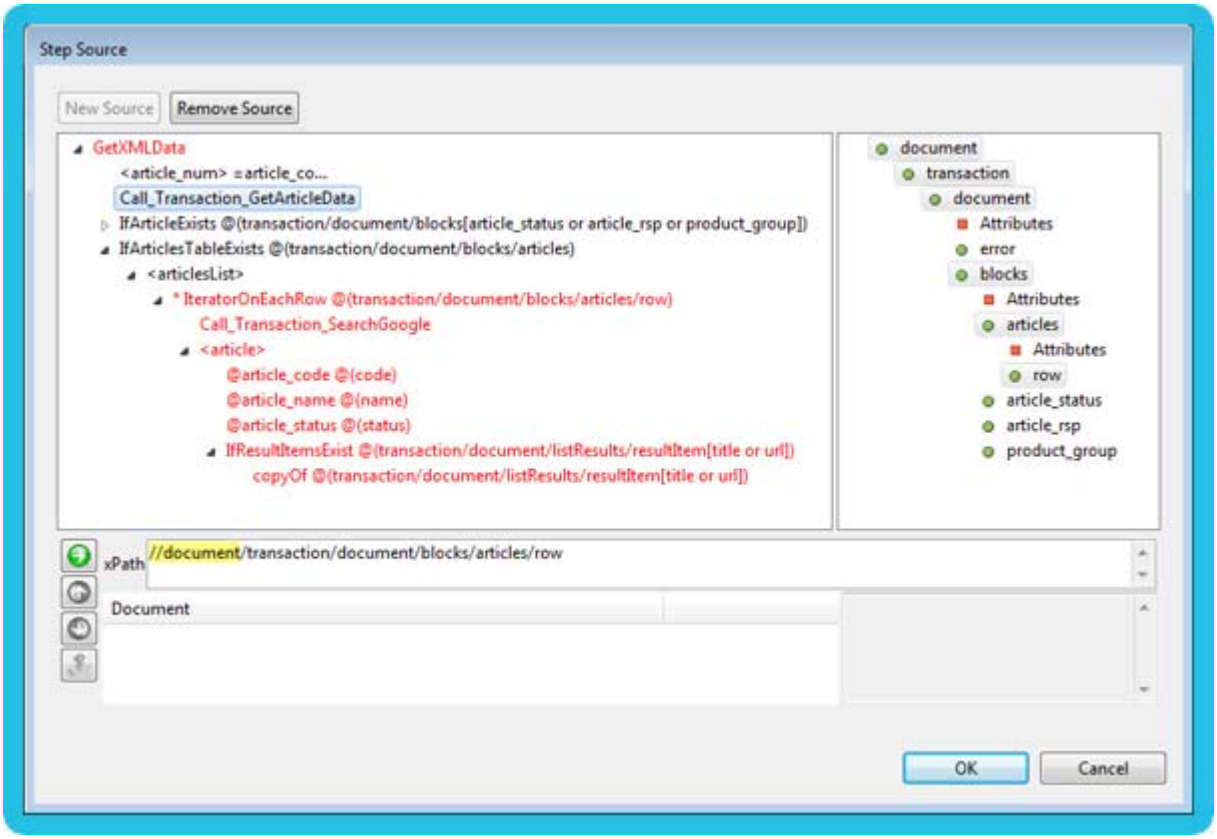


Figure 2 - 135: Iterator step - Source configuration

When executing the `Default_Test_Case` test case defined for the sequence, the Iterator step iterates on each row element from the `GetArticleData` transaction, calls the `searchGoogleWithLimit` transaction passing as input variable the `name` value of the iterated row and creates an XML output structure including data from both transactions execution (the current row and the associated Google search response).

JITERATOR 

OBJECT DESCRIPTION

Defines a loop step iterating on list items result from a JavaScript expression.

Also called *For Each* step, the *jIterator* step:

- defines a JavaScript expression as input list to work on, i.e. the name of a multi-valued variable, the name of a defined JavaScript Array, or the name of a NodeList variable created by a previous *jSource* step, etc.,
- iterates on each item of the specified input list,
- contains child steps that are executed on each iteration, as other loop steps (for example see "*Iterator*", "*jWhile*" and "*jDoWhile*" steps documentation and examples).

In the iteration, child steps can access and use:

- the current iterated item through a JavaScript variable named `item`, which type depends on the iterated Array or NodeList,
- the current iteration index through a JavaScript variable named `index`, which is an integer.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the list to iterate on. This property is a JavaScript expression that is evaluated during the sequence execution and gives a list of items (JavaScript `Array` or `NodeList`). If the expression doesn't output a list object or if the expression is left blank, the step has no data to work on: the loop does not execute its child steps and the parent sequence execution continues
Is active	boolean	standard	Defines whether the step is active.
Max. iterations	JS expression	standard	Defines the maximum number of iterations. Intended mostly for testing purposes, this (optional) parameter limits the number of times the iterator loops. By default, it is not filled, so the *jIterator* loops on each item from the list.

Property	Type	Category	Description
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Starting index	JS expression	standard	Defines the index from which the *jIterator* should start to iterate. In the case you do not want to start an iteration at the first item (index `0`), you can specify a starting index in this property. This property is a JavaScript expression that is evaluated during the sequence execution. By default, it is set to `0` for starting at the first item of the input list. If the defined starting index does not exist in the input list, the loop does not execute its child steps and the parent sequence execution continues.

EXAMPLES

Let's consider a `IterateOnJS` sequence, which defines one single-valued variable named `startIndex` and one multi-valued variable named `table`. This sequence iterates on the elements of the `table` variable, starting to iterate at the index `startIndex`, and generates an XML output out of it.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Sequencer objects examples*** *in the* ***New Project*** *wizard.*
Associated project (not mandatory for running this example) can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

In order to iterate on the multi-valued variable, which is a JavaScript Array, a *jIterator* step is created with the following parameters:

```
jIterator [
  expression: table
  startingIndex: startIndex
  output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

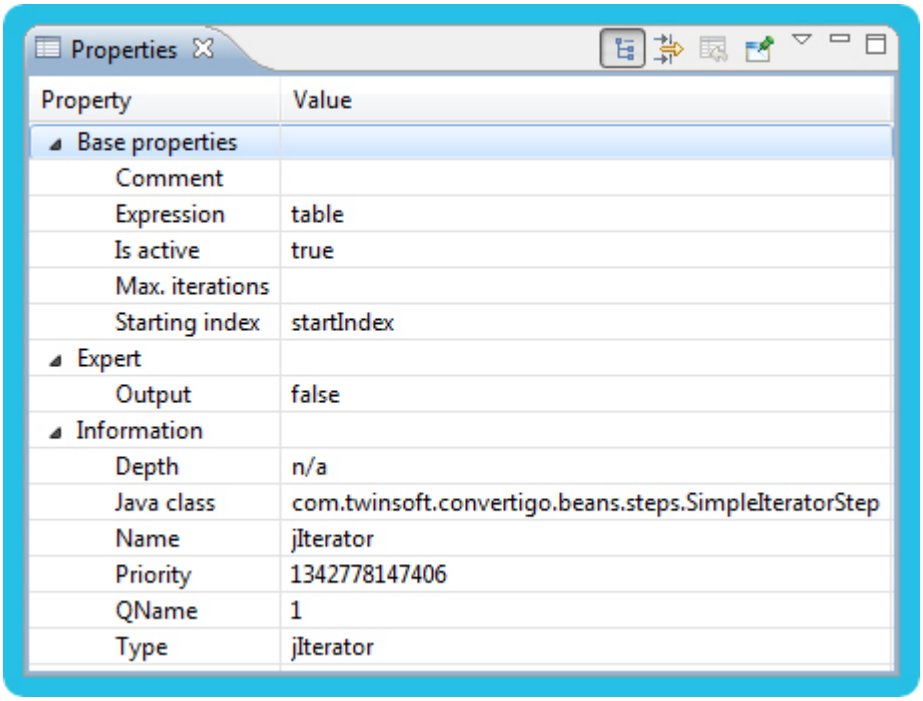


Figure 2 - 136: jIterator step - Configuration example

The **Expression** property is set to a JavaScript expression using the variable `table`, on which we want to iterate. This variable is set by default to a set of five values on which to iterate.

The **Starting index** property is set to a JavaScript expression using the variable `startIndex`, this variable is set by default to `0`.

The step is created in the **Steps** folder of the sequence, including various other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

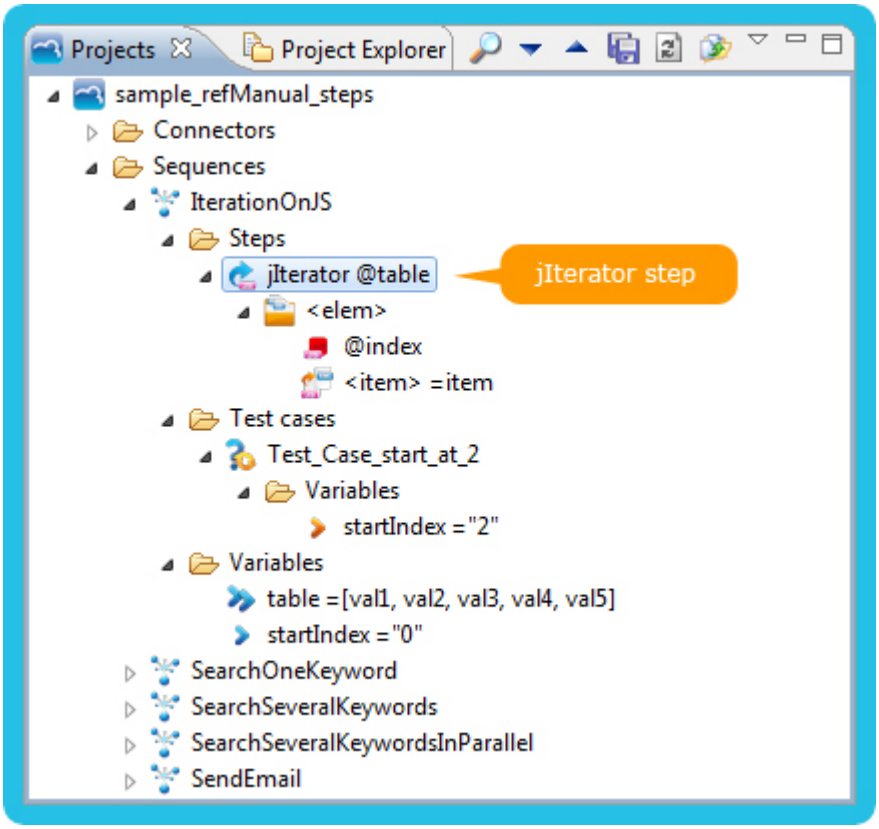


Figure 2 - 137: jIterator step - Object in Projects view, with sequence and other steps

Run the sequence directly with the default values or run the `Test_Case_start_at_2` test case defined for the sequence. You should see the XML output of the sequence being different depending on the `startIndex` variable value. For example, here is the sequence output result in sequence editor after executing the test case:

```
<document connector="" context="studio_sample_ref
  <elem index="2">
    <item>val3</item>
  </elem>
  <elem index="3">
    <item>val4</item>
  </elem>
  <elem index="4">
    <item>val5</item>
  </elem>
</document>
```

Figure 2 - 138: jIterator step - XML result of the sequence after execution

RETURN (SEQUENCER) 

OBJECT DESCRIPTION

Defines a RETURN step.

A *Return* steps exits the current sequence in which it is positioned.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

jBreak 

Object Description

Defines a BREAK step.

A *jBreak* step executes a JavaScript expression and exits the current loop step.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the step value. This property is a JavaScript expression that is evaluated during the sequence execution and gives the step's result.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Examples

Let's consider a `SearchOneKeyword` sequence, which defines two variables named `input` and `maxResult.` This sequence calls the `searchGoogleWithLimit` transaction, set in the context of the *"Starting With Convertigo Web Integrator"* tutorial, with its `input` variable as `keyword` and a hard-coded `maxPages` variable value. The sequence then tests whether the transaction has returned an error message or not. In the first case, it ends by raising an Exception, in the second case, it iterates on the results list and copies to sequence output the `maxResult` first items.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Sequencer objects examples*** *in the* ***New Project*** *wizard.*
Associated project can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

In order to exit the loop when `maxResult` result items are copied to the sequence's output, a

jBreak step is created with the following parameters:

```
jBreak [
  expression: <empty>
  output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

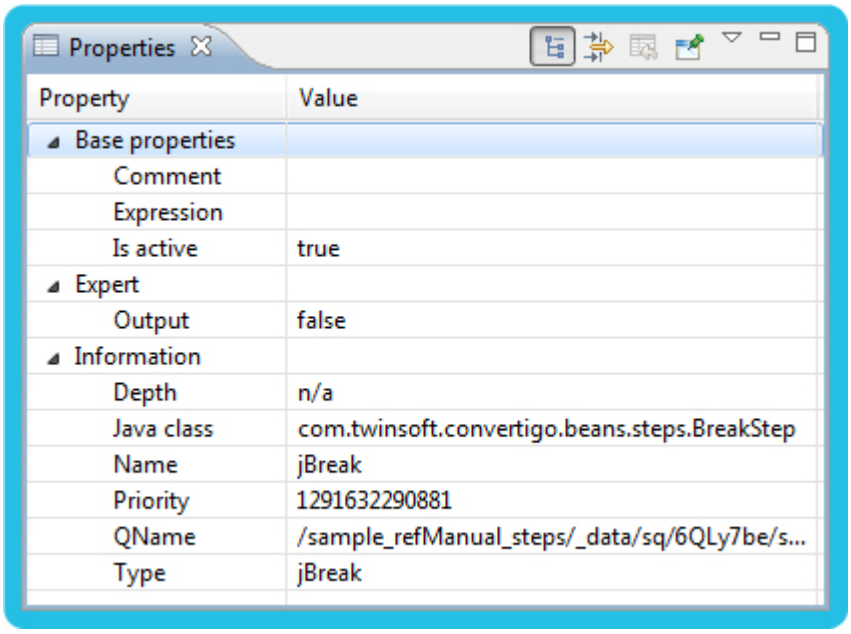


Figure 2 - 139: jBreak step - Configuration example

The **Expression** property is left empty as no JavaScript code needs to be executed before exiting the loop.

The step is created in the **Steps** folder of the sequence, under the loop step and next to various other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

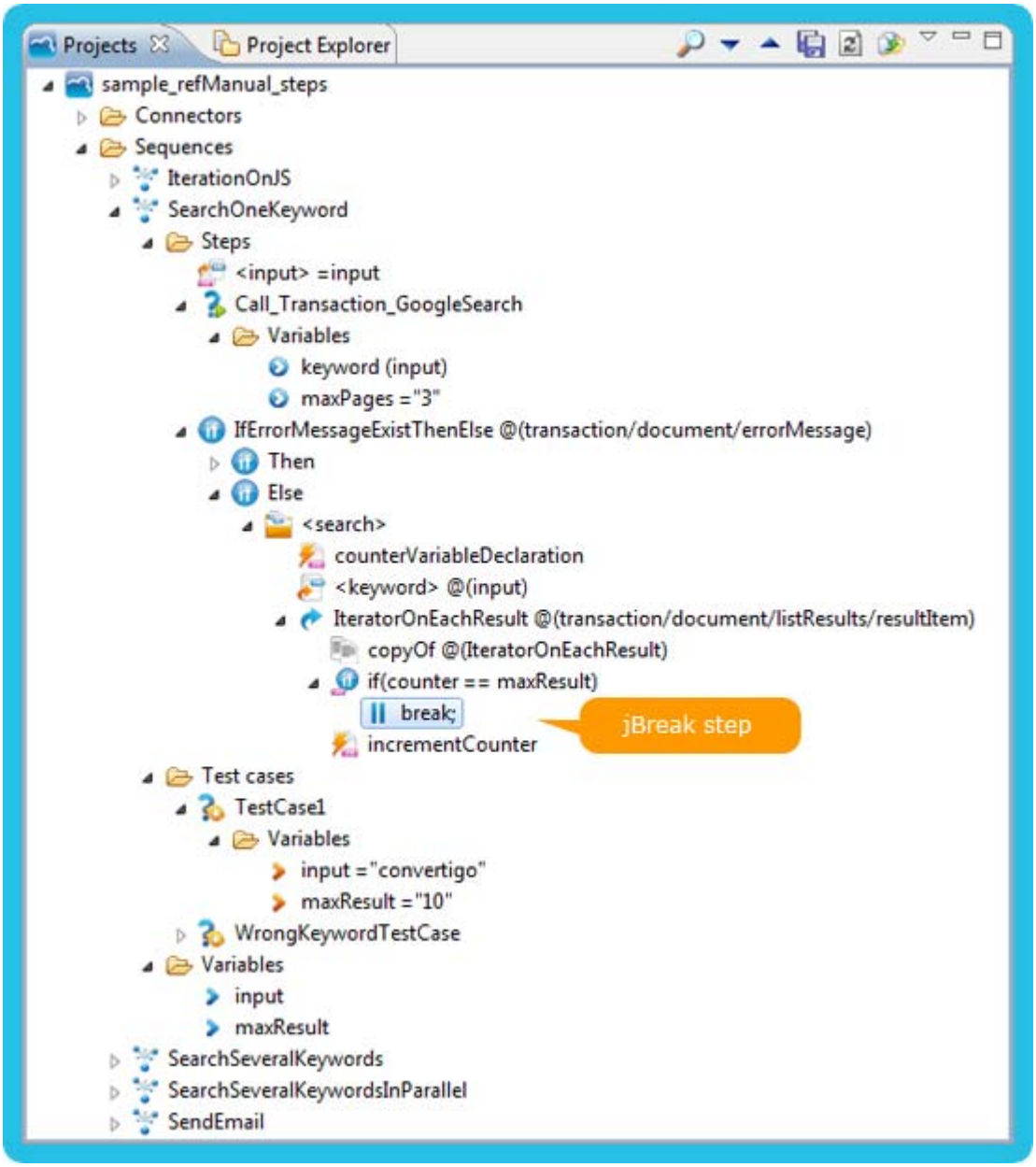


Figure 2 - 140: jBreak step - Object in Projects view, with sequence and other steps

When executing the `TestCase1` test case defined for the sequence, with `input` to "convertigo" and `maxResult` to "10", the sequence calls the `searchGoogleWithLimit` transaction with `input` variable passed as `keyword` and `maxPages` to fixed value "3". Then, the result items are copied to the sequence output XML. When arriving at the `maxResult` iteration, the sequence copies the item and then exits the loop thanks to the *jBreak* step.

After execution, the sequence XML output contains the 10 result items that have been copied:

```
<document connector="" context="studio_sample_refManual_steps:SearchOneKeyword" conte
  <search>
    <keyword>convertigo</keyword>
    <resultItem>
      <title>Convertigo - Enterprise Mashups</title>
      <url>www.convertigo.com/</url>
    </resultItem>
    <resultItem>
      <title>About Convertigo and their Enterprise Mashup Server</title>
      <url>www.convertigo.com/en/about-us.html</url>
    </resultItem>
    <resultItem>
      <title>Convertigo Shared Cloud: Secure and Flexible</title>
      <url>www.convertigo.com/en/overview/convertigo-cloud.html</url>
    </resultItem>
    <resultItem>
      <title>Convertigo Enterprise Mashup Server</title>
      <url>download.convertigo.com/webrepository/c-ems-datasheet.pdf</url>
    </resultItem>
    <resultItem>
      <title>Convertigo (convertigo) on Twitter</title>
      <url>twitter.com/convertigo</url>
    </resultItem>
    <resultItem>
      <title>I-Cubed - Bridging the Enterprise</title>
      <url>www.i-cubed.com/enterprise-mashups.html</url>
    </resultItem>
    <resultItem>
      <title>Convertigo - Company Profile | LinkedIn</title>
      <url>www.linkedin.com/companies/convertigo</url>
    </resultItem>
    <resultItem>
      <title>Convertigo in SOA environment</title>
      <url>www.slideshare.net/convertigo/convertigo-in-soa-environment</url>
    </resultItem>
    <resultItem>
      <title>Convertigo Websphere Siteminder - Convertigo</title>
      <url>convertigo.fr/...convertigo.../117-convertigo-websphere-siteminder.html</url>
    </resultItem>
    <resultItem>
      <title>Videos for convertigo</title>
      <url>youtube.com</url>
      <url>youtube.com</url>
    </resultItem>
    <resultItem>
      <title>Convertigo Salesforce | Computer Detail Reviews</title>
      <url>computerdetailreviews.com/reviews-convertigo-salesforce</url>
    </resultItem>
  </search>
</document>
```

Figure 2 - 141: jBreak step - Resulting XML after executingSearchOneKeyword sequence

SERIAL 

OBJECT DESCRIPTION

Defines a step executing child steps in series.

All child steps of a *Serial* step are executed one after another, it is similar to the basic behavior of step execution when they are positioned just under the parent *Generic Sequence*.

A *Serial* step is completed (i.e. the sequence will continue flow execution) when all child steps have been completed. This means the step following a *Serial* step starts right after the last child step is completed.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

PARALLEL 

OBJECT DESCRIPTION

Defines a step executing child steps in parallel.

A *Parallel* step executes steps simultaneously in parallel contexts. The maximum number of contexts is set by the value of the **Max. threads** property.

Each child step is executed in a dedicated thread. When a child thread is completed, all of its resources are released. As a consequence, a step defined outside a *Parallel* step cannot source any information from it.

To do so, it is recommended that you:

- create a *Complex* step as a parent of the *Parallel* step,
- generate information from the *Parallel* step into the *Complex* step,
- use the *Complex* step as a source outside the *Parallel* step.

A *Parallel* step is completed (i.e. the sequence will continue flow execution) when all child threads have been completed. This means the step following a *Parallel* step starts right after all child threads have been completed.

Convertigo contexts are created for each child step executed in parallel. These contexts are automatically named after parent *Parallel* step properties.

If *Call transaction* or *Call sequence* steps are child of a *Parallel* step, contexts can be named after their **Context** property or automatically if this property is not specified.

Every automatically named context will be deleted after the *Parallel* step execution is completed. Explicitly named contexts will remain for further transaction or sequence use.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Max. threads	JS expression	standard	Defines the maximum number of simultaneously processed threads. If this number is inferior to the number of child steps to be executed simultaneously, all child steps cannot start. In this case, **Max. thread** child steps start executing. The others wait for threads to be available.

Property	Type	Category	Description
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

IfFileExists 

Object Description

Defines an `IF` conditional step looking for the existence of a file or a directory.

The *IfFileExists* step is one of Convertigo Sequencer conditional steps. This step contains other steps executed only if the file or directory defined through the **Source** property exists.

Note: In Convertigo Studio, when an *IfFileExists* step is created in a sequence, it can be easily replaced by an *IfFileExistsThenElse*, using the right-click menu on the step and choosing the option **Change to** > **IfFileExistsThenElse**. The **Source** property remains the same and the steps present in the *IfFileExists* are moved to the *Then* sub-step.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	JS expression	standard	Defines the path of the file or directory which existence has to be checked. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory which existence has to be checked. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

IfFileExistsThenElse 

Object Description

Defines an `IF...THEN...ELSE...` conditional step looking for the existence of a file or a directory.

The *IfFileExistsThenElse* step is one of the Convertigo Sequencer conditional steps. This step contains two child steps (*Then* and *Else*) which are executed depending on whether the file or directory defined through the **Source** property exists or not:

- *Then* step and child steps are executed when the source file or directory exists.
- *Else* step and child steps are executed when the source file or directory does not exist.

Note: In Convertigo Studio, when an *IfFileExistsThenElse* step is created in a sequence, it can be easily replaced by an *IfFileExists*, using the right-click menu on the step and choosing the option **Change to** > **IfFileExists**. The **Source** property remains the same and the steps present in the sub-steps are:

- steps present in the *Then* step are moved to the *IfFileExists*,
- steps present in the *Else* step are deleted.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	JS expression	standard	Defines the path of the file or directory which existence has to be checked. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory which existence has to be checked. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

JAVASCRIPT STEPS

SEQUENCE JS 

OBJECT DESCRIPTION

Defines a scripting step.

This helpful step allows to handle JavaScript code that will be executed in the sequence scope. This JavaScript code is able to:

- initialize variables,
- perform complex calculations,
- access the context object to get useful properties such as `contextID`, `httpSession`, `isCacheEnabled`, `lockPooledContext`, etc.,
- use some context methods to manipulate the result XML DOM, encode and decode data, abort sequence, etc.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the step value. This property is a JavaScript expression that is evaluated during the sequence execution and gives the step's result.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

jSource 

Object Description

Defines a step extracting a list of nodes from a source into a variable in JavaScript scope.

The *jSource* step gets a list of nodes from the source defined in the **Source** property and sets a JavaScript variable in the current executed sequence JavaScript scope. This variable contains a `Java NodeList` object, i.e. a list of XML nodes get from the source.

The variable is named after the **Variable name** property value. It exists while the sequence is running.

If only one node matches, the variable is also a `NodeList` containing only one `Node` (index is 0). If no node matches, the variable is finally an empty `NodeList`, containing no `Node` (`var_name.getLength() = 0`).

Notes:

- The variable contains a list of node elements get from a previously executed step. To access one (`Node`) of the list, use the following syntax in a step: `var_name.item(index)`.
- To access one element's text content (`String`), use the `element.getTextContent()` method, to retrieve the text of the element, or the `element.getNodeValue()` method, which result depends on the node type (will extract a text only if the `Node` is of `Text` or `Attribute` type).

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Property	Type	Category	Description
Source	XMLVector	expert	Defines the source to extract. This property allows defining a list of nodes from a previous step that are set in a JavaScript variable, as described in the main description of this step. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, depending on the step, the variable is created: • as an empty `NodeList` with no data (for *jSource* step), • `null` (for *jSimpleSource* step).
Variable name	String	standard	Defines the name of the JavaScript variable. If this variable exists in scope, its value is overridden. If the variable doesn't exist in scope, it is created.

jSimpleSource 

Object Description

Defines a step extracting a string from a source into a variable in Javascript scope.

The *jSimpleSource* step gets a single node from the source defined in the **Source** property and sets a JavaScript variable in the current executed sequence JavaScript scope. This variable contains a `String`.

The variable is named after the **Variable name** property value. It exists while the sequence is running.

If no node matches, the variable is `null`.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to extract. This property allows defining a list of nodes from a previous step that are set in a JavaScript variable, as described in the main description of this step. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, depending on the step, the variable is created: • as an empty `NodeList` with no data (for *jSource* step), • `null` (for *jSimpleSource* step).
Variable name	String	standard	Defines the name of the JavaScript variable. If this variable exists in scope, its value is overridden. If the variable doesn't exist in scope, it is created.

EXAMPLES

Let's consider a `SearchOneKeyword` sequence, which defines two variables named `input` and `maxResult.` This sequence calls the `searchGoogleWithLimit` transaction, set in the context of the *"Starting With Convertigo Web Integrator"* tutorial, with its `input` variable as `keyword` and a hard-coded `maxPages` variable value. The sequence then tests whether the transaction has returned an error message or not. In the first case, it ends by raising an Exception, in the second case, it iterates on the results list and copies to sequence output the `maxResult` first items.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Sequencer objects examples*** *in the* ***New Project*** *wizard.*
Associated project can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

When an error message is found in the transaction output XML, a *jException* step is created, using JavaScript expressions as input values for the exception's message and details (for more information about this step, see "*jException* step" documentation and examples). The error message from the transaction XML response has to be transferred to a JavaScript variable in order to be used in the *jException* step. To do so, a *jSimpleSource* step is created with the following parameters:

```
jSimpleSource [
   variable name: errorMessage
   source: [
      Call_Transaction_GoogleSearch step,
      //document/transaction/document/errorMessage
   ]
   output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

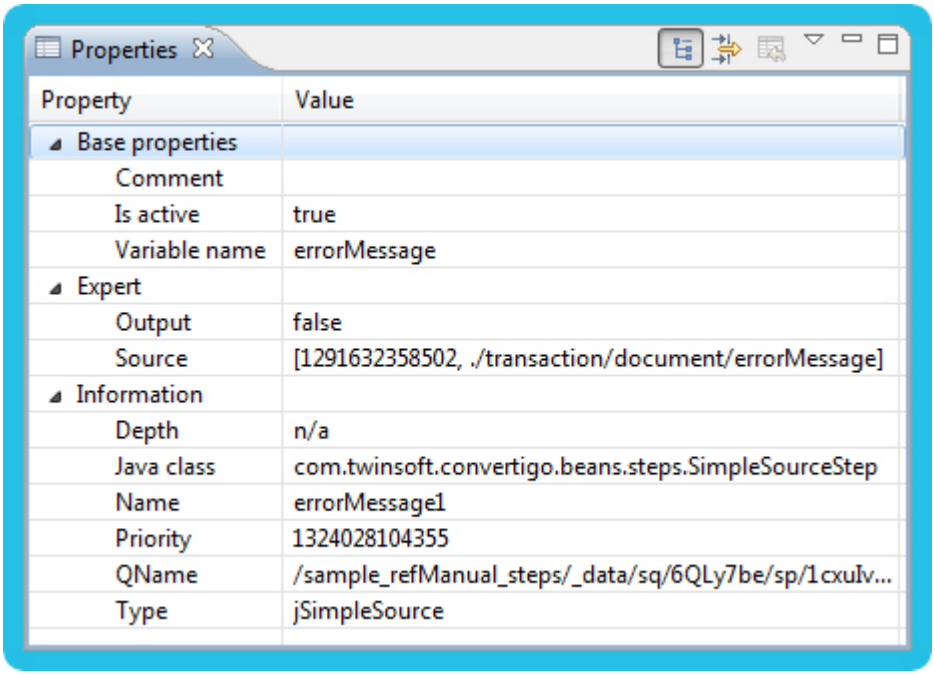


Figure 2 - 142: jSimpleSource step - Configuration example

The **Source** property points towards the `errorMessage` node from the previous *Call Transaction* step. This node belongs to the `GoogleSearch` transaction XML schema retrieved by the `Call_Transaction_GoogleSearch` step:

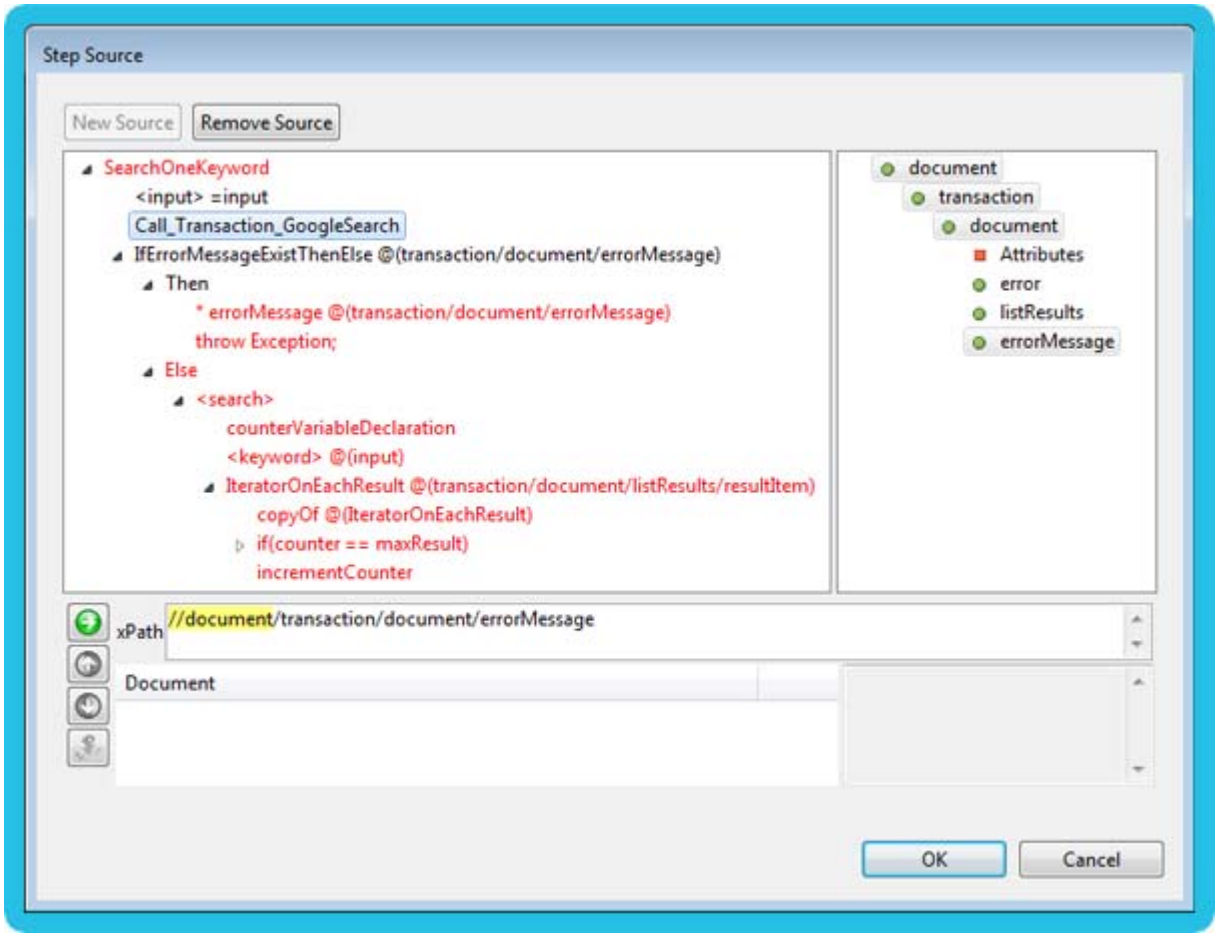


Figure 2 - 143: jSimpleSource step - Source configuration

The step is created in the **Steps** folder of the sequence, next to various other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

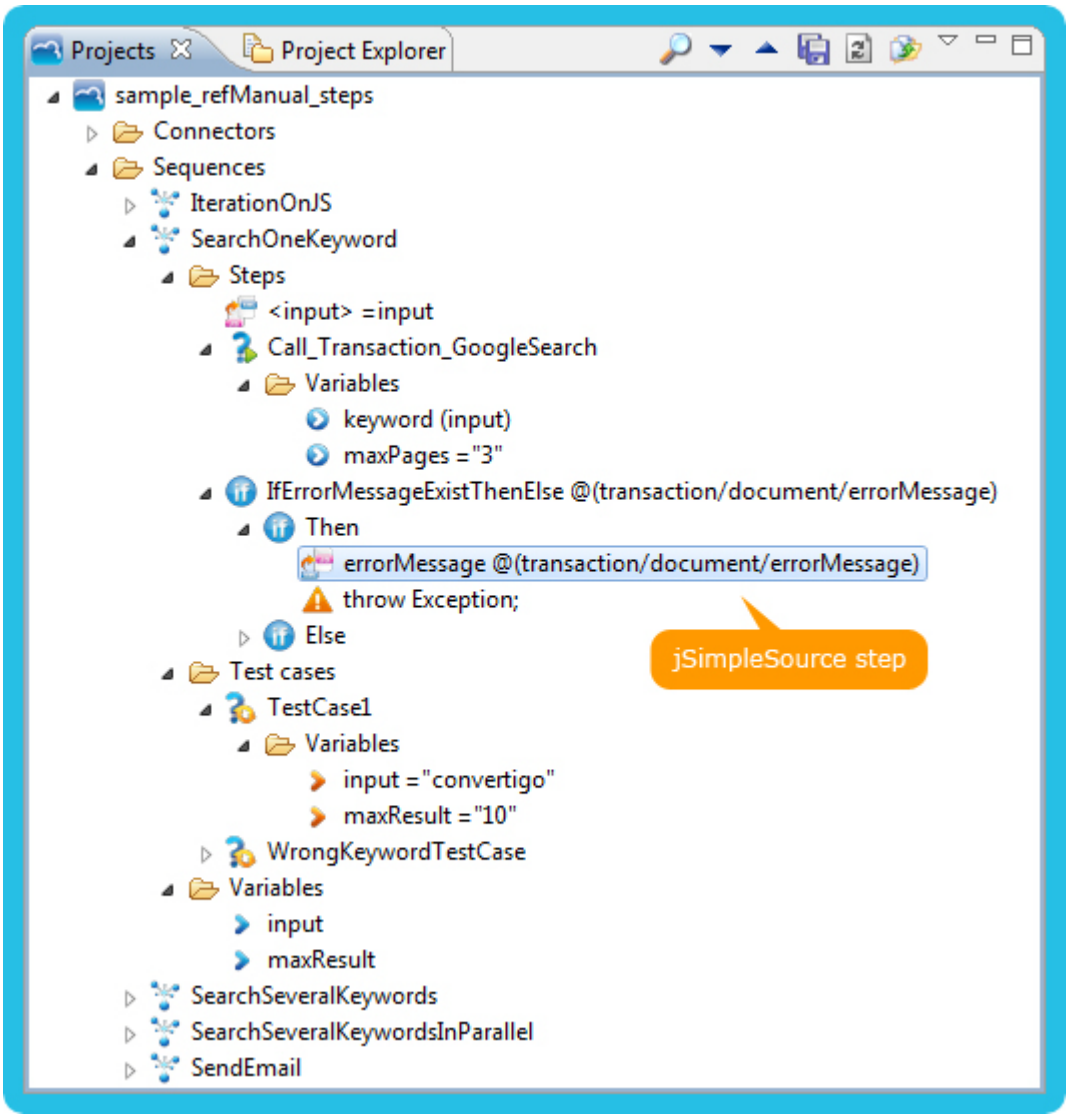


Figure 2 - 144: jSimpleSource step - Object in Projects view, with sequence and other steps

The created variable `errorMessage` is then used in the `jException` step to fill the exception's details. For more information about this step, see "*jException* step" documentation and examples.

jException 

Object Description

Raises a Convertigo Engine exception.

In some circumstances, it is necessary to explicitly raise a Convertigo Engine exception. This is reflected as a `SoapFaultException` for SOAP web service callers or by an error structure in XML output for any other caller.

Message and **Details** properties can be set to complex JavaScript expressions, mixing text strings and data from variables. These expressions are evaluated during the sequence execution and build a dynamic message and details output in the raised exception.

The error XML structure contains a `type` attribute, which value is automatically set to `c8o` in case of Exception. It allows to differentiate an irrecoverable Server error from a project/ applicative error created using an *Error* step (`type` attribute value is then `project`).

A *jException* step breaks the sequence execution flow, the sequence ends just after this step's execution (contrary to *Error* step which does not break the execution flow).

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Details	JS expression	standard	Provides additional details about the triggered error. This property allows the developer to dynamically add some details content in the raised Exception, depending on the sequence execution.
Is active	boolean	standard	Defines whether the step is active.
Message	JS expression	standard	Provides the (humanly readable) error message. This property allows the developer to dynamically define the message text of the raised Exception, depending on the sequence execution.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Examples

Let's consider a `SearchOneKeyword` sequence, which defines two variables named `input`

and `maxResult`. This sequence calls the `searchGoogleWithLimit` transaction, set in the context of the *"Starting With Convertigo Web Integrator"* Quick Guide, with its `input` variable as `keyword` and a hard-coded `maxPages` variable value. The sequence then tests whether the transaction has returned an error message or not. In the first case, it ends by raising an Exception, in the second case, it iterates on the results list and copies to sequence output the `maxResult` first items.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Sequencer objects examples*** *in the* ***New Project*** *wizard.*
Associated project can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

In order to raise the Exception when an error message is found in the transaction output XML, a *jException* step is created with the following parameters:

```
jException [
  message: "The Google Search transaction ended with an error."
  details: errorMessage
  output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

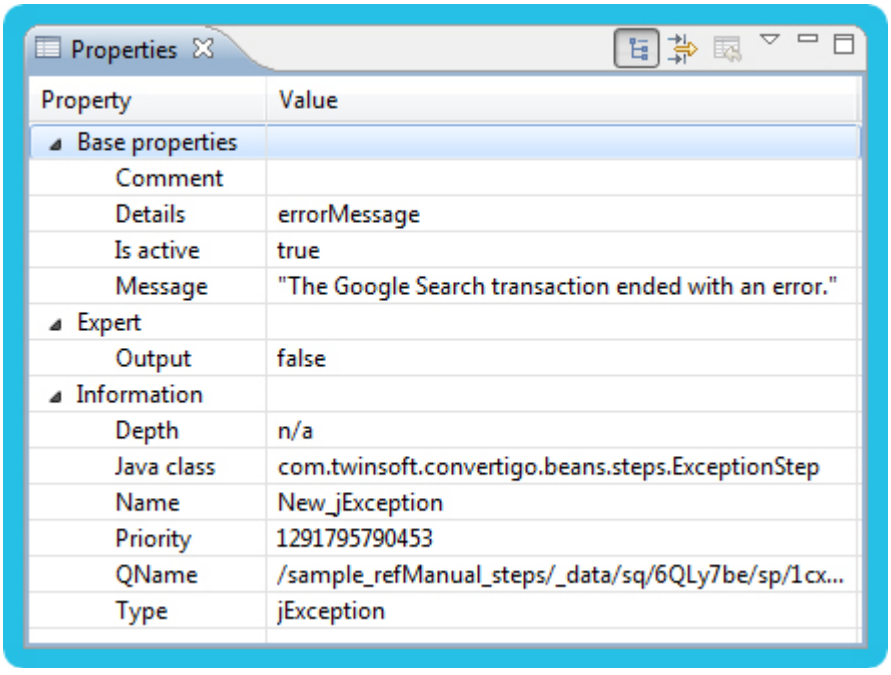


Figure 2 - 145: jException step - Configuration example

The **Details** property is set to a JavaScript expression using a variable named `errorMessage`. This variable is previously set in the sequence thanks to a *jSimpleSource* step (for more information about this step, see "*jSimpleSource* step" documentation and examples). To sum up, the evaluated details message uses the error message retrieved from

the `searchGoogleWithLimit` transaction.

The step is created in the **Steps** folder of the sequence, next to various other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

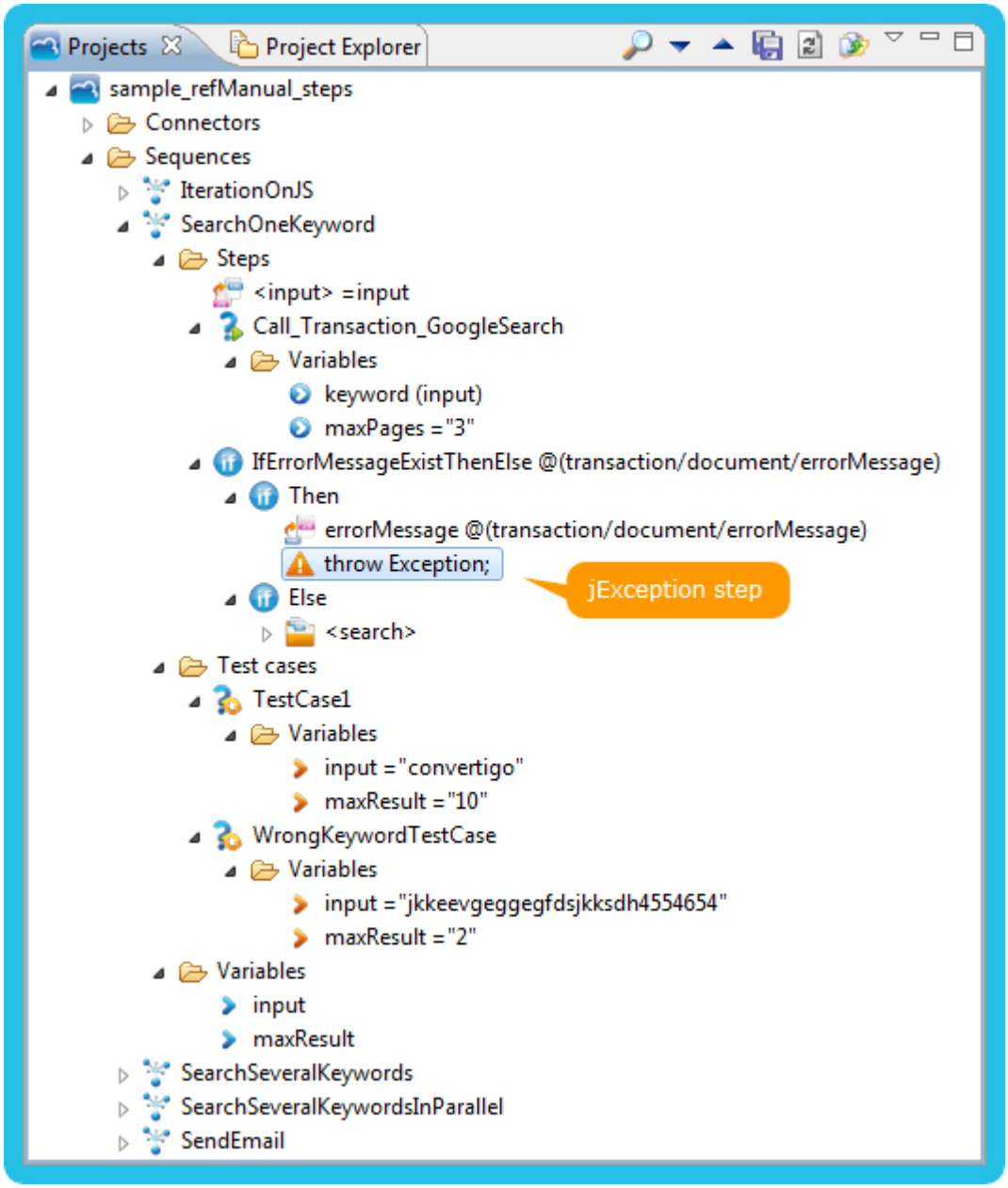


Figure 2 - 146: jException step - Object in Projects view, with sequence and other steps

Run the `WrongKeywordTestCase` test case defined for the sequence, with `input` variable to a value giving no result (you may have to change this value for a no result execution) and `maxResult` to "2". The sequence calls the `searchGoogleWithLimit` transaction with `input` variable passed as `keyword` and `maxPages` to fixed value "3". Then, the sequence detects the error message in the transaction output XML and raises an Engine Exception with the appropriate message and details. It is visible in the Engine log (here through the Administration Console):

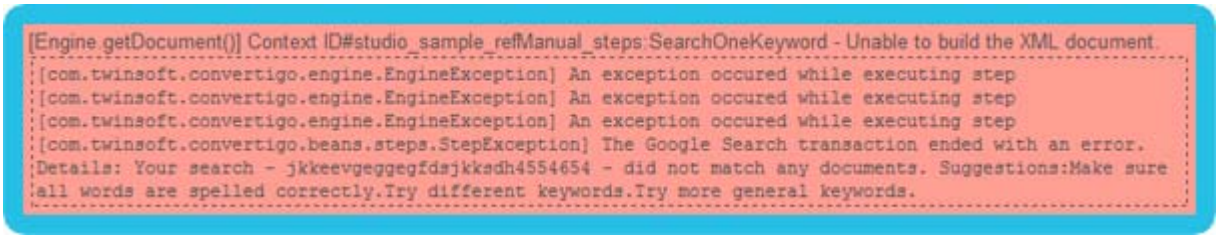


Figure 2 - 147: jException step - Exception message visible in Engine log

XML STEPS

ATTRIBUTE (SEQUENCER) 

OBJECT DESCRIPTION

Creates an XML attribute node.

The *Attribute* step adds an attribute node to parent generated XML element in the sequence XML output.

The XML attribute resulting from this step can be output in the response XML of the sequence if the **Output** property is set to `true`, or used as a source by any other following step.

The attribute is named after the value of the **Node name** property, its value is set thanks to a source defined in **Source** property. If no source is defined or if its results is empty, the XML attribute contains the value of the **Default value** property, if a value is defined in this property.

Note:

- An *Attribute* step can only be added under *Element* steps, *jElement* steps and *Complex* steps.
- No step can be added under an *Attribute* step.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Simple type QName	XmlQName	expert	Defines the schema base type to assign as a type to this simple XML element. This property allows to assign a simple XSD type to the simple XML element generated by this step. It can only be used when the step actually generates a simple XML element.
Attribute namespace	String	standard	Defines the namespace to use for this attribute. Leave it blank for no namespace.
Attribute namespace URI	String	standard	Defines the URI associated with the namespace. Leave it blank for no namespace.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default value	String	standard	Defines the default text value of the attribute. This property allows defining a default value to use when no source is defined or when the source result is empty.
Is active	boolean	standard	Defines whether the step is active.

Property	Type	Category	Description
Node name	String	standard	Defines the name of the generated XML attribute. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to use as value. This property allows defining a node or a list of nodes from a previous step used by current step as value. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step uses the value defined in **Default value** property, if a value is defined in this property. Otherwise, the step creates an empty attribute.

EXAMPLES

Let's consider the `GetXMLData` sequence set in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains three *Attribute* steps (`article_code`, `article_code`, `article_status`) which purpose is to add attributes to the `article` complex element generated by the parent *Complex* step.

The *Attribute* steps appear as follows:

- in the **Projects** view of the Convertigo Studio:

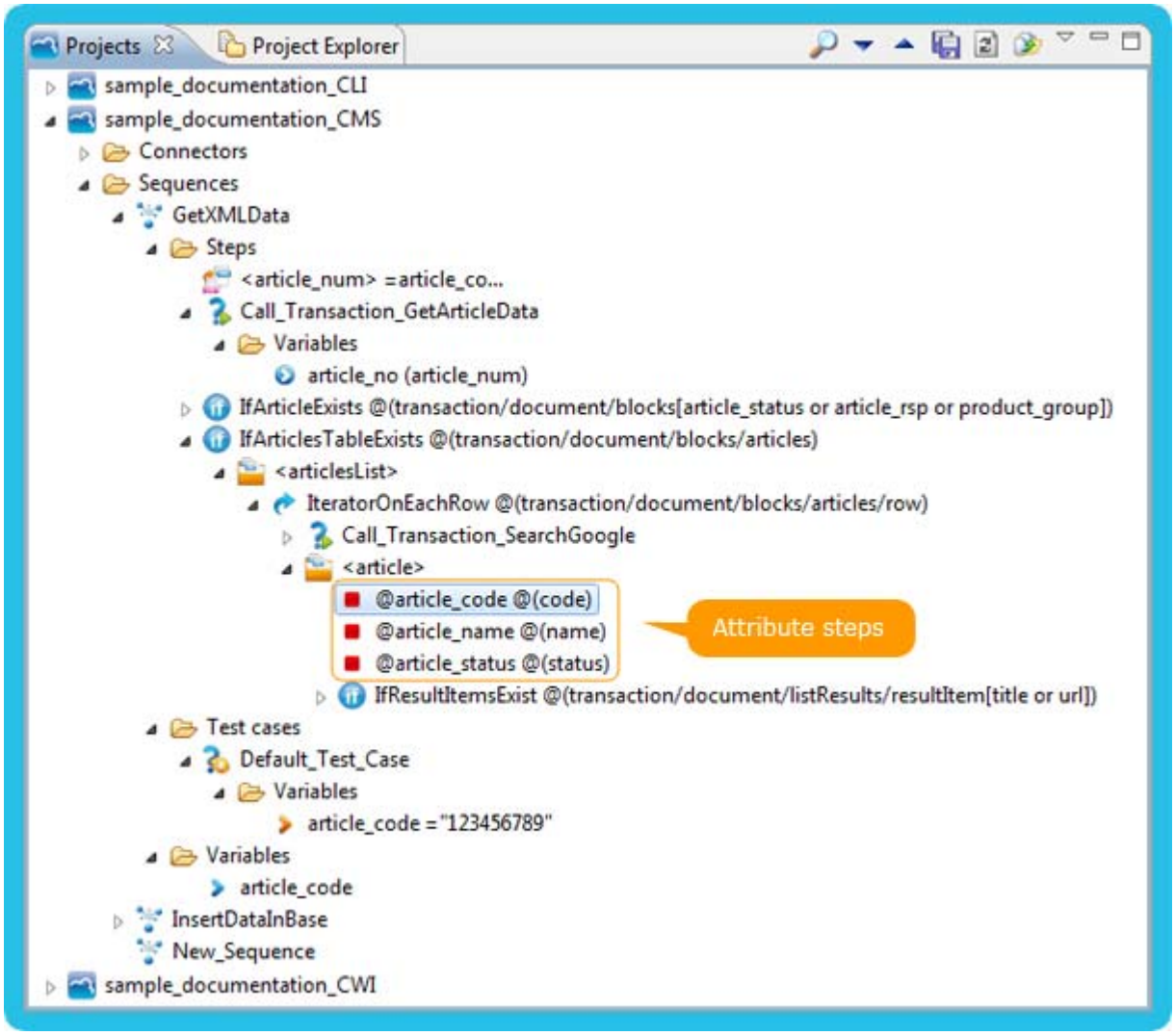


Figure 2 - 148: Attribute step - Article code, name and status in GetXMLData sequence

- in the **Properties** view of the Convertigo Studio (here, `article_code`):

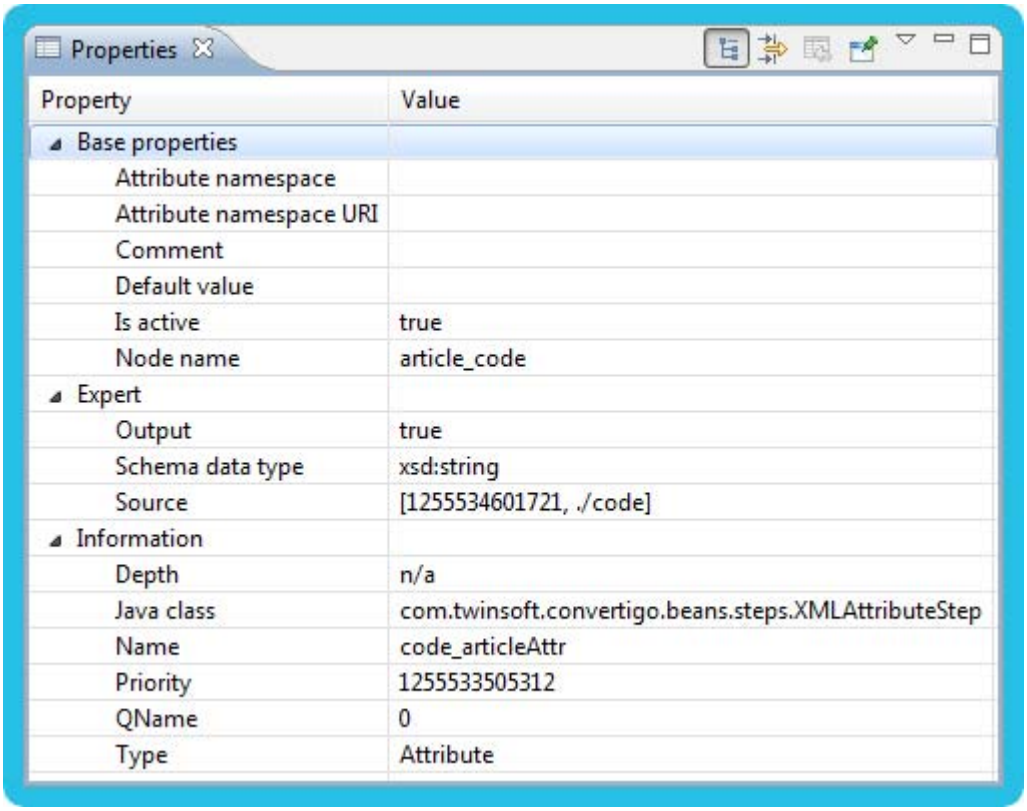


Figure 2 - 149: Attribute step - Configuration example

The **Output** property is set to `true` because we need attributes to be generated in the sequence XML output. The name of the generated attribute is given by the value of the **Node name** property: `article_code`.

When executing the `Default_Test_Case` test case defined for the sequence, the Iterator step iterates on each row element from the `GetArticleData` transaction, calls the `searchGoogleWithLimit` transaction passing as input variable the `name` value of the iterated row and creates the XML output structure including the three defined attributes. These attributes appear as follows in the resulting sequence XML:

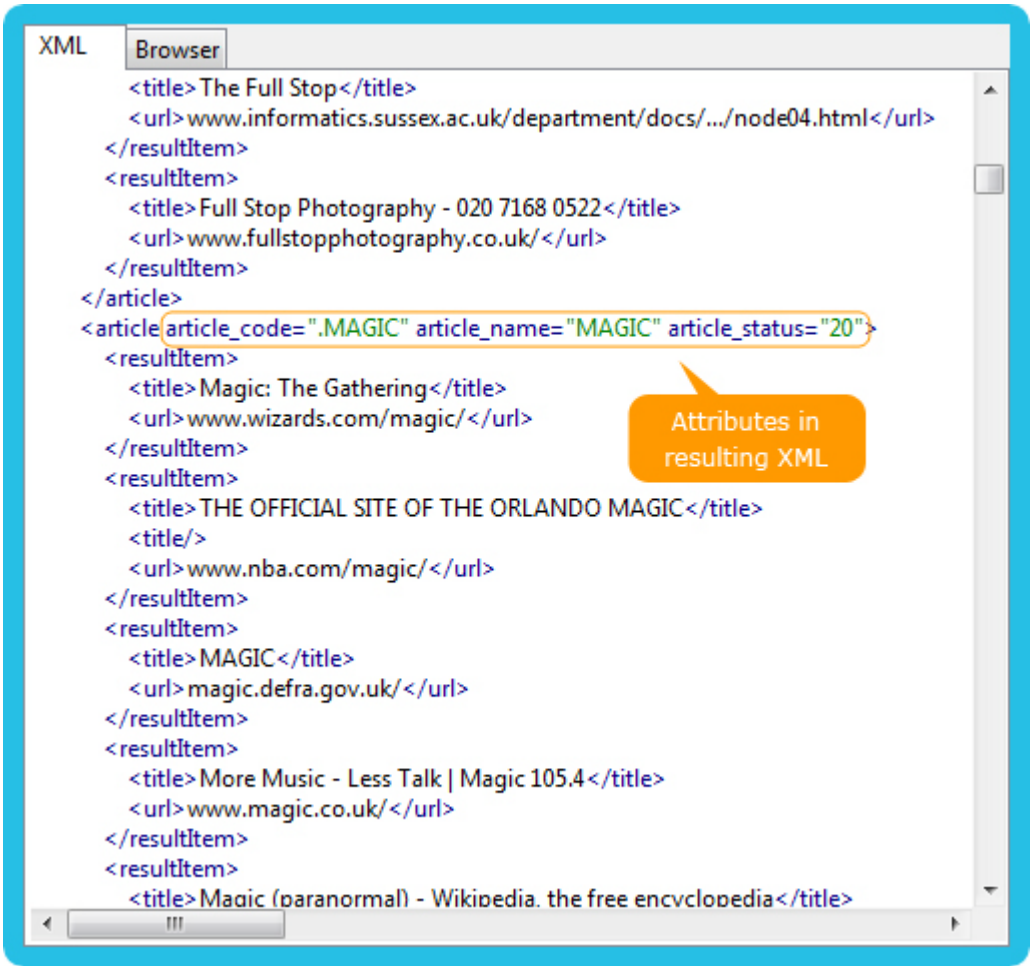


Figure 2 - 150: Attribute step - Attributes generated in GetXmlData sequence XML output

JATTRIBUTE 

OBJECT DESCRIPTION

Creates an XML attribute node based on a JavaScript expression.

The *jAttribute* step adds an attribute node to parent generated XML element in the sequence XML output.

The XML attribute resulting from this step can be output in the response XML of the sequence if the **Output** property is set to `true`, or used as a source by any other following step.

The attribute is named after the value of the **Node name** property, its value is set thanks to a JavaScript expression defined in **Expression** property. If the JavaScript expression is null, the XML attribute contains the value of the **Default value** property.

Notes:

- A *jAttribute* step can only be added under *Element* steps, *jElement* steps and *Complex* steps.
- No step can be added under a *jAttribute* step.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Simple type QName	XmlQName	expert	Defines the schema base type to assign as a type to this simple XML element. This property allows to assign a simple XSD type to the simple XML element generated by this step. It can only be used when the step actually generates a simple XML element.
Attribute namespace	String	standard	Defines the namespace to use for this attribute. Leave it blank for no namespace.
Attribute namespace URI	String	standard	Defines the URI associated with the namespace. Leave it blank for no namespace.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default value	String	standard	Defines the default text value of the node. This property allows defining a default value to use when no content is specified thanks to the **Expression** property of if this expression returns `null`.
Expression	JS expression	standard	Defines the expression evaluated to give the output text. This property is a JavaScript expression that is evaluated during the sequence execution and gives the text string to output in the generated attribute.
Is active	boolean	standard	Defines whether the step is active.

Property	Type	Category	Description
Node name	String	standard	Defines the name of the generated XML attribute. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

COPY 

OBJECT DESCRIPTION

Imports a copy of XML elements sourced from a previous step.

The *Copy* step duplicates and imports a list of nodes from a previously executed step to the sequence XML output. The XML elements resulting from this step can be used as a source by another step.

The list of nodes to duplicate is set thanks to a source defined in **Source** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to copy. This property allows defining a list of nodes from a previous step that are copied by this step. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: nothing is copied in the sequence output.

SORT 

OBJECT DESCRIPTION

Sorts XML nodes from a source using a sort key defined by an XPath.

The *Sort* step works as follows:

- It defines an input list to work on using a source, i.e. a list of nodes to be sorted from a previous step.
- It applies a common XPath on each item of the list to define a sort key for this node. The XPath is defined in the **Sort key XPath** property. The result of this XPath applied on each item of the list is the sort key. This sort key is the value that can actually be sorted (by number, by alphabetical order, etc.) and used to sort the matching nodes.
- It uses the sort keys to sort the nodes of the list, using options defined in other properties (**Sort order**, **Sort type** and **Sort options**).
- It finally outputs the sorted nodes, so they can be used as source by a following step or output by the *Sequence*.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Sort key XPath	String	expert	The XPath that is applied on each node of the list to define its sort key. This property allows to define the XPath that will be applied on each node of the source list to give the sort key of the node. The sort key of each node of the list is then used for sorting the list: each node is represented by its sort key during the sort algorithm.

Property	Type	Category	Description
Sort option	JS expression	expert	Defines some options to sort the sort key, depending on their type. Depending on the **Sort type** property value, this property contains options that are needed to make the comparison. For `Date` sort type (sort keys of date type), this property must contain the sort keys date format. For more information on usable symbols, see Appendix "*Date format - Usable symbols*".
Sort order	Order	expert	Defines the sorting order. This property allows to define the sorting order. It can take the following values: • `Ascending`: the sort is performed by ascending order, • `Descending`: the sort is performed by descending order.
Sort type	TypeOrder	expert	Defines on which data type the sort is performed. This property allows to define on which data type the sort is performed. It can take the following values: • `String`: the sort keys are of string type, the sort is performed by alphabetical order, • `Number`: the sort keys are of number type, the sort is performed by numerical order, • `Date`: the sort keys are of date type, the sort is performed chronologically, using the **Sort option** property to define the date format.
Source	XMLVector	expert	Defines the list of nodes to sort using a source. This property allows defining a list of nodes from a previous step on which current step works, i.e. the items to sort. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: the list of objects to sort is empty, nothing is sorted and the parent sequence execution continues.

COMPLEX 

OBJECT DESCRIPTION

Defines an empty XML element (with no text content).

The *Complex* step generates an output XML tag and can contain other steps generating XML (for example: *Element*, *Attribute* or *Complex* steps) in order to create any XML structure.

This structure can be output in the response XML of the sequence if the **Output** property is set to `true`, or used as a source by any other following step.

Note: Child steps have to be added under this step to create a data structure.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Complex type QName	XmlQName	expert	Defines a global schema Complex type to assign as a type to this structured XML element. This property allows to assign an XSD Complex type to the structured XML element generated by this step. The QName defined by this property can be: • a new Complex type name: the Complex type will be created from this structure in the project's schema, • an already defined Complex type name: the existing Complex type is used and possibly enhanced by the structure generated by this XML element (if not identical): the Complex type will be a union of all XML structures using it. To use an already existing Complex type name, the popup editor displays all types available in the project's schema: • in grey: all non editable schema types (Convertigo standard error schema, types imported through references, types of transactions, etc.), • in blue: all schema types dynamically generated from the project's sequences, • in green: all dynamic schema types explicitly named by **Assigned XSD Complex type QName** or **Assigned XSD Element ref QName** properties.

Property	Type	Category	Description
Assigned XSD Element ref QName	XmlQName	expert	Defines a global schema Element to assign as a reference to this structured XML element. This property allows to assign a referenced Element to the structured XML element generated by this step. The referenced Element and its corresponding XSD Complex type will also be created if non existing in the project's schema. The QName defined by this property can be: • a new referenced Element name: the referenced Element and the corresponding Complex type will be created from this structure in the project's schema, • an already defined referenced Element name: the existing referenced Element is used and its Complex type is possibly enhanced by the structure generated by this XML element (if not identical): the Complex type will be a union of all XML structures using it. To use an already existing referenced Element name, the popup editor displays all types available in the project's schema: • in grey: all non editable schema types (Convertigo standard error schema, types imported through references, types of transactions, etc.), • in blue: all schema types dynamically generated from the project's sequences, • in green: all dynamic schema types explicitly named by **Assigned XSD Complex type QName** or **Assigned XSD Element ref QName** properties.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

EXAMPLES

Let's consider the `GetXMLData` sequence set in the context of the "*Starting With Convertigo*

Mashup Sequencer" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains three *Complex* steps called `articleFound`, `articlesList` and `article` which purpose is to produce XML complex elements used to structure as required the XML output. As a result, the whole `GetXMLData` sequence XML output is made up of complex elements.

The three above-mentioned complex elements appear as follows in the **Projects** view of the Convertigo Studio:

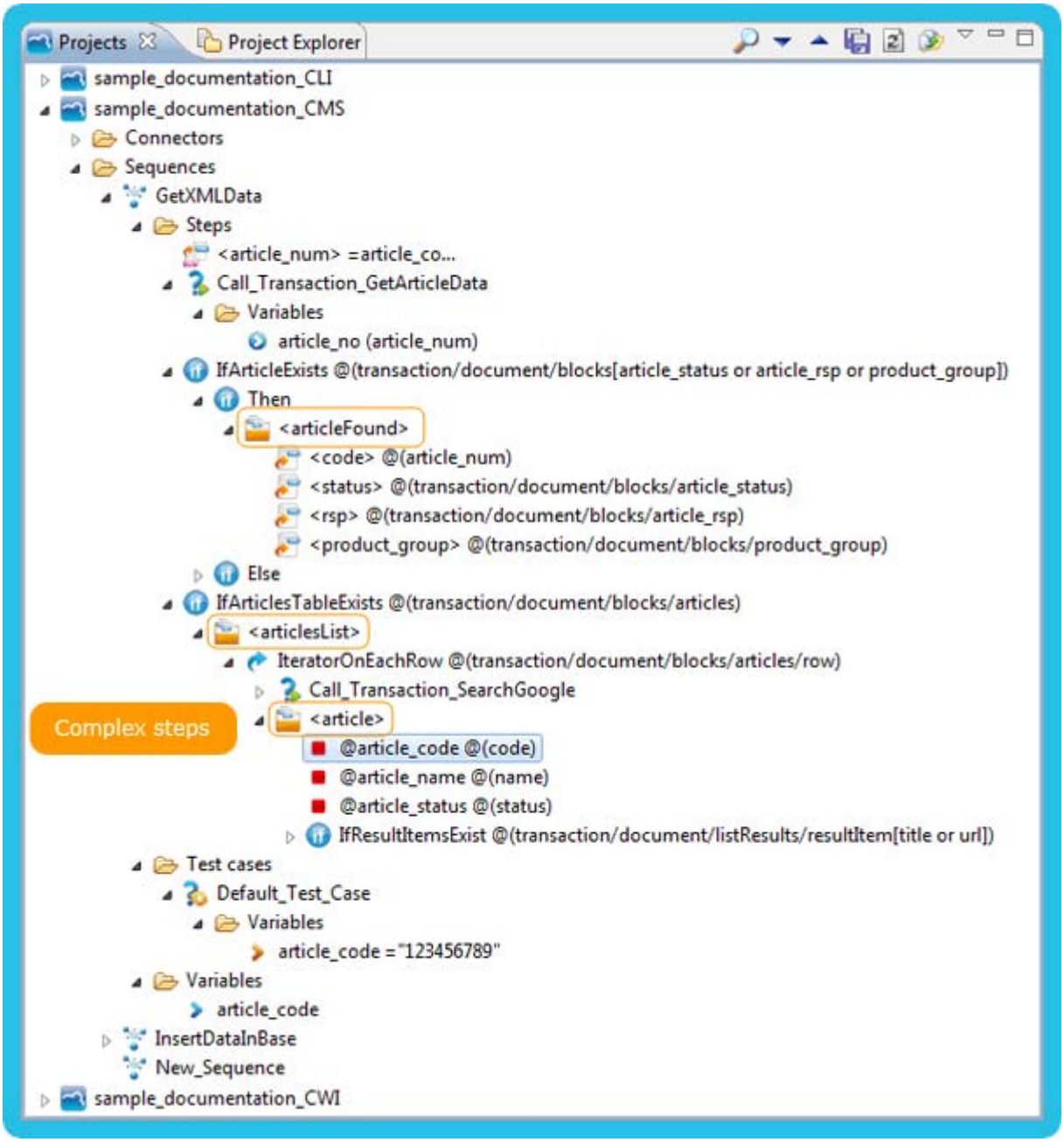


Figure 2 - 151: Complex step - GetXMLData sequence complex elements

Once executed, the sequence produces an XML output made up of complex elements:

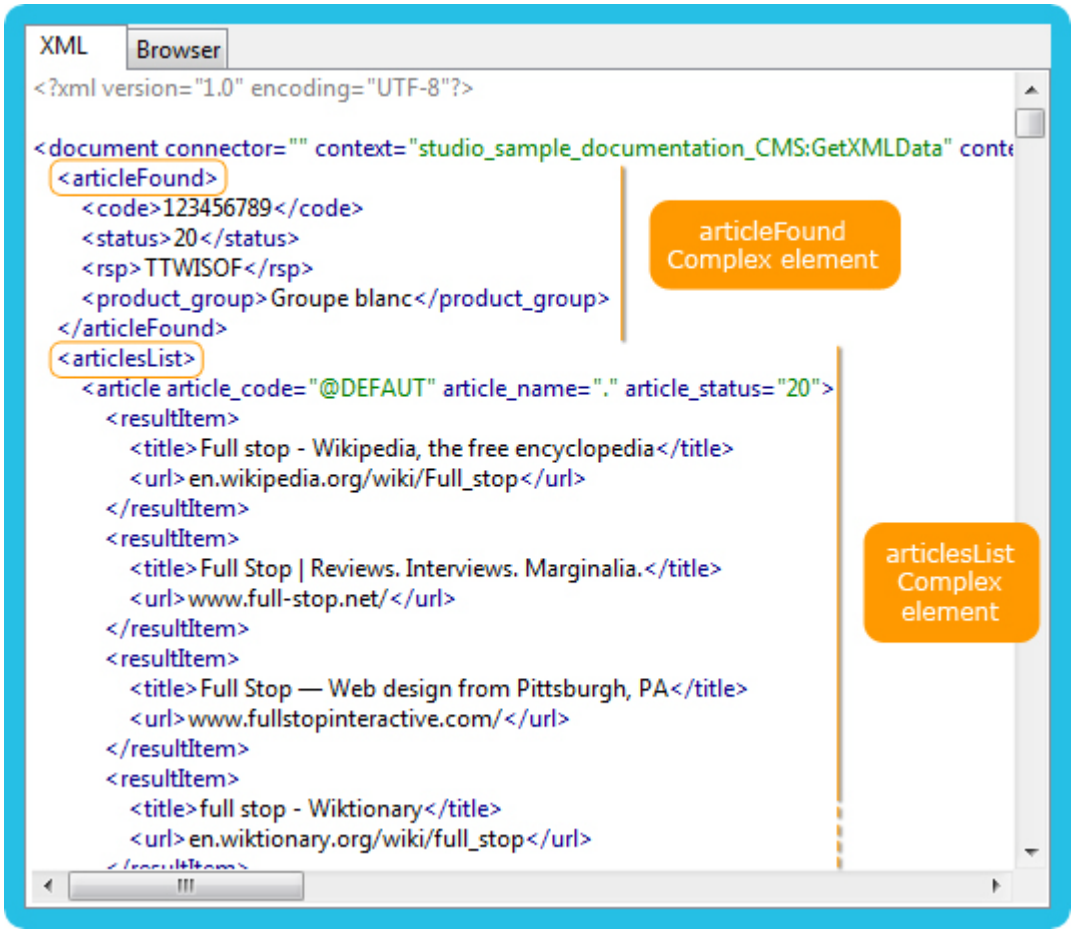


Figure 2 - 152: Complex step - GetXMLData sequence XML output

The `articleFound` *Complex* step appears as follows in the **Properties** view:

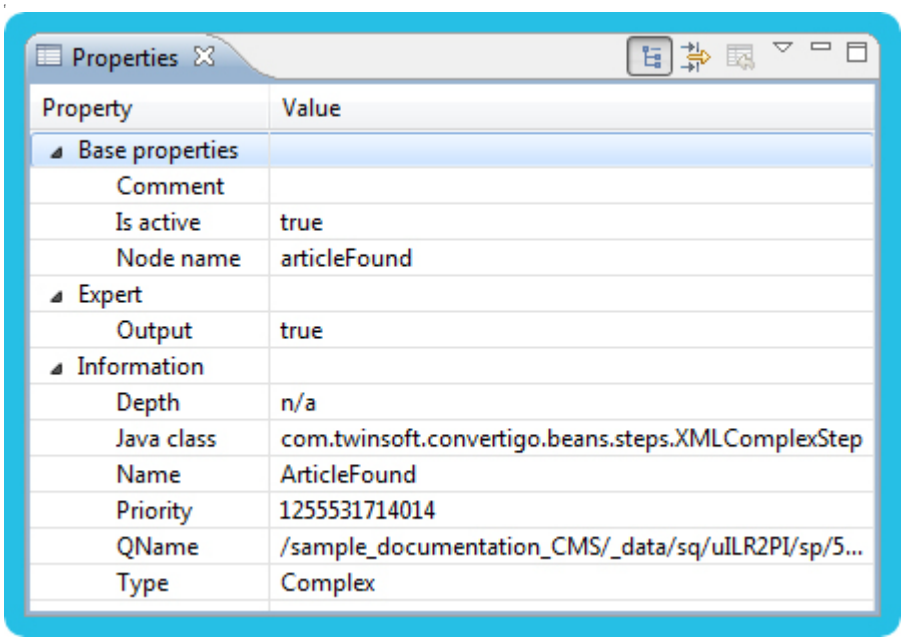


Figure 2 - 153: Complex step - Configuration example

The **Output** property is set to `true`, indicating that the step generates XML content in the sequence output XML. The generated tag name (`articleFound`, see Figure 2 - 152) is defined through the **Node name** property.

ERROR STRUCTURE 

OBJECT DESCRIPTION

Creates an XML structure describing an applicative error.

The *Error structure* step generates an output XML structure corresponding to an applicative error. This structure is created on a standard basis (error code, message, details) that is automatically managed by client applications developed with Convertigo Mobilizer and/or using the Convertigo Templating Framework.

The basic error structure elements are filled using the step's corresponding properties: **Code**, **Message** and **Details**. The structure can be enhanced with user-defined elements: to do so, simply add child steps under this *Error structure* step (the same way as for a *Complex* step).

This error structure contains a `type` attribute, which value is automatically set to `project`. It allows to differentiate a project/applicative error from an irrecoverable Server error (`type` attribute value is then `c8o`).

The error structure can be output in the response XML of the sequence if the **Output** property is set to `true`, or used as a source by any other following step.

An *Error structure* step does not break the sequence execution flow (contrary to *jException* step for example). Use the *Break* or *Return* steps when you need to stop the sequence execution flow after an *Error structure* step.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Code	SmartType	standard	A numeric error code to fill in error structure, identifying the error. This property is a "smart type" property, that allows to specify the error code. A "smart type" property can be of one of the following types: • a **text**: the value is therefore a hard-coded text value, • a **JavaScript expression**: the value is therefore a JavaScript expression that is evaluated at sequence execution, • a **source**: the value is a source and can be picked using the source picker. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. Notes: • If you use the **source** type for this property, the XPath application on target XML should give a text result. Otherwise, the first node's text content is taken. • If no error message text is defined by the **Message** property, the client project error dictionary can be used, if using the Internationalization framework, to retrieve the error message corresponding to this error code.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Details	SmartType	standard	Some technical information details about the error, to fill in error structure, mainly for debugging purposes. This property is a "smart type" property, that allows to specify the error details. A "smart type" property can be of one of the following types: • a **text**: the value is therefore a hard-coded text value, • a **JavaScript expression**: the value is therefore a JavaScript expression that is evaluated at sequence execution, • a **source**: the value is a source and can be picked using the source picker. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. Note: If you use the **source** type for this property, the XPath application on target XML should give a text result. Otherwise, the first node's text content is taken.
Is active	boolean	standard	Defines whether the step is active.

Property	Type	Category	Description
Message	SmartType	standard	An optional text message to fill in error structure, explaining the error. This property is a "smart type" property, that allows to specify the error message. A "smart type" property can be of one of the following types: • a **text**: the value is therefore a hard-coded text value, • a **JavaScript expression**: the value is therefore a JavaScript expression that is evaluated at sequence execution, • a **source**: the value is a source and can be picked using the source picker. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. Notes: • If you use the **source** type for this property, the XPath application on target XML should give a text result. Otherwise, the first node's text content is taken. • If this error message text is not present in output, the client project error dictionary can be used, if using the Internationalization framework, to retrieve the error message corresponding to the error code defined by the **Code** property.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

ELEMENT 

OBJECT DESCRIPTION

Creates an XML element with a text content.

The *Element* step adds an element node with text content to parent generated XML element in the sequence XML output.

The XML element resulting from this step can be output in the response XML of the sequence if the **Output** property is set to `true`, or used as a source by any other following step.

The element is named after the value of the **Node name** property, its value is set thanks to a source defined in **Source** property. If no source is defined or if its results is empty, the XML element contains the value of the **Default value** property, if a value is defined in this property.

Notes:

- Child steps can be added under this step to create a data structure.
- In Convertigo Studio, when an *Element* step is created in a sequence, it can be easily replaced by a *Concat* step, using the right-click menu on the step and choosing the option **Change to** > **Concat**. The **Node name** property remains the same. The **Source** and **Default value** properties are moved to the *Concat* step as two lines of the list of source items to concat, one with a source defined and one with a default value defined.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Complex type QName	XmlQName	expert	Defines a global schema Complex type to assign as a type to this structured XML element. This property allows to assign an XSD Complex type to the structured XML element generated by this step. The QName defined by this property can be: • a new Complex type name: the Complex type will be created from this structure in the project's schema, • an already defined Complex type name: the existing Complex type is used and possibly enhanced by the structure generated by this XML element (if not identical): the Complex type will be a union of all XML structures using it. To use an already existing Complex type name, the popup editor displays all types available in the project's schema: • in grey: all non editable schema types (Convertigo standard error schema, types imported through references, types of transactions, etc.), • in blue: all schema types dynamically generated from the project's sequences, • in green: all dynamic schema types explicitly named by **Assigned XSD Complex type QName** or **Assigned XSD Element ref QName** properties.
Assigned XSD Element ref QName	XmlQName	expert	Defines a global schema Element to assign as a reference to this structured XML element. This property allows to assign a referenced Element to the structured XML element generated by this step. The referenced Element and its corresponding XSD Complex type will also be created if non existing in the project's schema. The QName defined by this property can be: • a new referenced Element name: the referenced Element and the corresponding Complex type will be created from this structure in the project's schema, • an already defined referenced Element name: the existing referenced Element is used and its Complex type is possibly enhanced by the structure generated by this XML element (if not identical): the Complex type will be a union of all XML structures using it. To use an already existing referenced Element name, the popup editor displays all types available in the project's schema: • in grey: all non editable schema types (Convertigo standard error schema, types imported through references, types of transactions, etc.), • in blue: all schema types dynamically generated from the project's sequences, • in green: all dynamic schema types explicitly named by **Assigned XSD Complex type QName** or **Assigned XSD Element ref QName** properties.

Property	Type	Category	Description
Assigned XSD Simple type QName	XmlQName	expert	Defines the schema base type to assign as a type to this simple XML element. This property allows to assign a simple XSD type to the simple XML element generated by this step. It can only be used when the step actually generates a simple XML element.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default value	String	standard	Defines the default text value of the element. This property allows defining a default value to use when no source is defined or when the source result is empty.
Is active	boolean	standard	Defines whether the step is active.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	XMLVector	expert	Defines the source to use as value. This property allows defining a node or a list of nodes from a previous step on which current step works. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step uses the value defined in **Default value** property, if a value is defined in this property. Otherwise, the step creates an element with no data.

jElement 

Object Description

Defines an XML element based on a JavaScript expression.

The *jElement* step adds an element node with text content to parent generated XML element in the sequence XML output.

The XML element resulting from this step can be output in the response XML of the sequence if the **Output** property is set to `true`, or used as a source by any other following step.

The element is named after the value of the **Node name** property, its value is set thanks to a JavaScript expression defined in **Expression** property. If the JavaScript expression is null, the XML element contains the value of the **Default value** property.

Note: Child steps can be added under this step to create a data structure.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Complex type QName	XmlQName	expert	Defines a global schema Complex type to assign as a type to this structured XML element. This property allows to assign an XSD Complex type to the structured XML element generated by this step. The QName defined by this property can be: • a new Complex type name: the Complex type will be created from this structure in the project's schema, • an already defined Complex type name: the existing Complex type is used and possibly enhanced by the structure generated by this XML element (if not identical): the Complex type will be a union of all XML structures using it. To use an already existing Complex type name, the popup editor displays all types available in the project's schema: • in grey: all non editable schema types (Convertigo standard error schema, types imported through references, types of transactions, etc.), • in blue: all schema types dynamically generated from the project's sequences, • in green: all dynamic schema types explicitly named by **Assigned XSD Complex type QName** or **Assigned XSD Element ref QName** properties.

Property	Type	Category	Description
Assigned XSD Element ref QName	XmlQName	expert	Defines a global schema Element to assign as a reference to this structured XML element. This property allows to assign a referenced Element to the structured XML element generated by this step. The referenced Element and its corresponding XSD Complex type will also be created if non existing in the project's schema. The QName defined by this property can be: • a new referenced Element name: the referenced Element and the corresponding Complex type will be created from this structure in the project's schema, • an already defined referenced Element name: the existing referenced Element is used and its Complex type is possibly enhanced by the structure generated by this XML element (if not identical): the Complex type will be a union of all XML structures using it. To use an already existing referenced Element name, the popup editor displays all types available in the project's schema: • in grey: all non editable schema types (Convertigo standard error schema, types imported through references, types of transactions, etc.), • in blue: all schema types dynamically generated from the project's sequences, • in green: all dynamic schema types explicitly named by **Assigned XSD Complex type QName** or **Assigned XSD Element ref QName** properties.
Assigned XSD Simple type QName	XmlQName	expert	Defines the schema base type to assign as a type to this simple XML element. This property allows to assign a simple XSD type to the simple XML element generated by this step. It can only be used when the step actually generates a simple XML element.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default value	String	standard	Defines the default text value of the node. This property allows defining a default value to use when no content is specified thanks to the **Expression** property of if this expression returns `null`.
Expression	JS expression	standard	Defines the expression evaluated to give the output text. This property is a JavaScript expression that is evaluated during the sequence execution and gives the text string to output in the generated element.
Is active	boolean	standard	Defines whether the step is active.

Property	Type	Category	Description
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

EXAMPLES

Example 1

Let's consider a `SearchOneKeyword` sequence, which defines two variables named `input` and `maxResult.` This sequence calls the `searchGoogleWithLimit` transaction, set in the context of the *"Starting With Convertigo Web Integrator"* tutorial, with its `input` variable as `keyword` and a hard-coded `maxPages` variable value. The sequence then tests whether the transaction has returned an error message or not. In the first case, it ends by raising an Exception, in the second case, it iterates on the results list and copies to sequence output the `maxResult` first items.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Sequencer objects examples*** *in the* ***New Project*** *wizard.*
Associated project can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

In order to be able to call the `searchGoogleWithLimit` transaction with the `input` variable value into transaction `keyword` variable, a *jElement* step is created with the following parameters:

```
jElement [
   expression: input
   default value=<empty>
   node name="input"
   output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

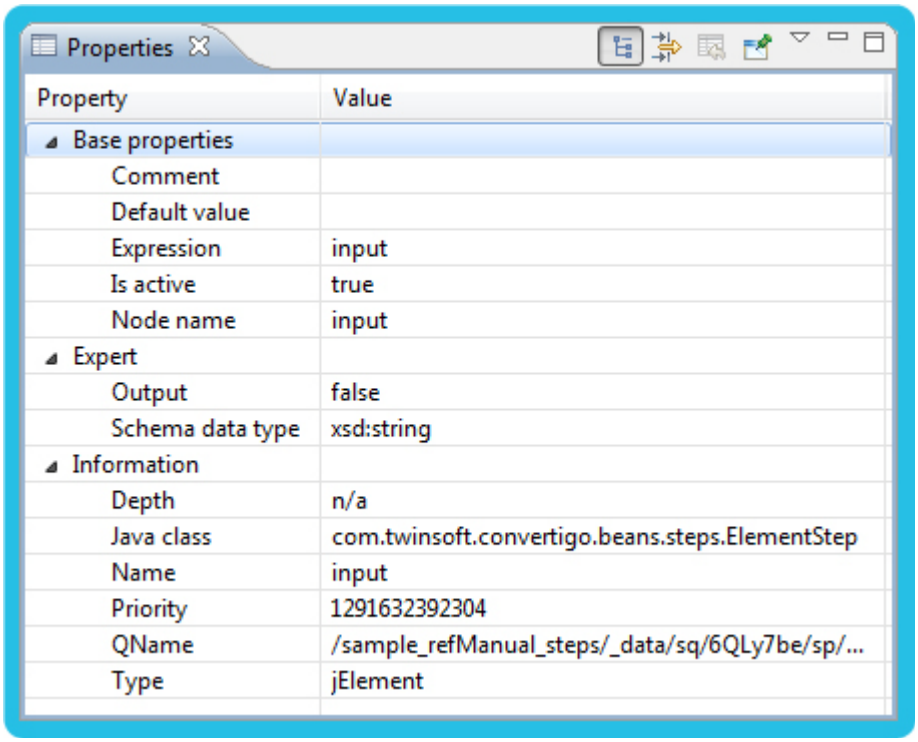


Figure 2 - 154: jElement step - Configuration example

The **Expression** property is set to `input` so the input variable value is set into this element when the sequence is executed. The **Output** property is set to `false` although this is a step generating XML because we don't need this data to appear in the sequence XML output.

The step is created in the **Steps** folder of the sequence, next to various other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

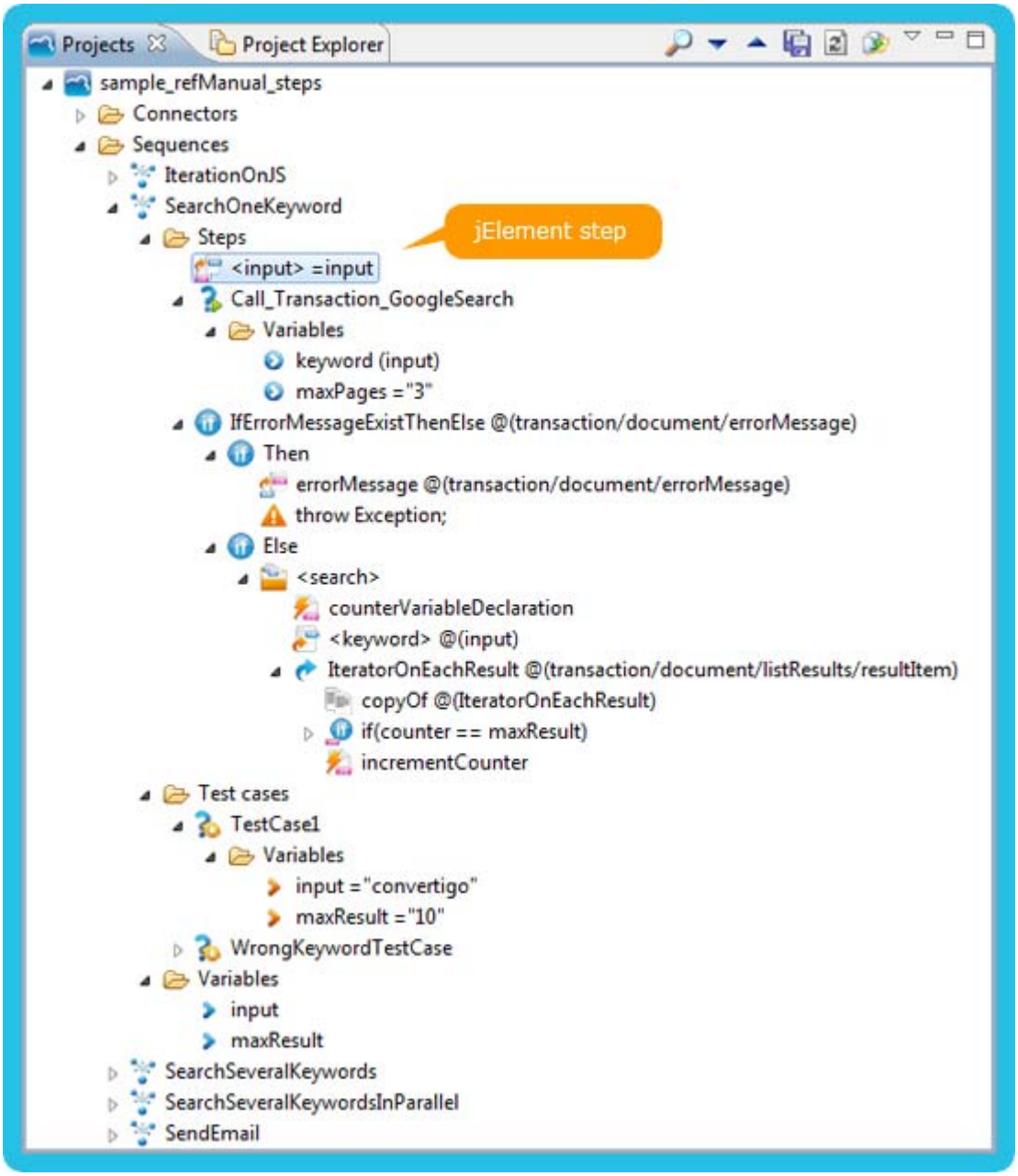


Figure 2 - 155: jElement step - Object in Projects view, with sequence and other steps

When executing the `TestCase1` test case defined for the sequence, with `input` to "convertigo" and `maxResult` to "10", the sequence calls the `searchGoogleWithLimit` transaction with `input` variable passed as `keyword` thanks to a source pointing on the XML element generated by the defined *jElement* step. The `maxPages` variable is passed with the fixed value "3" defined in its default value.

Example 2

Let's consider the `GetXMLData` sequence set in the context of the "*Starting with Convertigo Mashup Sequencer*" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by choosing ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains a *jElement* step called `article_num` which purpose is to generate an `article_num` simple element serving as source for the variable of the following *Call Transaction* step.

The `article_num` step **Expression** property is set to `article_code` to match the sequence variable (see Figure 2 - 157), so that the `GetArticleData` transaction called by the `Call_Transaction_GetArticleData` step can use the sequence variable as input variable:

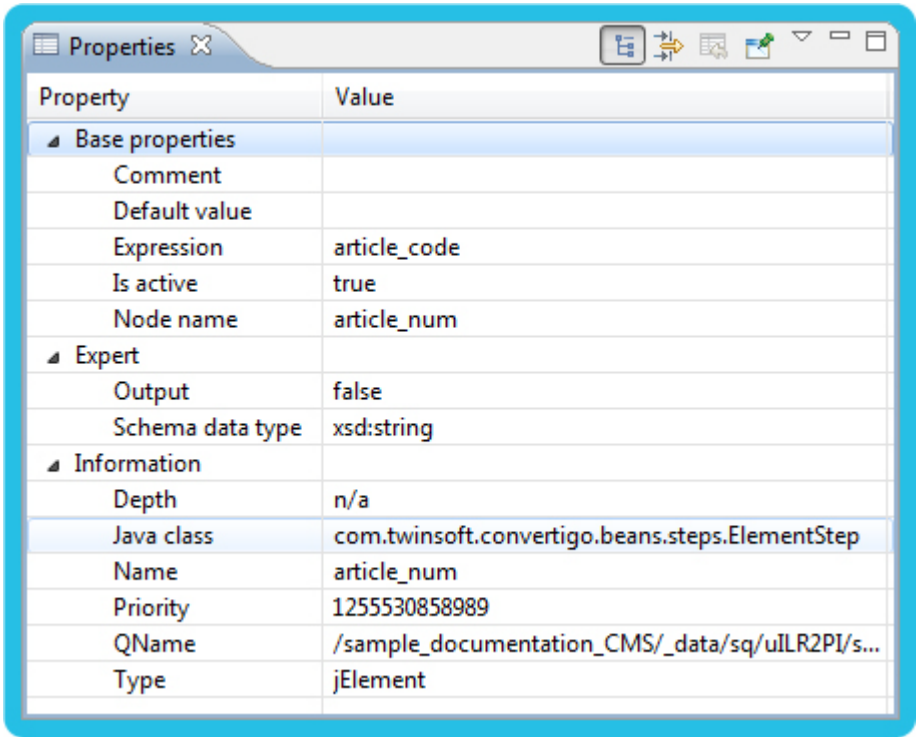


Figure 2 - 156: jElement step - article_num jElement step properties

The **Output** property is set to `false` because we do not need to generate this element in the sequence XML output. We just need its content to be immediately used by the second step (*Call Transaction*) as input variable in the transaction.

The `article_num` *jElement* step appears as follows in the **Projects** view of the Convertigo Studio:

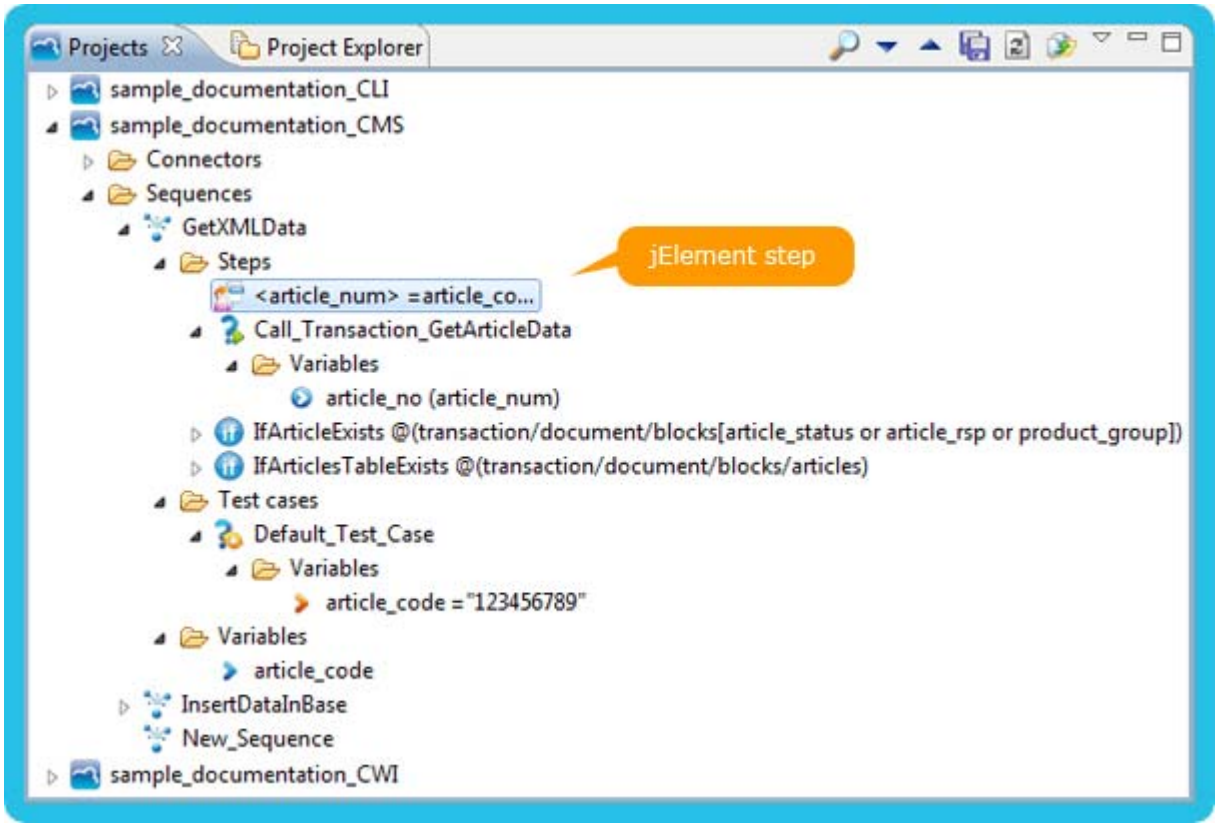


Figure 2 - 157: jElement step - article_num jElement step in GetXmlData sequence

SPLIT

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

TRANSFORM 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

COUNT

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

CONCAT 

OBJECT DESCRIPTION

Concatenates defined sources.

Concat steps are used to concatenate string elements into one new resulting XML element inserted in the output.

The *Concat* step uses an array of input strings (set using the **Sources** property) to be concatenated. An optional **Separator** parameter can also be added. If used, the separator is inserted in the resulting string between each concatenated element.

The resulting string is added to the sequence XML output and can be used as a new source for other steps.

Note: In Convertigo Studio, when a *Concat* step is created in a sequence, it can be easily replaced by an *Element* step, using the right-click menu on the step and choosing the option **Change to** > **Element**.

- The **Node name** property remains the same.
- The first source filled in the *Concat* step is moved to the **Source** property of the *Element* step.
- **Default value** properties defined in *Concat* step lines are concatenated and moved to the **Default value** property of the *Element* step.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Simple type QName	XmlQName	expert	Defines the schema base type to assign as a type to this simple XML element. This property allows to assign a simple XSD type to the simple XML element generated by this step. It can only be used when the step actually generates a simple XML element.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.

Property	Type	Category	Description
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Separator	String	expert	Defines the text to be used as a separator string. If set, this text is added between each text to concatenate. Default value is a white space, think about removing it if you do not want to use it.
Sources	XMLVector	expert	Defines a list of source items to use as values. This property allows defining a list of source items that are used to create the result value. Each source item contains three columns to be set: • **Description**: Defines a comment or description about this source item. • **Source**: Defines the source. A source is a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. • **Default value**: Defines the default value for this source. If the source's XPath doesn't match in the referenced step or if the source is left blank, the default value is used. Otherwise, the source item creates no data. Each source item may define a source and a default value. Note: A new source item can be added to the list using the blue keyboard icon. The source items defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.

EXAMPLES

The `InsertDataInBase` sequence set in the context of the "Starting With Convertigo Mashup Sequencer" *Quick Guide* contains a *Concat* step called `insertError`.

The purpose of the `insertError` step is to output error messages in the `InsertDataInBase` sequence XML when database errors occur. Error messages are made up of concatenated elements, either fixed or sourced.

The `insertError` step appears as follows in the **Projects View** of the Convertigo Studio:

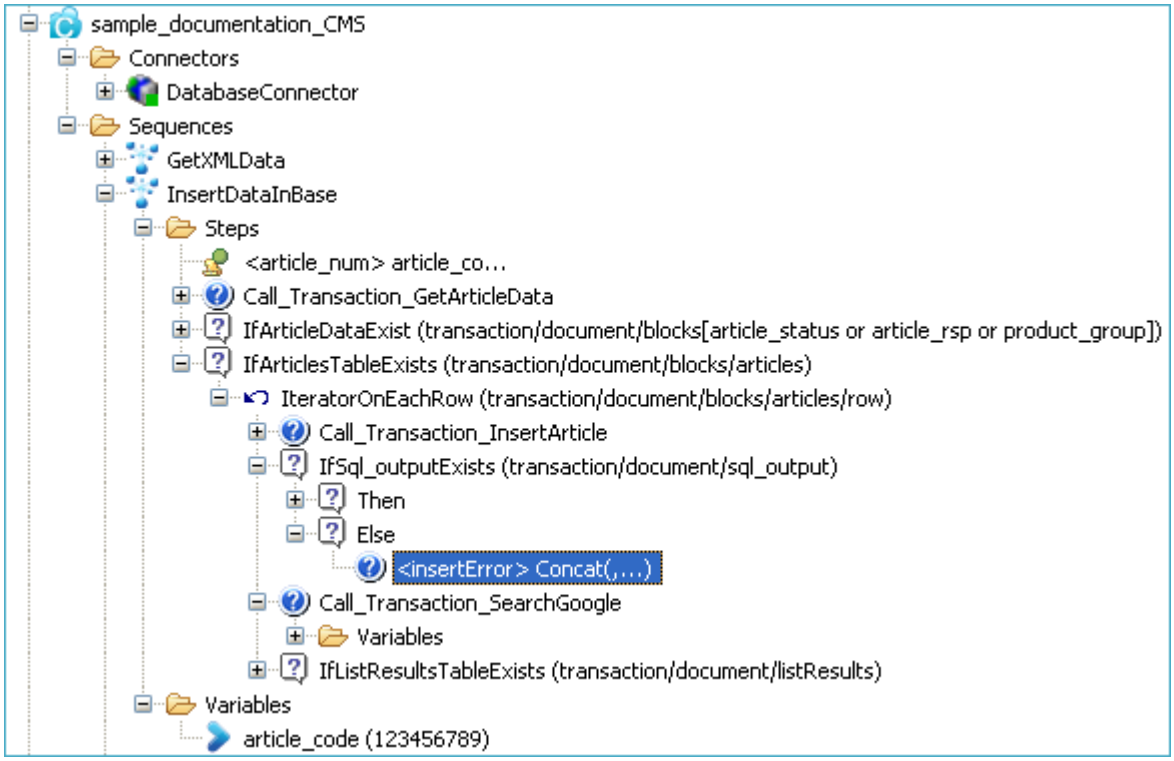


Figure 2 - 158: InsertInDataBase sequence insertError step

When a database error occurs, the `insertError` step generates a line in the XML output indicating that an error occured:

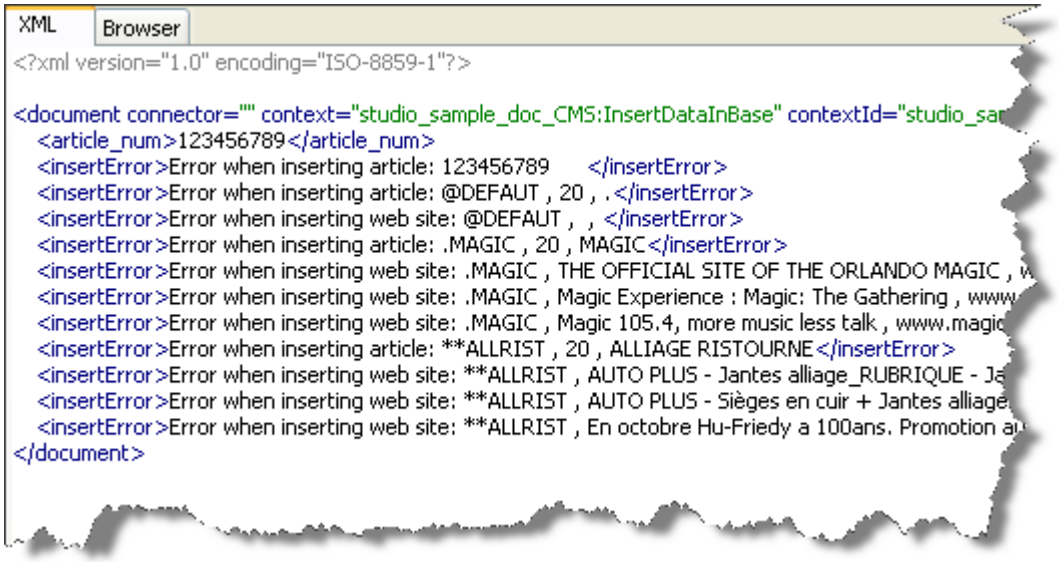


Figure 2 - 159: Error messages produced by the insertError step

As mentioned before, error messages are made up of fixed an sourced elements:

Figure 2 - 160: Fixed and sources elements of concatenated error messages

The `insertError` element appears as follows in the **Properties View**:

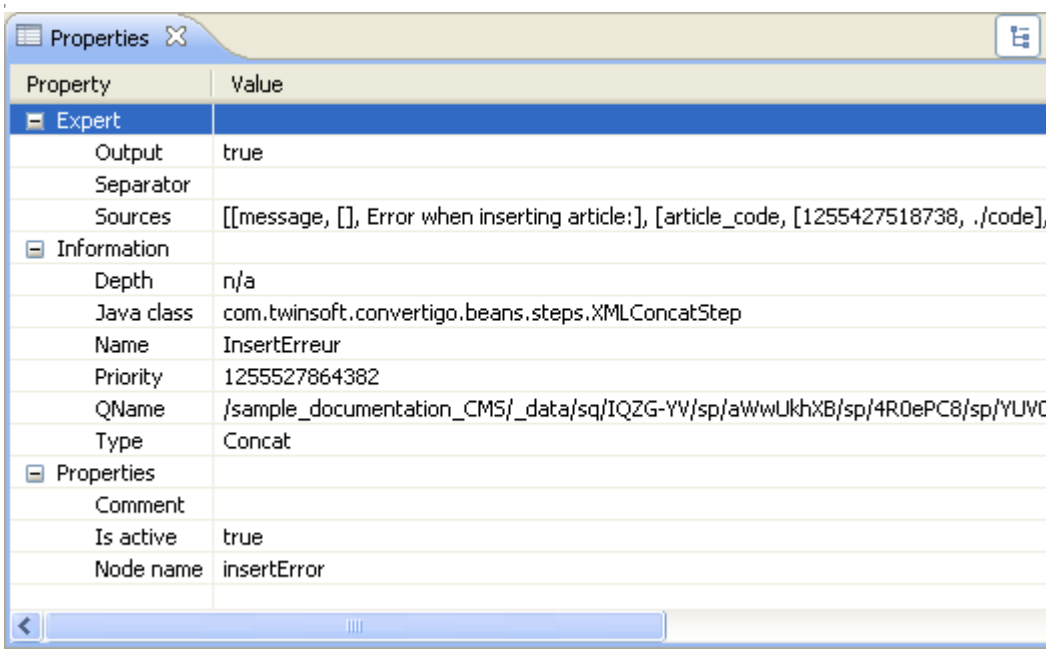


Figure 2 - 161: insertError Complex element properties

The **Output** property is set to `true`, indicating that the step generates XML content. The generated tag name (`insertError`, see Figure 2 - 161) is defined through the **Node name** property. The **Sources** property defines sources to be concatenated. They can be either fixed or variable (respectively **Default value** or **Source** columns, see Figure 2 - 162):

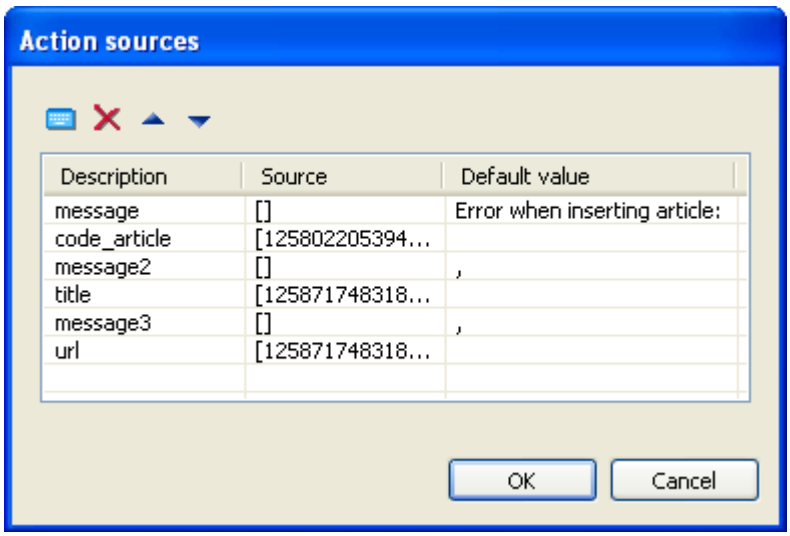


Figure 2 - 162: Action sources window (setting of the concatenated string)

DATE/TIME 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

GENERATE DATES 

OBJECT DESCRIPTION

Creates a list of XML elements containing dates based on input definitions.

Generate dates step is used to generate a list of dates. These dates are generated based upon **Input** properties.

Depending on **Split** property value, resulting dates can be:

- formatted, thanks to **Output** properties, into text in XML elements that are inserted in the sequence's XML output,
- split in several pieces of information (day of week, day date, month, year) that are added into an XML structure inserted in the sequence's XML output.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Assigned XSD Simple type QName	XmlQName	expert	Defines the schema base type to assign as a type to this simple XML element. This property allows to assign a simple XSD type to the simple XML element generated by this step. It can only be used when the step actually generates a simple XML element.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Input - Days of week	XMLVector	expert	Defines the days of week. This property defines the days of the week which dates have to be generated. Days of week are defined by numbers which can take different values depending on the **Java Calendar compatibility** property: • for Java Calendar compatible format - M:2, T:3, W:4, T:5, F:6, S:7, S:1; • for classic format - M:1, T:2, W:3, T:4, F:5, S:6, S:7. Notes: • For generating several days, separate days numbers by a comma (`","`). For example: `"2,3,4,5,6,7,1"` to generate all days with **Java Calendar compatibility** property to `true`. • The order of defined days numbers does not impact the dates generation. For example: `"2,3,4,5,6,7,1"` and `"5,2,6,4,3,7,1"` values give the same output result dates.
Input - End date	XMLVector	expert	Defines the end date using **Input - format** property value format. This property defines the date to which dates are generated (day included).

Property	Type	Category	Description
Input - Format	String	expert	Defines the input dates format. Input dates text must be formatted depending on this property. For example, if dates are entered in the following form: `09/09/2009`, the **Input - Format** property can be set to: • `MM/dd/yyyy`, with the **Input - Locale** property set to `US`, • `dd/MM/yyyy`, with the **Input - Locale** property set to `FR`. For more information on usable symbols, see Appendix "*Date format - Usable symbols*".
Input - Locale	String	expert	Defines the input dates locale. Input dates text must be formatted depending on this property. For example, with the **Input - Format** property set to `dd MMMM yyyy`, the **Input - Locale** property can be set to: • `US`, if entered dates look like `09 September 2009`, • `FR`, if entered dates look like `09 septembre 2009`.
Input - Start date	XMLVector	expert	Defines the start date using **Input - format** property value format. This property defines the date from which dates are generated (day included).
Is active	boolean	standard	Defines whether the step is active.
Java Calendar compatibility	boolean	expert	Defines whether input/output properties values are compatible with the `Java Calendar`. If this property is set to `false`, the input/output properties values use traditional calendar notations: • weekdays go from 1 to 7, • months go from 01 to 12, • days go from 01 to 31. If this property is set to `true`, the input/output properties values use Java Calendar notations: • weekdays go from 1 to 7 (but order differs from traditional calendar, see the **Input - Days of week** property), • months go from 0 to 11, • days go from 1 to 31.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.

Property	Type	Category	Description
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Output - Format	String	expert	Defines the dates output format in "`no split`" case. This property defines the resulting date format when dates are generated as texts (see **Split** property description). In this case, text of generated dates is formatted depending on **Output - Format** property. For example, if the **Output - Format** property is set to `yyyy MM dd`, the `09/09/2009` resulting date would be written: `2009 09 09`. For more information on usable symbols, see Appendix "*Date format - Usable symbols*".
Output - Locale	String	expert	Defines the dates output locale in "`no split`" case. This property defines the resulting date locale when dates are generated as texts. Text is formatted depending on this property. For example, if the date is `09/09/2009` and the resulting **Output - Format** property is set to `MMMM`, the resulting date would be written: • "September", with the **Output - Locale** property set to US, • "septembre", with the **Output - Locale** property set to FR.

Split	boolean	expert	Defines whether dates should be split into several pieces of data or written as text.
			If this property is set to `false` (i.e. "`no split`" format), each generated date is created with the following format:
			`<date>date into` **Output format** `format</date>`.
			If this property is set to `true` (i.e. "`split`" format), each generated date is created with the following format:
			`<date>`
			`<dayOfWeek>value of dayOfWeek</dayOfWeek>`
			`<day>value of day</day>`
			`<month>value of month</month>`
			`<year>value of year/year>`
			`</date>`

CONVERTIGO REQUEST STEPS

CALL TRANSACTION 

OBJECT DESCRIPTION

Defines a step invoking a transaction.

The *Call Transaction* step enables to call any existing transaction from the same project or another. It provides input variables to the target transaction, and returns XML data from the call.

Variables to be used for the call must be described at step level by adding *Variables* child objects. You can manually set variables or use the **Import variables from the target transaction** contextual menu to automatically copy the variable definitions from the target transaction.

The target transaction returns structured XML data, its XML schema has to be generated while developing the transaction and is automatically imported to the *Call Transaction* step while configuring its **Transaction** property. Thus, the transaction's schema is known by the calling step and elements from the transaction result can be correctly sourced from it.

Notes:

- A *Call Transaction* step with all its properties filled and including the target variables can be easily created at once in the Convertigo Studio **Projects** view. To do so, drag-and-drop with `Ctrl` key pressed a transaction from its parent connector to a sequence or a block step where the *Call Transaction* step has to be created.
- The client/server HTTP session of parent sequence is spread to the called transaction context, even if it is called internally (**Internal invoke** property set to `true`).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Connection string	XMLVector	expert	Replaces the connection parameters of target connector. The connection string represents different data depending on connector type: • *HTTP / HTML connector*: replaces the connector URL string made up of the connector's **Server** name, server **Port**, **Root path** and transaction's **Sub path** properties. • *Javelin connector*: replaces the address set in the **Connection address** property, made up of **Connection parameter**, **Host name**, host **Port** and **Connection type** sub properties.

Property	Type	Category	Description
Context name	JS expression	standard	Defines the specific context name to use (one is automatically created otherwise). This property is a JavaScript expression that is evaluated at sequence execution. If not empty, the computed context name is appended to current session's `JSessionID` to define the context ID of the context that is created. The execution context of called transaction / sequence is named: • after the **Context name** property of the *Call Transaction* / *Call Sequence* step, • automatically thanks to parent sequence parameters, if the **Context name** property is not specified. Every automatically named context will be deleted at the end of the sequence execution. Explicitly named contexts will remain for further transaction or sequence use. To re-use a named context, call the transaction / sequence in the same session and pass the context name through: • the **Context name** property of *Call Transaction* / *Call Sequence* step, • the `__context` parameter sent to Convertigo while calling the transaction / sequence. Note: The creation or the destruction of context is effective in server mode only.
Internal invoke	boolean	standard	Defines if the called transaction/sequence should be called internally (through the Convertigo engine) or externally (i.e. via the web application server, in HTTP). Since version 6.3.3 of Convertigo, the HTTP session of parent sequence is spread to called transaction/sequence context even if the transaction/sequence is called using internal invoke.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Transaction	String	standard	Defines the target project, connector from this project and transaction to request. The target transaction must be one of the transactions of one of the connectors from an existing project, the project in which the *Call Sequence* step is added or another project opened in the same Convertigo. This property is set by selecting the target transaction in a list of values of the following form: `<project_name>.<connector_name>.<transaction_name>` to avoid mistakes in case of transactions with the same name in several projects.

EXAMPLES

Example 1

Let's consider a `SearchOneKeyword` sequence, which defines two variables named `input` and `maxResult.` This sequence calls the `searchGoogleWithLimit` transaction, set in the context of the *"Starting With Convertigo Web Integrator"* tutorial, with its `input` variable as `keyword` and a hard-coded `maxPages` variable value. The sequence then tests whether the transaction has returned an error message or not. In the first case, it ends by raising an Exception, in the second case, it iterates on the results list and copies to sequence output the `maxResult` first items.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Sequencer objects examples*** *in the* ***New Project*** *wizard.*
Associated project can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

In order to call the `searchGoogleWithLimit` transaction, a *Call Transaction* step is created with the following parameters:

```
Call Transaction [
  transaction=
sample_documentation_CWI.GoogleConnector.searchGoogleWithLimit
  internal invoke=true
  output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

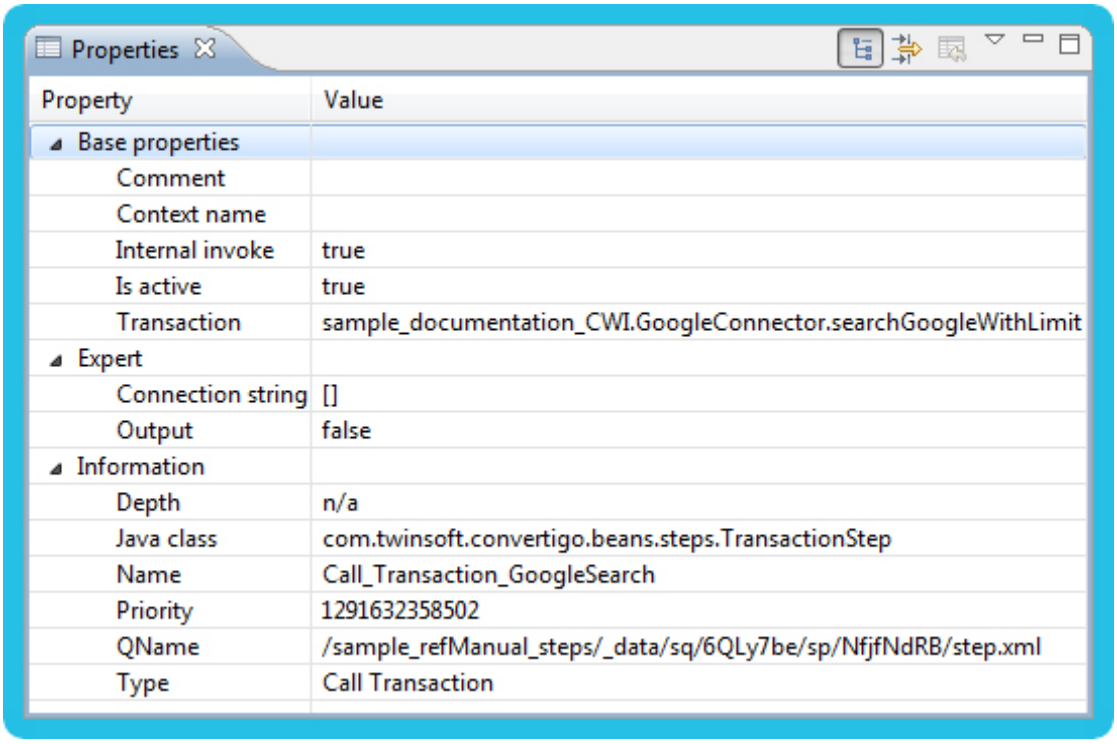


Figure 2 - 163: Call Transaction step - Configuration example

The step is created in the **Steps** folder of the sequence, next to various other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

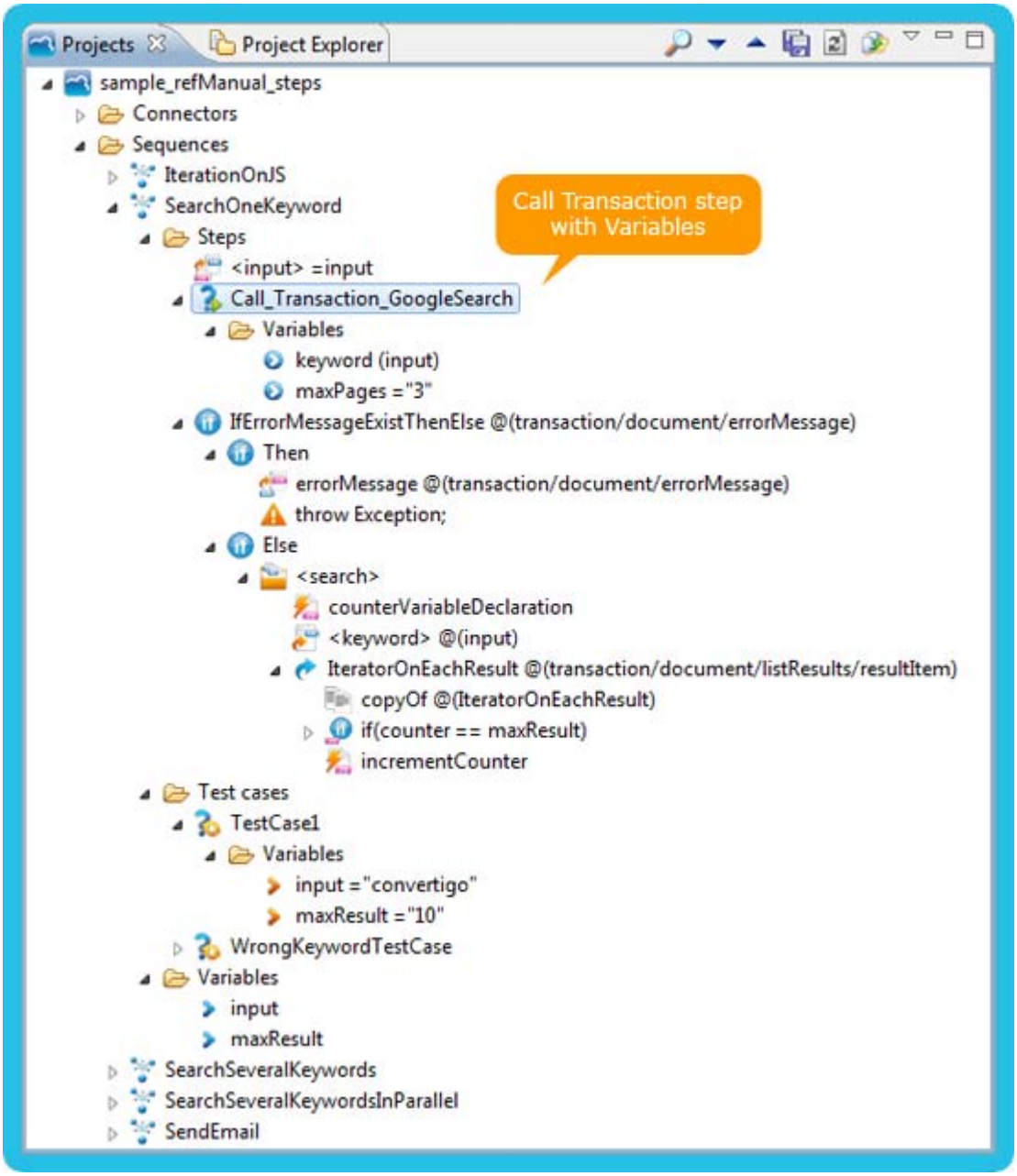


Figure 2 - 164: Call Transaction step - Object in Projects view

Two *Variable* objects are created in the *Call Transaction* step's **Variables** folder, in order to pass the variables values defined in the sequence to the called transaction. They are created by using the **Import variables from target transaction** right-click context menu on the *Call Transaction* step. For more information about these variables creation and parametrization, see *Step Variable* object documentation.

When executing one of the test cases defined for the sequence, the sequence calls the `searchGoogleWithLimit` transaction with `input` variable passed as `keyword` and `maxPages` to fixed value "3". Then, the sequence gets the transaction result and continues executing.

Example 2

Let's consider the `GetXMLData` sequence set in the context of the "*Starting with Convertigo*

Mashup Sequencer" tutorial.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Mashup Sequencer" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Mashup sequencing*** *in the* ***New Project*** *wizard.*
Associated projects can be opened by selecting ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Legacy integration*** *and* ***Web integration*** *in the* ***New Project*** *wizard.*

This sequence contains a *Call Transaction* step called `Call_Transaction_GetArticleData` which purpose is to call the `GetArticleData` transaction set in the context of the "*Starting with Convertigo Legacy Integrator*" tutorial.

The `Call_Transaction_GetArticleData` step appears as follows:

- in the **Projects** view of the Convertigo Studio:

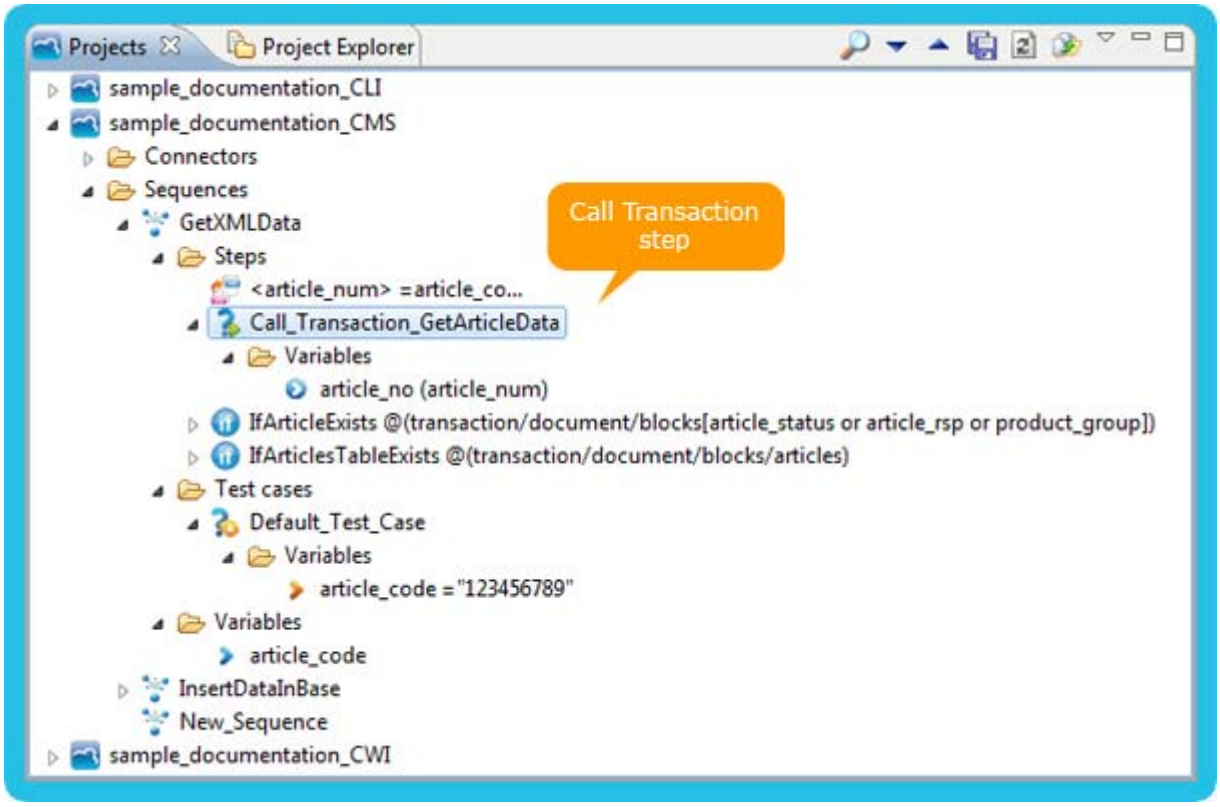


Figure 2 - 165: Call_Transaction_GetArticleData step in GetXMLData Sequence

- in the **Properties** view of the Convertigo Studio:

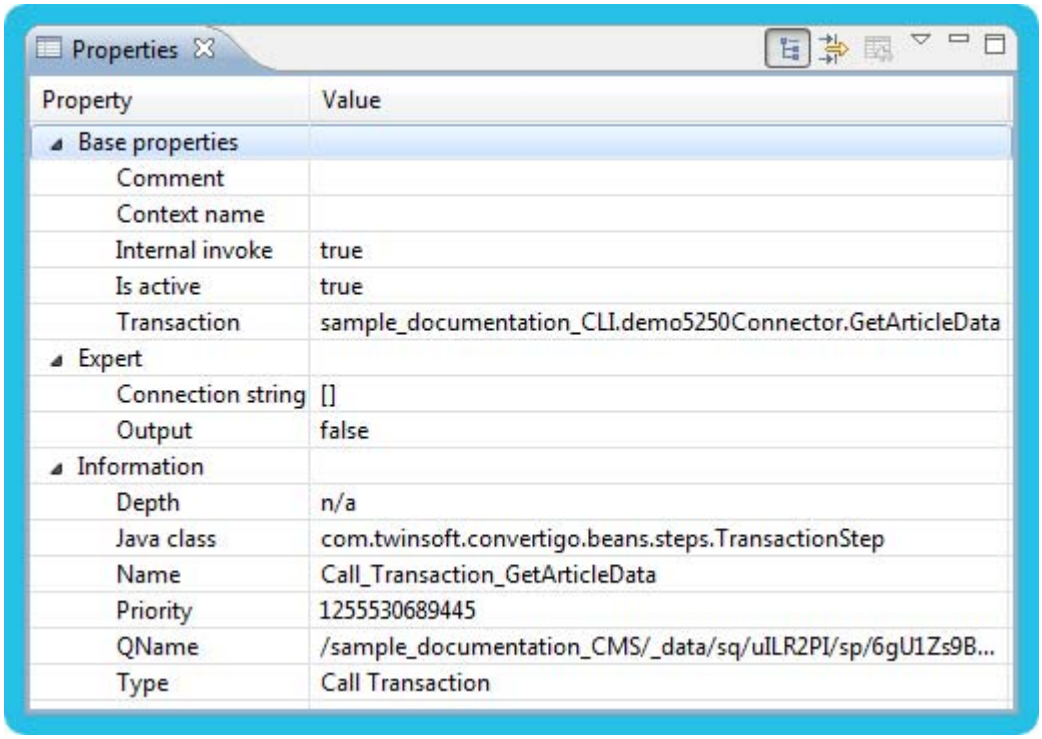


Figure 2 - 166: Call_Transaction_GetArticleData step properties

The purpose of the sequence being to generate a well structured XML output with limited information only, the whole `GetArticleData` transaction XML output do not need to be added to the sequence XML output. The *Call Transaction* step **Ouput** property is therefore set to `false`.

The **Connection string** value is empty because we do not need the default connection string (made up of initial connector parameters) to be overridden by a replacement value.

The transaction's result can the be used as a source by steps set further down the sequence, thanks to its schema that was set when the transaction was created.

CALL SEQUENCE 

OBJECT DESCRIPTION

Defines a step invoking a sequence.

The *Call Sequence* step enables to call any existing sequence from the same project or another. It provides input variables to the target sequence, and returns XML data from the call.

Variables to be used for the call must be described at step level by adding *Variables* child objects. You can manually set variables or use the **Import variables from the target sequence** contextual menu to automatically copy the variable definitions from the target sequence.

The target sequence returns structured XML data, its XML schema has to be generated while developing the sequence and is automatically imported to the *Call Sequence* step while configuring its **Sequence** property. Thus, the sequence's schema is known by the calling step and elements from the sequence result can be correctly sourced from it.

Notes:

- A *Call Sequence* step with all its properties filled and including the target variables can be easily created at once in the Convertigo Studio **Projects** view. To do so, drag-and-drop with `Ctrl` key pressed a sequence from its parent project to a sequence or a block step where the *Call Sequence* step has to be created.
- The client/server HTTP session of parent sequence is spread to the called sequence context, even if it is called internally (**Internal invoke** property set to `true`).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Context name	JS expression	standard	Defines the specific context name to use (one is automatically created otherwise). This property is a JavaScript expression that is evaluated at sequence execution. If not empty, the computed context name is appended to current session's `JSessionID` to define the context ID of the context that is created. The execution context of called transaction / sequence is named: • after the **Context name** property of the *Call Transaction* / *Call Sequence* step, • automatically thanks to parent sequence parameters, if the **Context name** property is not specified. Every automatically named context will be deleted at the end of the sequence execution. Explicitly named contexts will remain for further transaction or sequence use. To re-use a named context, call the transaction / sequence in the same session and pass the context name through: • the **Context name** property of *Call Transaction* / *Call Sequence* step, • the `__context` parameter sent to Convertigo while calling the transaction / sequence. Note: The creation or the destruction of context is effective in server mode only.
Inherit context	boolean	standard	Defines whether the context used by the current sequence for transaction's steps should also be used by the target sequence. Sequences are executing all child transactions (transactions called thanks to *Call transaction* steps) in a context automatically created (except for transactions called thanks to a *Call transaction* step with **Context** property set). For other child transactions, the automatically created context can be passed to a child sequence (called thanks to a *Call Sequence* step) for it to re-use this context for executing its child transactions. To do so, set this property to `true`.
Internal invoke	boolean	standard	Defines if the called transaction/sequence should be called internally (through the Convertigo engine) or externally (i.e. via the web application server, in HTTP). Since version 6.3.3 of Convertigo, the HTTP session of parent sequence is spread to called transaction/sequence context even if the transaction/sequence is called using internal invoke.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Property	Type	Category	Description
Sequence	String	standard	Defines the target project and sequence to request from this project. The target sequence must be one of the sequences from an existing project, the project in which the *Call Sequence* step is added or another project opened in the same Convertigo. This property is set by selecting the target sequence in a list of values of the following form: `<project_name>.<sequence_name>` to avoid mistakes in case of sequences with the same name in several projects.

FILE MANAGEMENT STEPS

READ XML 

OBJECT DESCRIPTION

Reads an XML file content and loads it into the step's XML.

The *Read XML* step reads any XML file and loads its content. As a consequence, the content of the XML file is available as a source for other following steps.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	JS expression	expert	Defines the path of the file to read. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file to read. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

READ CSV 

OBJECT DESCRIPTION

Reads a CSV file content and loads it into the step's XML.

The *Read CSV* step reads any CSV file and loads its content as an XML. As a consequence, the content of the CSV file is available as a source for other following steps.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Column tag	String	standard	Defines the column tag name. Any tag name to use for columns in XML can be configured using this property. Default value is `col`.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Encoding	String	standard	Defines the encoding used in the CSV file. Default value for encoding is `iso-8859-1`.
Is active	boolean	standard	Defines whether the step is active.
Line tag	String	standard	Defines the lines tag name. Any tag name to use for lines in XML can be configured using this property. Default value is `line`.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Separator	String	expert	Defines the CSV default separator symbol. Any separator character can be configured using this property. Default value is {{Computer}};{{-Computer }}.
Source	JS expression	expert	Defines the path of the file to read. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file to read. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

Property	Type	Category	Description
Title line	boolean	expert	Defines whether the CSV file has a title line or not. If set to `true`, the first line of the CSV file is handled as a title line which means that each cell of the first line is used as tag name for the following lines, containing content. More precisely, for each cell of the first line, if the cell contains data, the tag associated with the corresponding column is named after this data. Otherwise or if the property is set to `false`, the tag is named after the **Column tag** property.
Vertical direction	boolean	expert	Defines the array reading direction. If set to `true`, the reading direction is vertical. Otherwise, it is horizontal.

WRITE XML 

OBJECT DESCRIPTION

Writes XML content in an XML file.

The *Write XML* step allows outputting XML content in an XML file on the disk. It can either create a new XML file or update an existing XML file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Append Result	boolean	expert	Defines whether the XML must be appended at the end of the file. If set to `true`, and if the file exists, the step appends the XML at the end of the file. If set to `false`, it overrides the current file content.
Append timestamp	boolean	standard	Defines whether the file name should be created with a timestamp. If set to `true`, the date is concatenated to the file name in `yyyymmddHHmmssSSS` format.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default root tag name	String	standard	Defines the root element tag name of the resulting XML to be written, if none is defined by the source. Setting this property allows adding a root element named after this property value in the XML file written.
Encoding	String	standard	Defines the encoding used in output file. Default used encoding is `ISO-8859-1`.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Output file	JS expression	standard	Defines the output file path including the file name. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path and name of the file to write. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

Property	Type	Category	Description
Source	XMLVector	expert	Defines the source data to write. This property allows defining a list of nodes from a previous step used as data root to be written in the file. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the XML output document of the sequence (i.e., sequence resulting XML) is used as source. In this case, the step behavior can be seen as a sequence output dump. If REST or SOAP interfaces are used to call parent sequence, the XML output document is normally returned to the sequence caller.

WRITE CSV 

OBJECT DESCRIPTION

Writes XML content in a CSV file.

The *Write CSV* step allows outputting XML content in a CSV file on the disk. It can either create a new CSV file or update an existing CSV file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Append Result	boolean	expert	Defines whether the XML must be appended at the end of the file. If set to `true`, and if the file exists, the step appends the XML at the end of the file. If set to `false`, it overrides the current file content.
Append timestamp	boolean	standard	Defines whether the file name should be created with a timestamp. If set to `true`, the date is concatenated to the file name in `yyyymmddHHmmssSSS` format.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Encoding	String	standard	Defines the encoding used in output file. Default used encoding is `ISO-8859-1`.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Output file	JS expression	standard	Defines the output file path including the file name. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path and name of the file to write. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
Separator	String	expert	Defines the CSV separator symbol to be used. By default, it uses the character `;` as separator in the CSV file.

Property	Type	Category	Description
Source	XMLVector	expert	Defines the source data to write. This property allows defining a list of nodes from a previous step used as data root to be written in the file. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the XML output document of the sequence (i.e., sequence resulting XML) is used as source. In this case, the step behavior can be seen as a sequence output dump. If REST or SOAP interfaces are used to call parent sequence, the XML output document is normally returned to the sequence caller.
Title line	boolean	expert	Defines whether data tags are named after the first line of data (titles). If set to `true`, the first line of the file is handled as a title line. For each XML tag providing data in this line, the tag content is added as title in the title line. If set to `false`, no title line is defined in the data nor in the file.

WRITE BINARY FROM BASE64 

OBJECT DESCRIPTION

Writes a binary file from a Base64 content.

The *Write binary from Base64* step allows writing a `Base64` content from a response XML in a binary file on the disk.

It can either create a new file or update an existing file, if a file of the same path and name already exists.

The file extension has to be defined: it corresponds to the type of binary file to write. It can be set in the **Output file** property, at the end of the file path.

A `Base64` content must be used as input, defined by the **Source** property. Such input content could be picked in output XML of transactions, for example:

- in an *HTML transaction* XML response: using the *Print screen* extraction rule or the *Get attachment* statement in the transaction would add `Base64` content to output,
- in an *SQL transaction* XML response: if the database contains a column with `Base64` data, this content would be present in transaction output,
- etc.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Append timestamp	boolean	standard	Defines whether the file name should be created with a timestamp. If set to `true`, the date is concatenated to the file name in `yyyymmddHHmmssSSS` format.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Property	Type	Category	Description
Output file	JS expression	standard	Defines the output file path including the file name. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path and name of the file to write. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
Source	XMLVector	expert	Defines the source data to write. This property allows defining a list of nodes from a previous step used as data root to be written in the file. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the XML output document of the sequence (i.e., sequence resulting XML) is used as source. In this case, the step behavior can be seen as a sequence output dump. If REST or SOAP interfaces are used to call parent sequence, the XML output document is normally returned to the sequence caller.

COPY FILE 

OBJECT DESCRIPTION

Copies a file or a directory to an another path.

The *Copy file* step duplicates a file or a directory from a path to another keeping the same name.

Note: **Source** parent folder and **Destination** folder cannot be the same.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Destination	JS expression	expert	Defines the destination directory path. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the destination folder, that must be an existing folder. Otherwise, the copy will not be possible. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Overwrite	boolean	expert	If a file or folder with the same name exists in **Destination** directory, this property defines whether to overwrite it. By default this property is set to `false`, so the file or folder will not be overwritten if already present in **Destination** directory.
Source	JS expression	expert	Defines the path of the file or directory to copy. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory to copy. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

DUPLICATE FILE 

OBJECT DESCRIPTION

Duplicates a file or a directory in the same path.

The *Duplicate file* step duplicates a file or a directory in a given path updating its name.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Name of the copy	JS expression	expert	Defines the name of the duplicated object (file or directory). Duplicating in the same parent folder, the copied file or directory name must be updated. This name must be different from original file or directory name.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Overwrite	boolean	expert	If a file or folder with the same name as the **Name of the copy** property exists in current directory, this property defines whether to overwrite it. By default this property is set to `false`, so the previously existing file or folder will not be overwritten if already present in current directory.
Source	JS expression	expert	Defines the path of the file or directory to duplicate. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory to duplicate. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

MOVE FILE 

OBJECT DESCRIPTION

Moves a file or a directory to an another path.

The *Move file* step copies a file or a directory from a path to another keeping the same name and removes the original one.

Note: **Source** parent folder and **Destination** folder cannot be the same.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Destination	JS expression	expert	Defines the destination directory path. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the destination folder, that must be an existing folder. Otherwise, the copy will not be possible. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Overwrite	boolean	expert	If a file or folder with the same name exists in **Destination** directory, this property defines whether to overwrite it. By default this property is set to `false`, so the file or folder will not be overwritten if already present in **Destination** directory.
Source	JS expression	expert	Defines the path of the file or directory to move. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory to move. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

RENAME FILE 

OBJECT DESCRIPTION

Renames a file or a directory.

The *Rename* step renames a file or a directory.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
New name	JS expression	expert	Defines the new name for file or directory.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Overwrite	boolean	expert	Defines whether the destination file or directory should be overwritten if exists.
Source	JS expression	expert	Defines the path of the file or directory to rename. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory to rename. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

DELETE FILE 

OBJECT DESCRIPTION

Deletes a file or a directory.

The *Delete* step removes a file or a directory.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	JS expression	expert	Defines the path of the file or directory to delete. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file or directory to delete. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

CREATE DIRECTORY 

OBJECT DESCRIPTION

Creates a new directory.

The *Create directory* step creates a new directory on disk.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Create non existent parent directories	boolean	expert	Defines whether the non existent but necessary parent directories should be created or not. By default, this property is set to `true`: parents directories specified in path but not existing on disk are also created. If set to `false`, the directory will be created only if all parent directories are existing.
Destination	JS expression	expert	Defines the destination path. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the destination folder. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

LIST DIRECTORY 

OBJECT DESCRIPTION

Defines a step able to list the entries of a directory.

A *List directory* step lists all the non hidden files of the first level contained in the directory specified by the **Source directory** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source directory	JS expression	standard	Defines the source directory path. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the directory which content has to be listed. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

HTTP SESSION MANAGEMENT

SET AUTHENTICATED USER 

OBJECT DESCRIPTION

Sets a user ID as the authenticated user ID of the current context/session.

The *Set authenticated user* step allows to set a user ID as the authenticated user ID in the current context/session and thereby, sets the current context/session as authenticated.

The user ID is set using the **User ID** property.

Note: Although its **Output** property is set to `false` by default, this step generates an `authenticatedUserID` XML Element in output, that should always contain the user ID value if the step succeeds.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

Property	Type	Category	Description
User ID	SmartType	standard	Defines the user ID that has to be set as authenticated user. This property is a "smart type" property, that allows to define the user ID to set in authentication. A "smart type" property can be of one of the following types: • a **text**: the value is therefore a default text value, • a **JavaScript expression**: the value is therefore a JavaScript expression that is evaluated at sequence execution, • a **source**: the value is a source and can be picked using the source picker. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. Note: If you use the **source** type for this property, the XPath application on target XML should give a text result. Otherwise, the first node's text content is taken.

GET AUTHENTICATED USER 

OBJECT DESCRIPTION

Gets the authenticated user ID from the context/session.

The *Get authenticated user* step allows to retrieve in an XML Element the authenticated user ID from the context/session, if the context/session is authenticated. Otherwise, it returns an empty value.

The element is named after the value of the **Node name** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

REMOVE AUTHENTICATED USER 

OBJECT DESCRIPTION

Removes the authenticated user ID from the context/session.

The *Remove authenticated user* step allows to remove the authenticated user ID from the context/session. The context/session is not authenticated anymore.

Note: Although its **Output** property is set to `false` by default, this step generates an `authenticatedUserID` XML Element in output, that should always be empty if the step succeeds.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

GET FROM SESSION 

OBJECT DESCRIPTION

Gets a stored variable/object from the session.

The *Get from session* step allows to easily retrieve a value previously stored (thanks to the *Set in session* step for example) using its key.

Note: The HTTP session is shared by all contexts that are executed for a same user's requests.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Key	String	standard	The key of the variable/object to retrieve, i.e. the stored variable name. The variable/object was stored in session using a key, also called name. This property allows to specify the name of the variable/object to retrieve.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

SET IN SESSION 

OBJECT DESCRIPTION

Stores a variable/object in the session.

The *Set in session* step allows to easily store a value that will be recoverable using its key.

Note: The HTTP session is shared by all contexts that are executed for a same user's requests.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Key	String	standard	The key of the variable/object to store in session, i.e. the variable name. The variable/object to store in session is identified by a key, also called name. This property allows to specify the name of the variable/object to store (in order to be recoverable later using the same key, for example using the *Get from session* step).
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Value	SmartType	standard	The variable/object to store in session, i.e. the value. This property is a "smart type" property, that allows to specify the variable/object to store in session. A "smart type" property can be of one of the following types: • a **text**: the value is therefore a default text value, • a **JavaScript expression**: the value is therefore a JavaScript expression that is evaluated at sequence execution, • a **source**: the value is a source and can be picked using the source picker. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution.

OTHERS

INPUT VARIABLES 

OBJECT DESCRIPTION

Defines an XML element containing dynamically the input variables of parent **Sequence**.

Placed at the beginning of a **Sequence**, this step allows steps ordered after to use the **Sequence** input variables as source.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the step is active.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

SMTP SEND 

OBJECT DESCRIPTION

Defines a step able to send emails through an SMTP server.

The *SMTP send* step is used to make the sequence send an email to designated email addresses through an SMTP server. This is useful for monitoring a sequence progress or completion.

When executed, an *SMTP send* step tries to send an email using a specified set of parameters. If ever the specified SMTP server does not support relaying or anonymous sending, the *SMTP send* step supports authentication.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attachments	XMLVector	standard	Defines a list of file attachments to send with the email. This property is an array of files to send as email attachments. Each email attachment is a pair of values: • **Filepath**: the path of the local file to send, including its original name, defined as a JavaScript expression that is evaluated during the sequence execution, • **Filename**: the name of the file as attached in the email, defined as a JavaScript expression that is evaluated during the sequence execution. The filepaths are either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. Note: A new attachment can be added to the list using the blue keyboard icon. The attachments defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Authentication type	SmtpAuthType	expert	Defines the SMTP authentication type. You can choose the authentication used by the *SMTP send* step amongst the following types: • `None`: no authentication, this value is set by default, • `Basic`: basic authentication, • `STARTTLS`: authentication using STARTTLS, • `SSL/TLS`: authentication using SSL/TLS. All authentication types use the username and password set in the **SMTP user** and **SMTP password** properties.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Content-type	JS expression	expert	Defines the content-type of the email content. This property is a JavaScript expression that is evaluated during the sequence execution and allows to override the default content-type. If this property is left empty, the default content-type is: • `text/plain; charset=UTF-8` in standard text email, • `text/html; charset=UTF-8` in HTML content email, i.e. if an XSL file is defined in the **XSL file** property.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Recipients email addresses	JS expression	standard	Defines recipient email addresses. This property is a JavaScript expression that is evaluated during the sequence execution and gives the list of recipient email addresses. This property contains a list of email addresses, separated by semi-colons or commas. The syntax to use is of the following form: `<type>:<email address>`, where `<type>` can be `To, Cc` or `Bcc`. For example, `To:myself@mydomain.com`. Notes: If not specified, the first address is always considered the main recipient (`To`), following addresses are considered secondary recipients (`Cc`).
SMTP password	String	expert	Defines the SMTP server authentication user password. Used alongside **SMTP user** to authenticate on the SMTP server. To prevent authentication, leave both **SMTP user** and **SMTP password** properties empty. Convertigo then establishes anonymous connection on the SMTP server.
SMTP port	String	expert	Defines the listening port of the SMTP server. Default is `25` for non-auth servers, it can be `587` or `465` for TLS/SSL or STARTTLS servers.
SMTP server	String	standard	Defines the name or IP address of the SMTP server. This server must be able to deliver emails to the domains used in recipients addresses. In some cases, you may have to use authentication.
SMTP user	String	expert	Defines the SMTP server authentication username. If this parameter is used, the step tries to authenticate on the SMTP server using it along with **SMTP password**.

Property	Type	Category	Description
Sender email address	String	expert	Defines the email address of the sender. This property is a text that contains an email address, but can also accept a value of this form `Convertigo <noreply@fakedomain.fake>` to add the name of the email address owner. It is useful if you want the receiver(s) to be able to answer the received email. This property is used depending on the SMTP server, it can be: • informative and have no consequence in the email sending, • automatically replaced by the SMTP server by the real email address matching the authentication, • used by the SMTP server to send the email, • etc. Consult your SMTP server documentation for more information about the `FROM` email field.
Source	XMLVector	expert	Defines the source to build email body. This property allows defining a list of nodes from a previous step used to build the email body content. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. The resulting nodes are written in the email body content depending on the nodes types: • `Attribute/Text node/Comment/CDATA section`: the node text content is directly copied to the email body content, • `Element`: the element's DOM is pretty printed in the email body content with nice indentation to easily read the XML, • `Other`: the node's DOM is pretty printed in the email body content. If the XPath doesn't match or if the source is left blank, the XML output document of the sequence (i.e., sequence resulting XML) is used as source. In this case, the step behavior can be seen as a sequence output dump.
Subject	JS expression	standard	Defines the email subject. This property is a JavaScript expression that is evaluated during the sequence execution and gives the email subject. Notes: It is recommended to not leave it empty.

Property	Type	Category	Description
XSL file	JS expression	standard	Defines the XSL file path to apply on the XML content to send an HTML email content. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path and name of the XSL file to use to transform the XML data in HTML content. This has as result to send an HTML content email instead of an XML/text email. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. If the path is empty, not XSL transformation is applied and the mail content is a plain XML/text.

EXAMPLES

The following examples of the *SMTP send* step are based on a `SendEmail` sequence from the Convertigo Mashup Sequencer project named `sample_refManual_steps`.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **Sequencer objects examples** in the **New Project** wizard.*
*Associated project, not used in this example, can be opened by selecting **Convertigo Samples and Demos** > **Documentation samples** > **Web integration** in the **New Project** wizard.*

Example 1

Let's consider the `SendEmail` sequence, which generates a small XML stucture as sequence output, including no dynamic data. After generating the XML output, we want to email the sequence result to a recipient email address. To do so, a *SMTP send* step is created with the following parameters:

```
SMTP send [
   comment: Example 1
   recipients email addresses: your_email@convertigo.com
   SMTP server: smtp.gmail.com
   subject: Convertigo sequence report
   authentication type: STARTTLS
   sender email address: c8o.exemples@gmail.com
   SMTP user: c8o.exemples@gmail.com
   SMTP password: **********
```

- `SMTP port: 587`

```
   source: [
      Complex_seq_result,
      //document/seq_result
   ]
   output=false
```

]

These parameters are edited in the **Properties** view of the Convertigo Studio:

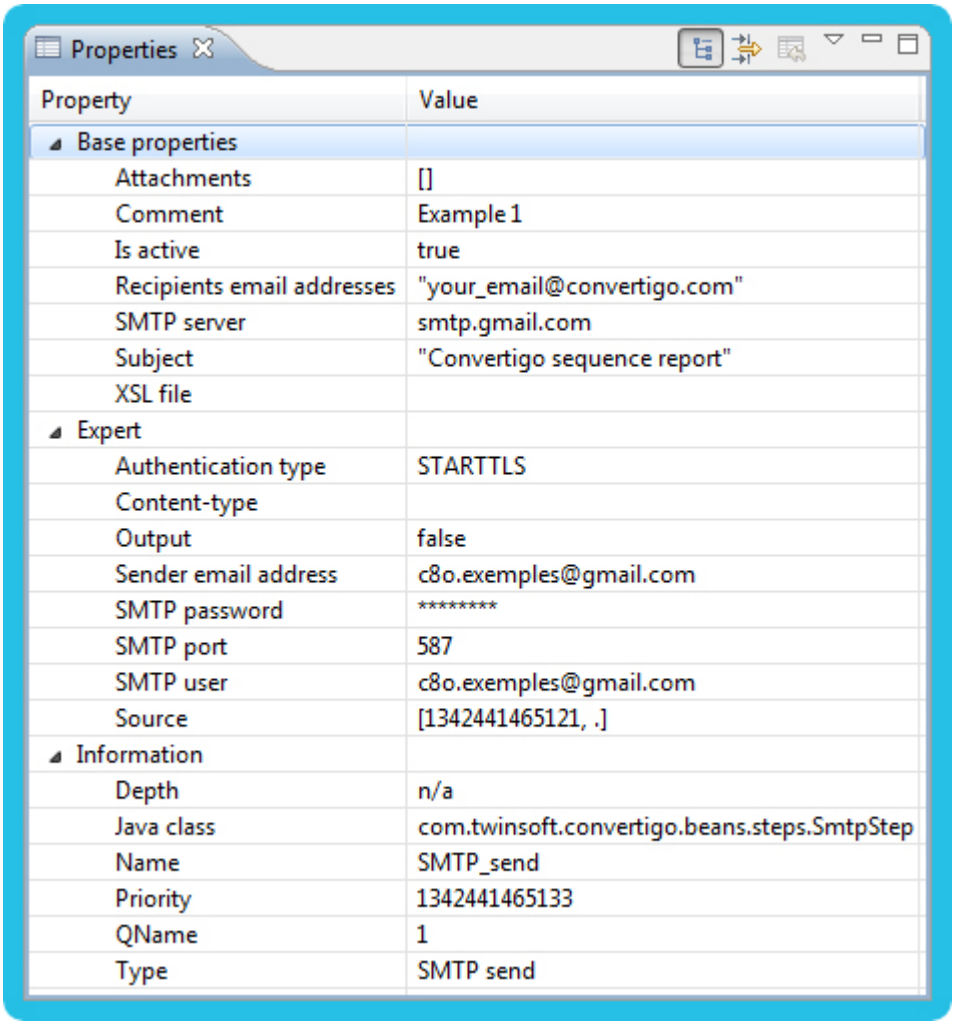


Figure 2 - 167: SMTP send step - Example 1 - Configuration example

The **SMTP server** and related parameters are configured with a Gmail examples account dedicated to these examples. This account is connected with STARTTLS authentication.

The **Recipients email addresses** parameter is configured with a fake email address. In order to test this sequence execution, you can edit this property to set your own email address. You will then receive the email.

The **Source** property points towards the root node from the previous *Complex* step, which is the basis of the XML structure generated as sequence output:

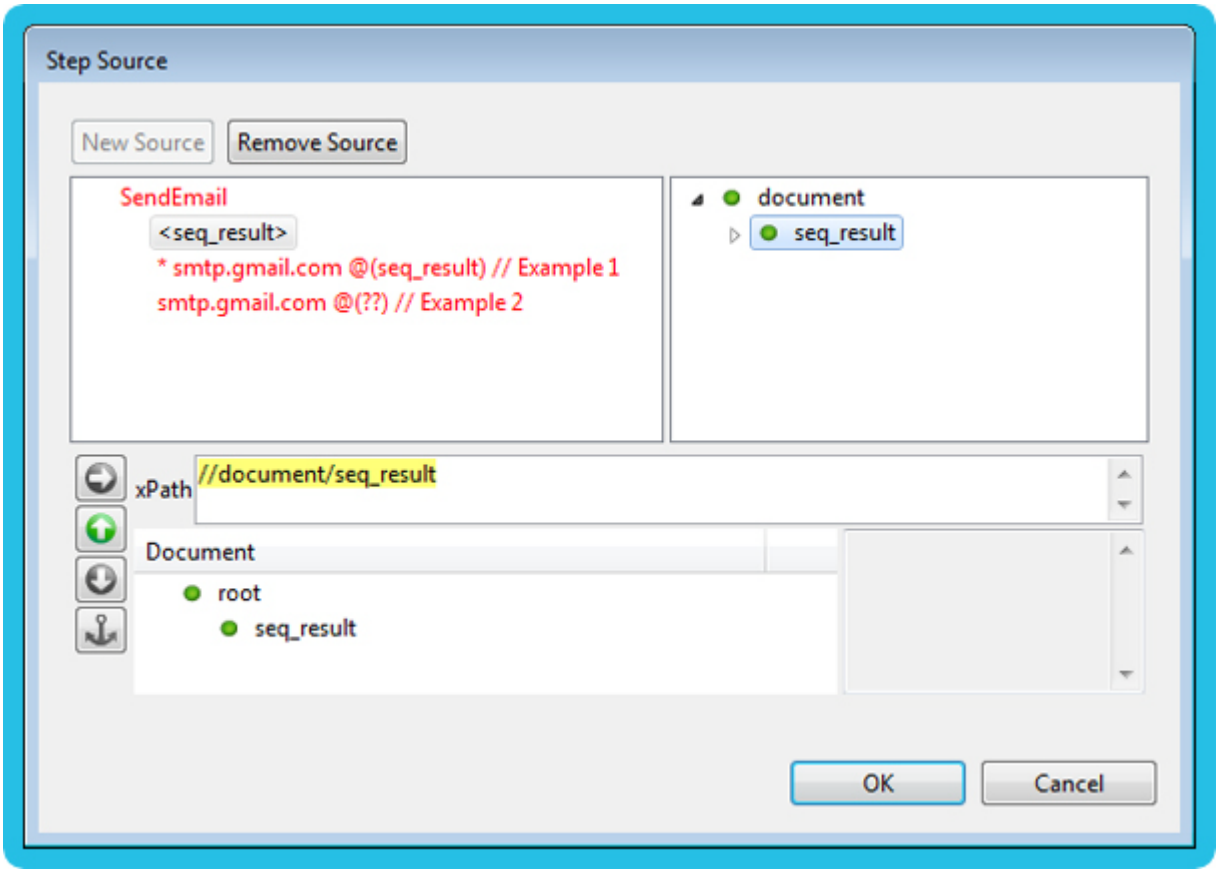


Figure 2 - 168: SMTP send step - Example 1 - Source configuration

The whole `seq_result` complex step as well as its content during execution will be send as email body.

The *SMTP send* step is created in the **Steps** folder of the sequence, next to the other steps used to implement the sequence behavior described above. It appears as follows in the **Projects** view:

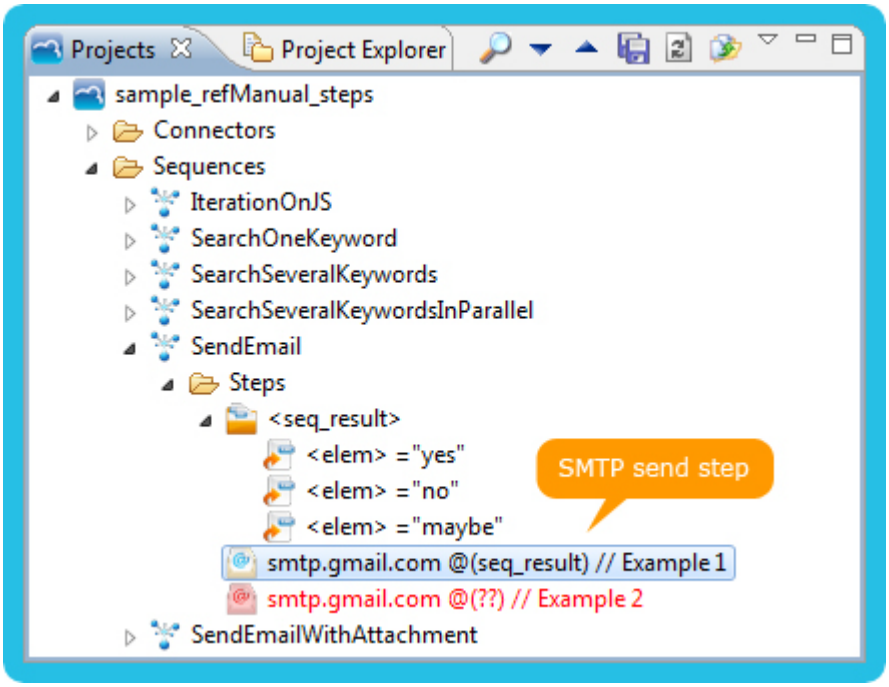


Figure 2 - 169: SMTP send step - Example 1 - Object in Projects view, with sequence and other steps

After configuring your email address as recipient, execute the sequence and check your emails to see the sent email. It is a text email of the following form:

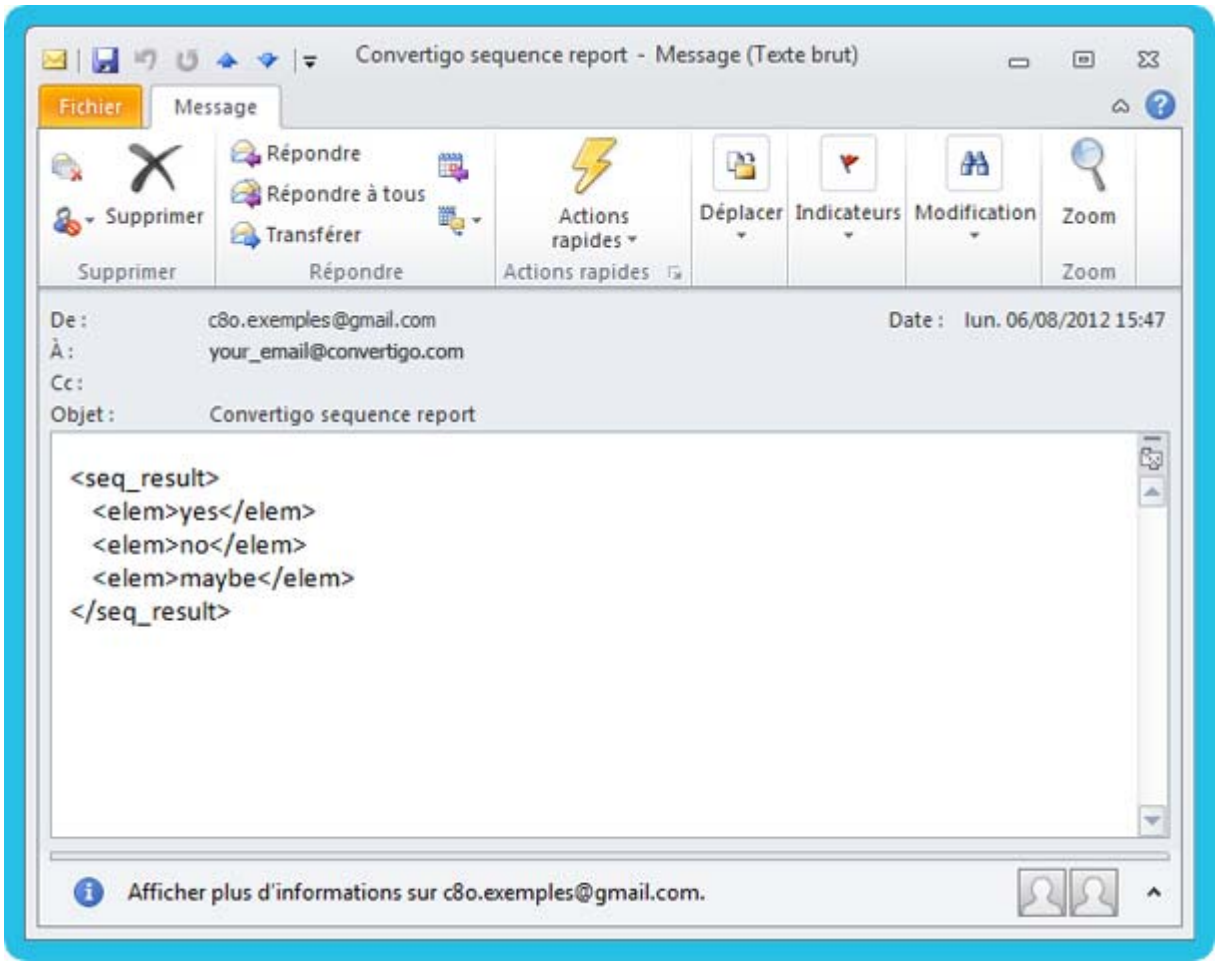


Figure 2 - 170: SMTP send step - Example 1 - Received text email

Example 2

Let's consider the same `SendEmail` sequence, which generates a small XML stucture as sequence output. This XML output is the following:

```
<document>
   <seq_result>
     <elem>yes</elem>
     <elem>no</elem>
     <elem>maybe</elem>
   </seq_result>
</document>
```

Now we want to send an email using this sequence result in order to build an HTML email. To do so, the previously generated *SMTP send* step is duplicated and updated with the following parameter values:

```
SMTP send [
   Comment: Example 2
   recipients email addresses: your_email@convertigo.com
   SMTP server: smtp.gmail.com
   subject: Convertigo sequence report
   authentication type: STARTTLS
   sender email address: c8o.exemples@gmail.com
   SMTP user: c8o.exemples@gmail.com
   SMTP password: **********
```

- `SMTP port: 587`

```
   source: []
   XSL file: ".//xsl/email.xsl"
   output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

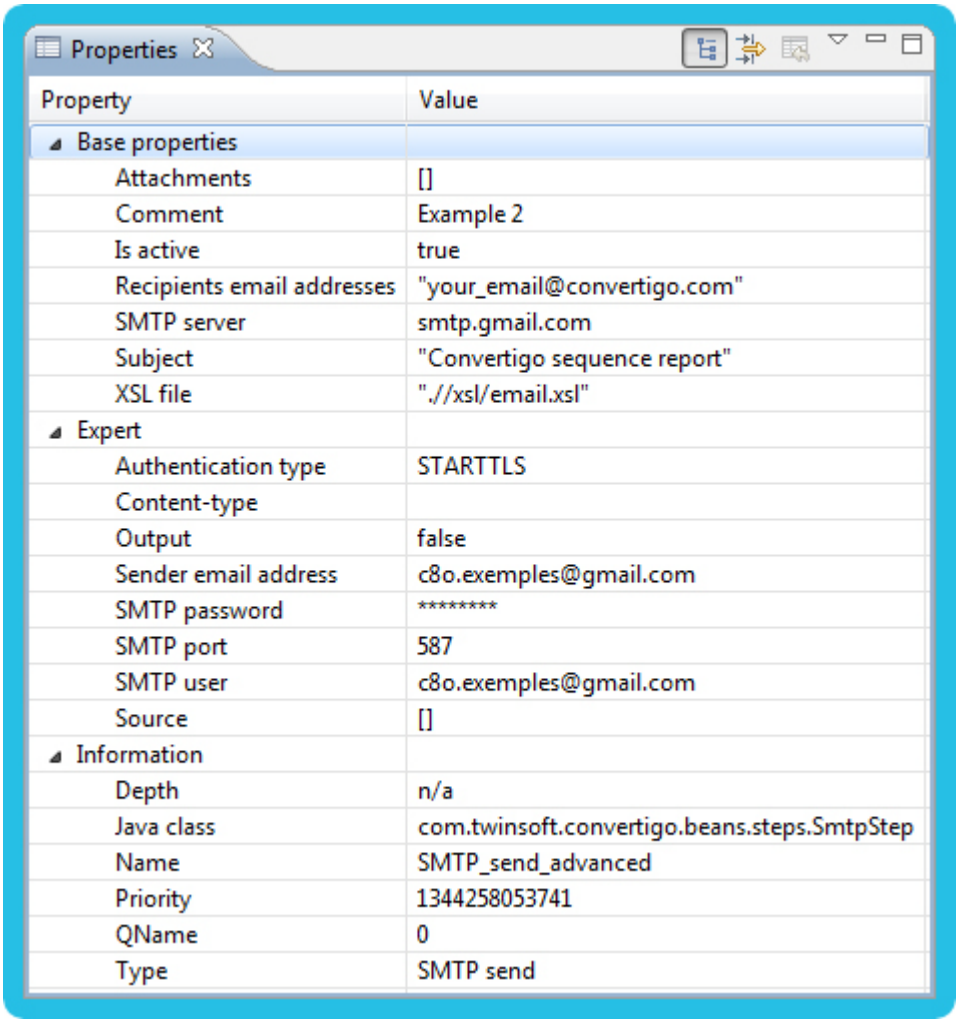


Figure 2 - 171: SMTP send step - Example 2 - Configuration example

The **XSL file** parameter is configured with a JavaScript string between quotes as the path to the XSL file to use is not dynamic. This path is relative to the project folder, it begins by `.//` and then we see that the `email.xsl` file in in a folder named `xsl`. In the Project Explorer view, we can see the file and folder:

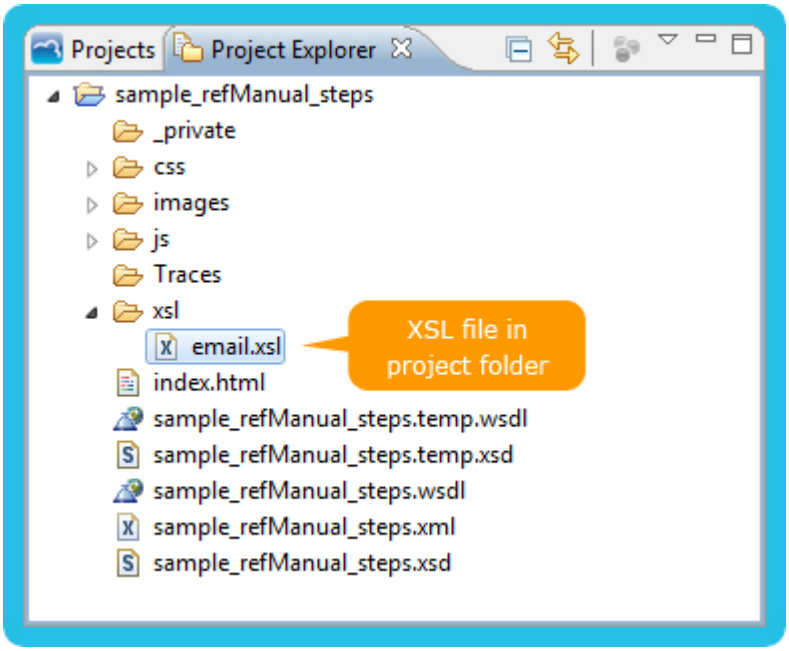


Figure 2 - 172: SMTP send step - Example 2 - XSL file in resources

The **Source** property has been unset so as the sequence XML output is used as source. The whole `document` as well as its content during execution will be used to build the email body.

The *SMTP send* step is created in the **Steps** folder of the sequence. It appears as follows in the **Projects** view:

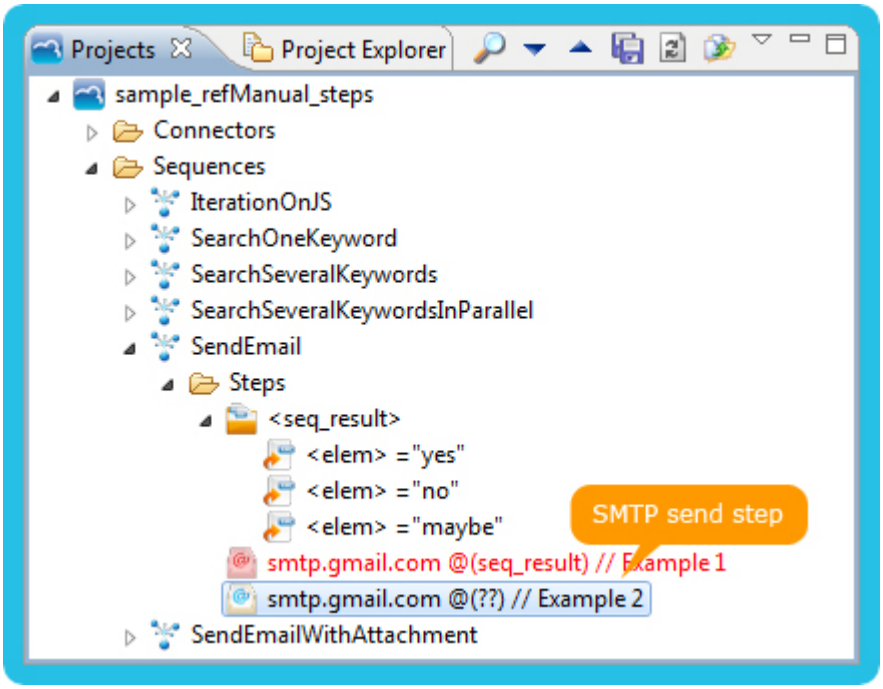


Figure 2 - 173: SMTP send step - Example 2 - Object in Projects view, with sequence and other steps

Let's have a look at the XSL file content to understand how it is applied. Opening the `email.xsl` file in the XSL editor displays the following code:

Figure 2 - 174: SMTP send step - Example 2 - XSL file content

We can easily identify the parts of the file that are applying a transformation on the XML `document` root, on the `seq_result` node and on the `elem` nodes.

After configuring your email address as recipient, execute the sequence and check your emails to see the sent email. It is an HTML email of the following form:

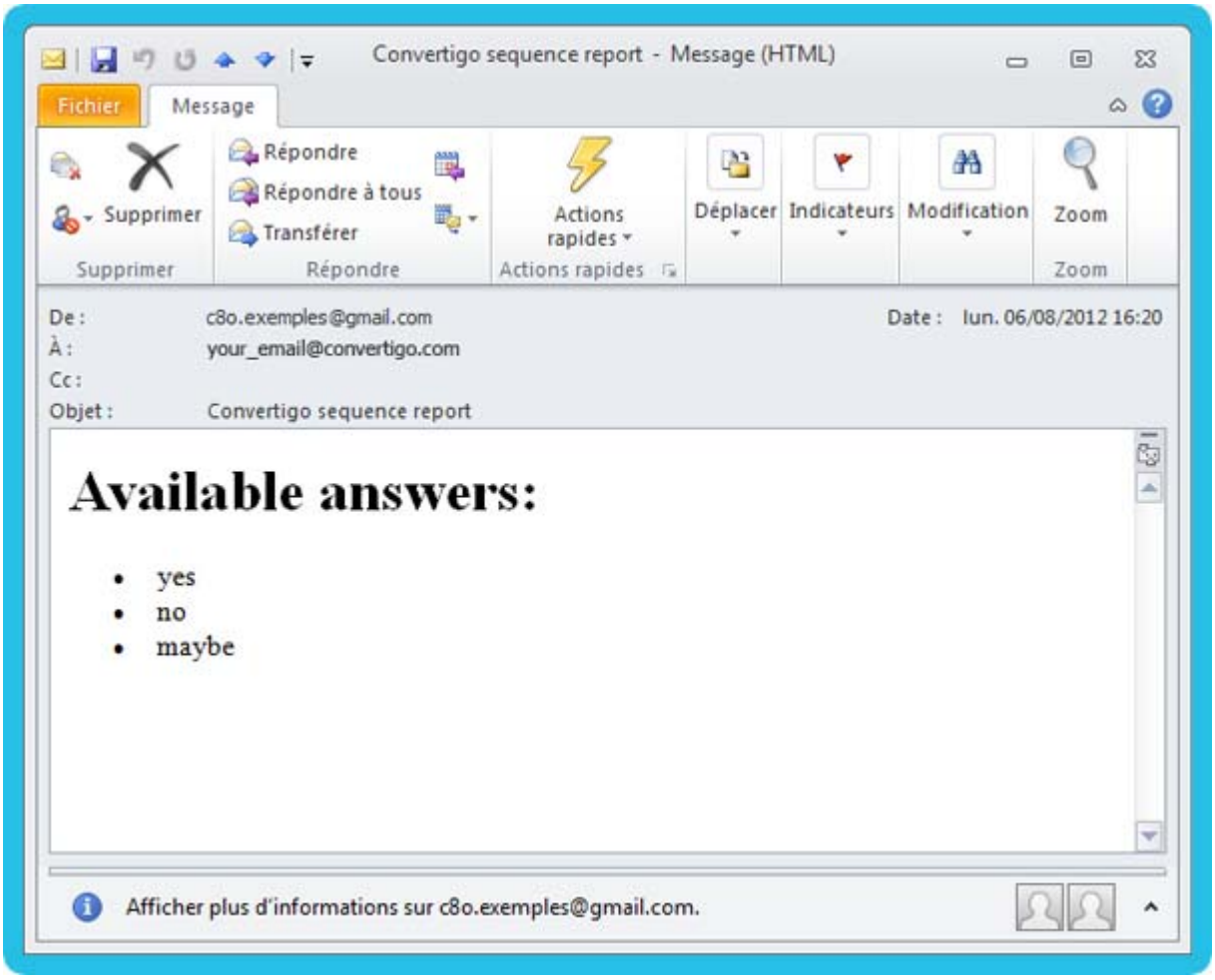


Figure 2 - 175: SMTP send step - Example 2 - Received HTML email

Example 3

Let's consider the `SendEmailWithAttachement` sequence, which generates a message in an XML Element, not included in the sequence's XML output. We want to send an email with the message contained in the Element, to a recipient email address passed as variable (named `recipient`), and including attached files. To do so, a *SMTP send* step is created with the following parameters:

```
SMTP send [
   comment: Example 3
   recipients email addresses: recipient
   SMTP server: smtp.gmail.com
   subject: Sending attachments
   authentication type: STARTTLS
   sender email address: c8o.exemples@gmail.com
   SMTP user: c8o.exemples@gmail.com
   SMTP password: **********
```

- `SMTP port: 587`

```
   source: [
      Complex_seq_result,
      //document/message/text()
```

```
  ]
  attachments: [
    filepath: .//images/New-York.jpg, filename: nyc.jpg
    filepath: .//images/New-York-Tourist-Map.jpg, filename: map.jpg
  ]
  output=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

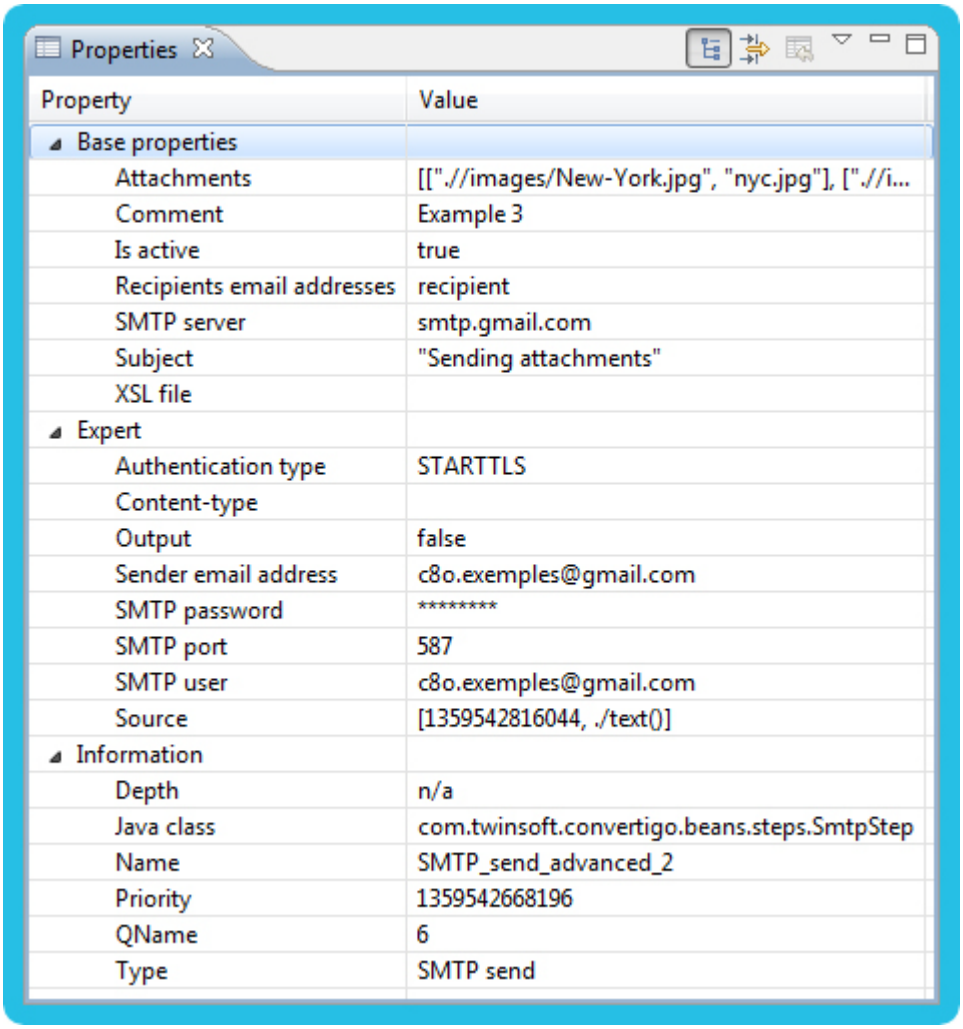


Figure 2 - 176: SMTP send step - Example 3 - Configuration example

The **Attachments** property is edited in the associated editor:

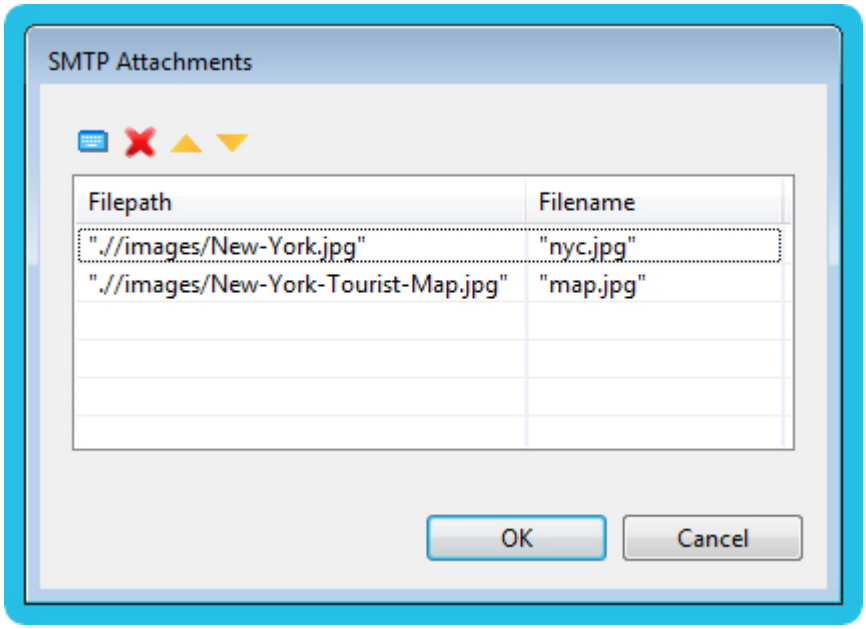


Figure 2 - 177: SMTP send step - Example 3 - Attachments property edition

Beware that the two columns of each attachement are JavaScript expressions.

The **Filepath** properties are configured with a JavaScript string between quotes: the paths of the files to attach are not dynamic. These paths are relative to the project folder, they begin by `.//` and then we see that they are located in an `images` folder. In the Project Explorer view, we can see the folder and files:

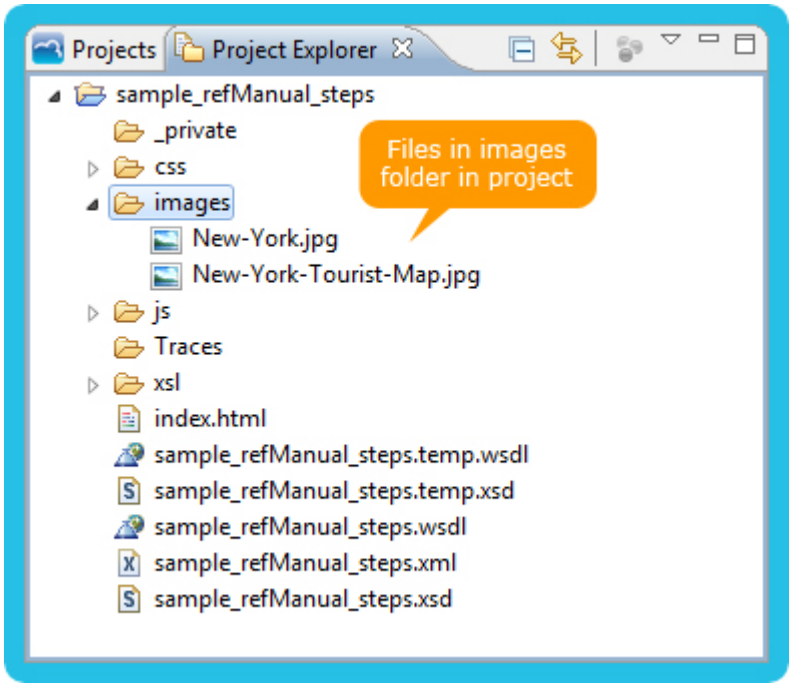


Figure 2 - 178: SMTP send step - Example 3 - Files in project resources

The **Recipients email addresses** property is configured with a JavaScript expression using

the sequence `recipient` variable. In order to test this sequence execution, you can edit the variable value in the test case created for the sequence to set your own email address. You will then receive the email.

The **Source** property points towards the text node inside the `message` *Element* step:

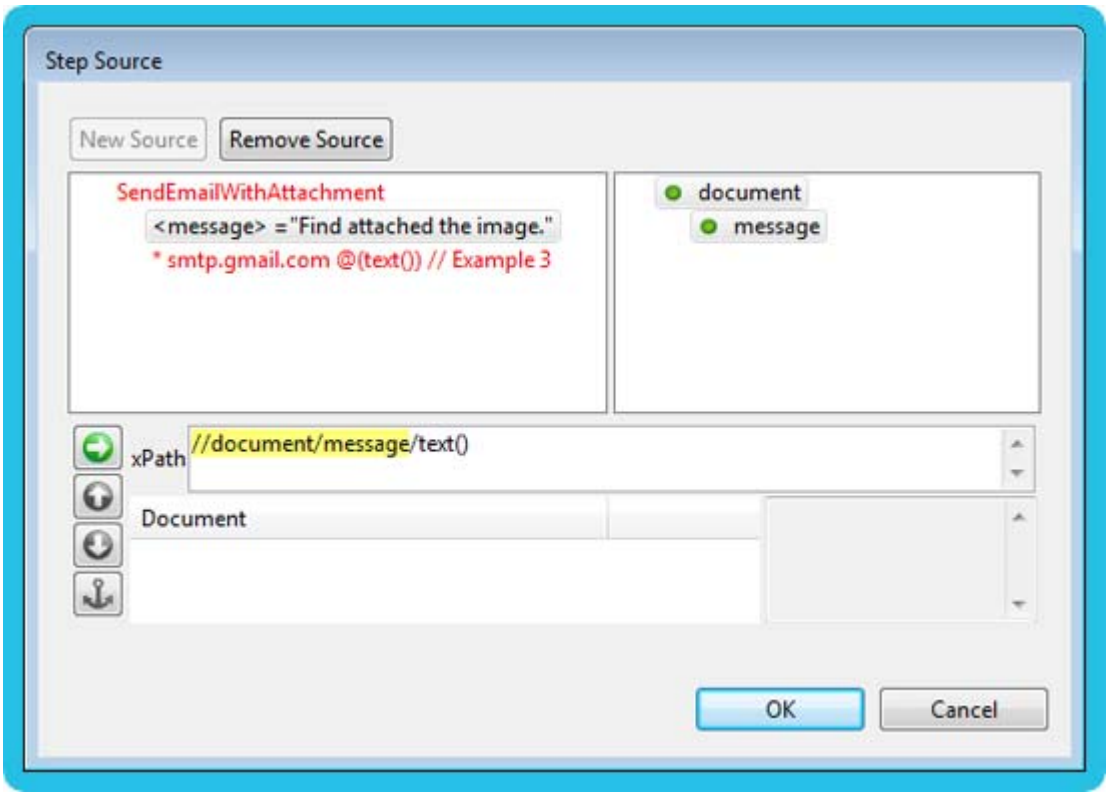


Figure 2 - 179: SMTP send step - Example 3 - Source configuration

Only the text content will be sent as email body.

The *SMTP send* step is created in the **Steps** folder of the sequence, next to the other steps used to implement the sequence behavior described above, the sequence variable and test case. It appears as follows in the **Projects** view:

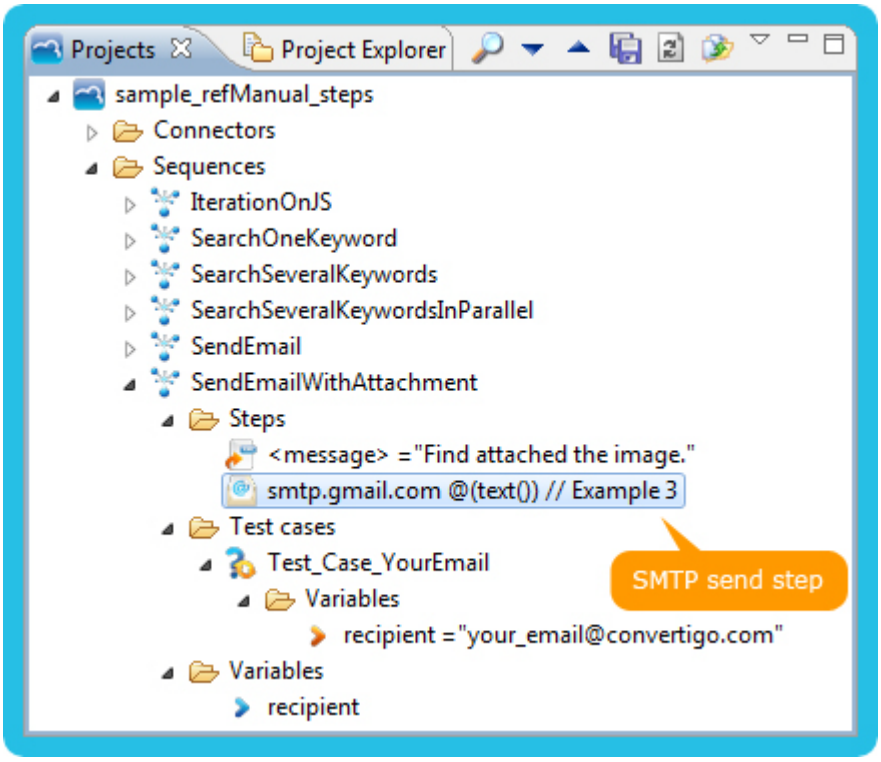


Figure 2 - 180: SMTP send step - Example 3 - Object in Projects view, with sequence and other objects

After configuring your email address as recipient in the test case variable, execute the test case and check your emails to see the sent email. It is a text email of the following form:

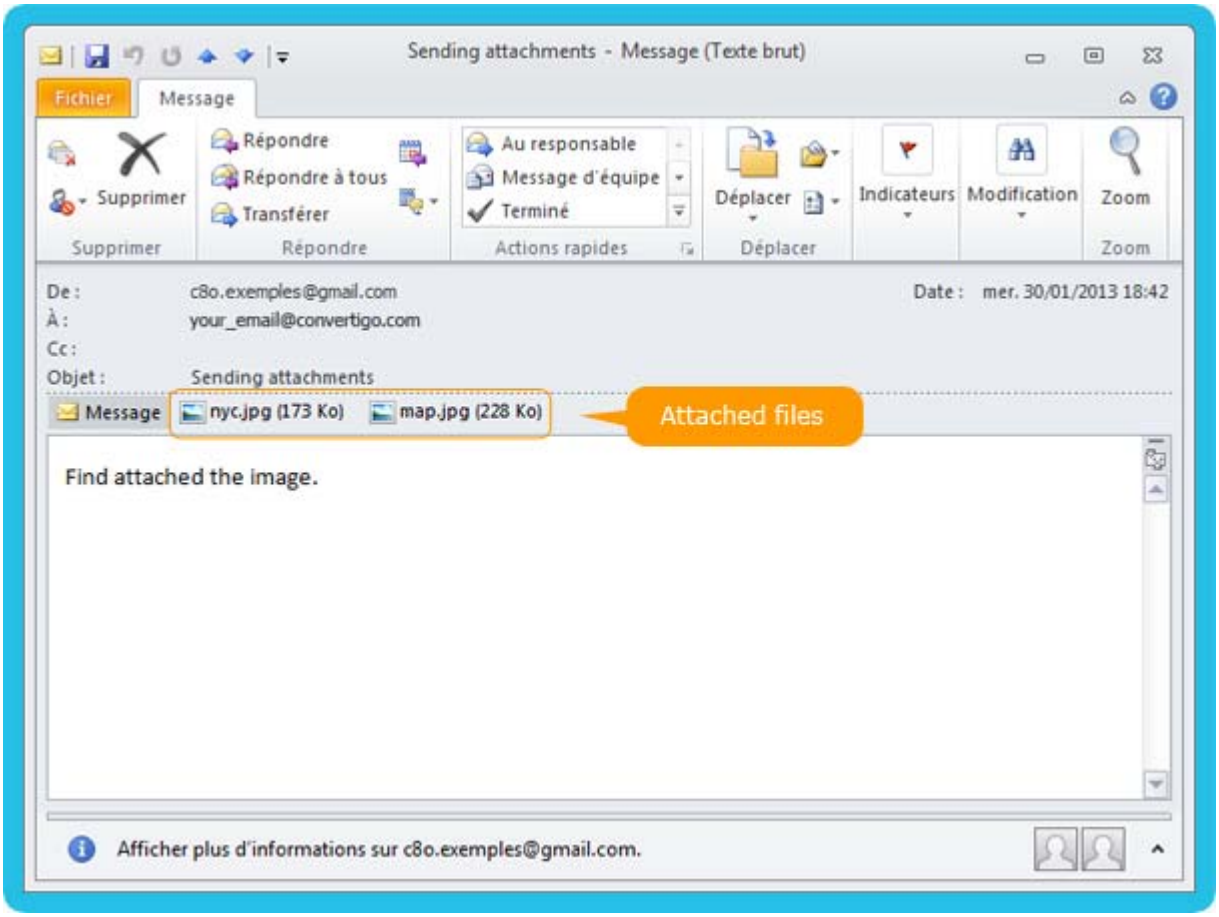


Figure 2 - 181: SMTP send step - Example 3 - Received text email with attached files

The two attachment files are received in the email, using the new names given by the **Attachments** property.

PUSH NOTIFICATIONS 

OBJECT DESCRIPTION

Defines a step able to send notifications to mobile devices.

The *Push Notifications* step is used to make the sequence send a notification to mobile devices using one of the standard `APNS` (Apple), or `GCM` (Android) channels.

The list of devices to which send the notification is configured using the **Device tokens** property. The data to be sent in the notification is configured using the **Notification data** property.

Other properties (which names start by APNS or Google) are those to use in order to configure the technical parameters of the push system.

Note: For more information about using Push notifications in Convertigo, please refer to the article in our technical blog: http://www.convertigo.com/en/how-to/technical-blog/entry/using-convertigo-push-manager.html

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
APNS certificate password	JS expression	standard	Defines the password of the Apple .p12 certificate file. This property is a JavaScript expression that is evaluated during the sequence execution and gives the valid password for the `.p12` certificate file.
APNS client certificate	JS expression	standard	Defines an Apple .p12 certificate file to use for push on iOS devices. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of a `.p12` certificate file. The `.p12` file must be generated by Apple's Certificate portal (https://developer.apple.com/account/overview.action). This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. Note: Do not put the .p12 certificate in the `DisplayObject` folder of the current Convertigo project as it will be packaged within the mobile application during the build process. This would lead to a security breach.
APNS notification type	ApnsNotificationType	standard	Defines the type of push notification for Apple's APNS. This property allows to define which Apple's APNS push type is to be used. It can take one of the following values: • `Message`: sends messages, • `Badge`: notifies application's badge, • `Sound`: plays a sound. Default value is `Message`.

Property	Type	Category	Description
Android push time to live	JS expression	standard	Defines the time to live (in seconds) of the push notification for Android devices. The **Android push time to live** property allows to define the time to live of the message sent for Android GCM. If the message is not delivered within this time, it will be discarded. This property is a JavaScript expression that is evaluated during the sequence execution and gives the time to live in seconds (it should be an integer value). Default value is `3600` seconds, i.e. one hour.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Device tokens	XMLVector	standard	Defines the list of tokens identifying the mobile devices to notify. The **Device tokens** property is of source type, defining a list of nodes from a previous step for current step to work on. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: no device is selected, no notification is sent, and the parent sequence execution continues. Notes: • The mobile device tokens must be known from the Convertigo Server so a list of tokens can be used in this property. Generally, the tokens are generated by the mobile device itself and are sent to Convertigo Server by executing a "storing" sequence. The "storing" sequence should store the tokens in a database (or else), so that they can be retrieved and used in this property. • The mobile device tokens are destination aware: an Android Google Cloud Messaging token will start by `gcm:` and Apple's APNS tokens will start by `apns:`. The tokens stored server-side already contain this piece of information.
Google API key	JS expression	standard	Defines the Google Cloud messaging API key to use for push on Android devices. This property is a JavaScript expression that is evaluated during the sequence execution and gives the Google Cloud messaging API key. This key can be obtained by following the Google Cloud Messaging documentation (http://developer.android.com/google/gcm/gs.html).
Is active	boolean	standard	Defines whether the step is active.

Property	Type	Category	Description
Notification data	XMLVector	standard	Defines the data to be sent in the notification. The **Notification data** property is of source type, defining a list of nodes from a previous step for current step to work on. A **source** is defined as a reference on a step previously existing in the parent sequence, associated with an XPath applied on the step's result DOM. At runtime, the XPath is applied on the step's current execution result XML and extracts a list of XML nodes resulting from this execution. If the XPath doesn't match or if the source is left blank, the step has no data to work on: notification is sent with no data. This content can be a text message, a number to be displayed on a badge, or the name of a sound to play, depending on the **APNS notification type** property value (only for iOS devices).
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

REMOVE CONTEXT 

OBJECT DESCRIPTION

Defines a step which removes a named Convertigo context.

The *Remove context* step removes a Convertigo context that was created by:

- a previous *Call Transaction* or *Call Sequence* step for which a specific context name was defined,
- an `__context` parameter sent to Convertigo while previously calling a transaction/sequence.

The name of the context to remove is specified through the **Context name** property.

Note: The creation or the destruction of a named context is effective in server mode only.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Context name	JS expression	expert	Defines the name of the context to remove. This property is a JavaScript expression that is evaluated at sequence execution. The computed context name is appended to the current session `JSessionID` to define the context ID that is removed.
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

PROCESS EXECUTE 

OBJECT DESCRIPTION

Defines a step able to execute a process.

A *Process execute* step executes the string command specified by the **Command line** property in a separate subprocess.

The subprocess environment parameters and working directory may be defined through the **Environment parameters** and **Execution directory** properties. If left empty, they're inherited from the current process.

Depending on the value of the **Wait for end** property, the step will wait or not until the subprocess has terminated.

Note: Only real programs can be executed thanks to this step. In other words, you cannot execute commands interpreted by a shell (Windows DOS or Linux Bash for example).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Command line	JS expression	standard	Defines the process command line. This property is a JavaScript expression that is evaluated at sequence execution. JavaScript variables and code are supported in this property. The syntax of this command line depends on the operating system where Convertigo is installed.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Encoding	String	standard	Defines the encoding used for the process output. Default value is `UTF-8`. If value is left empty, the default encoding of the Java virtual machine is used.
Environment parameters	XMLVector	standard	Defines the process environment parameters. This property allows to define a list of environment parameters to define for the process execution. For each environment parameter, two columns have to be set: • **Variable**: defines the name of the parameter, • **Value**: defines the value of the parameter. Notes: • A new environment parameter can be added to the list using the blue keyboard icon. The environment parameters defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • If left empty, environment parameters are inherited from the current process, Convertigo.

Property	Type	Category	Description
Execution directory	String	standard	Defines the process execution directory. If left empty, execution directory is inherited from the current process, Convertigo. For a project running in Convertigo Studio, the default directory is the installation directory (were is found the `ConvertigoStudio.exe` file). For a project running in Convertigo Server, the default directory is the application server root folder (tomcat folder for a standard Server installation on Windows).
Is active	boolean	standard	Defines whether the step is active.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Wait for end	boolean	standard	Specifies whether the sequence should wait for the end of the process before continuing with next step. Default value is `true`, so the following step in the parent sequence is executed only after the process execution has returned.

LOG (SEQUENCER) 

OBJECT DESCRIPTION

Produces output data in log file.

This step outputs a message in the Convertigo logger defined in the **Logger** property, for the log level defined in the **Level** property.

The message to output is generated from the JavaScript expression defined in **Expression** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the text to output. This property is a JavaScript expression that is evaluated during the sequence execution and gives the text string to output in log file.
Is active	boolean	standard	Defines whether the step is active.
Level	String	standard	Defines the `log level` on which the log applies. This property defines the minimum level of log for which the message has to be output. The message will be output for any log level superior or equals to this property value. Log levels possible values are the following, by ascending order: • `ERROR`, • `WARN`, • `INFO`, • `DEBUG`, • `TRACE`.
Logger	String	standard	Defines the `logger` on which the log applies. This property defines `Context` logger as default logger. This value can be updated. Possible logger values are the following: • `Engine`: the message will be seen as output by the Convertigo Engine, • `Context`: the message will be seen as output by the running Context, • `Context.User`: the message will be seen as output in the running Context, defined by the User, • `Context.Audit`: the message will be seen as output in the running Context, in a separate Audit logger.

Property	Type	Category	Description
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.

HASH CODE 

OBJECT DESCRIPTION

Generates a hash code from a given file.

The *Hash code* step generates a hash code from a given file using a predefined algorithm, that can be configured using the **Hash algorithm** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Hash algorithm	HashAlgorithm	standard	Defines the algorithm to use for file hashing. This property can take one of the following values: • `MD5`: uses `MD5` algorithm to generate the hash from the file, • `SHA-1`: uses `SHA-1` algorithm to generate the hash from the file.
Is active	boolean	standard	Defines whether the step is active.
Node name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.
Output	boolean	expert	Defines whether the XML generated by this step should be appended to the resulting XML. Set this property to `true` to add the step's resulting XML to the sequence's output XML (default value for steps generating XML). Set this property to `false` to prevent the steps's XML result to appear in the sequence's output XML. Setting this property to `false` does not prevent the step's generated XML from being used as a source by other steps.
Source	JS expression	standard	Defines the path of the file to hash. This property is a JavaScript expression that is evaluated during the sequence execution and gives the path of the file to hash. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.

2.4 SAP

2.4.1 Main objects

SAP CONNECTOR 

OBJECT DESCRIPTION

Establishes connection with SAP Systems using the JCo Connectors.

A *SAP connector* enables Convertigo to connect to any SAP NetWeaver application (such as SAP ERP, etc.), execute BAPIs (Business APIs) and extract data into a proper XML document from them.

The access to the target SAP application is configured using the **Application server host** property, the **System Number** property and the **Client** property. Credentials are defined in the **User name** and the **User password** properties. The language can be configured using the **Language** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Application server host	String	standard	Defines the host name or IP address of the SAP application server. This property defines the DNS name or IP address of the target SAP application server.
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Client	String	standard	Defines the SAP client. This property defines the SAP client, allowing to target a client environment in the SAP server.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.
Language	String	standard	Defines the SAP language. This property defines the SAP language.
System number	String	standard	Defines the SAP system number. This property defines the SAP system number, also known as instance number, allowing to target the SAP instance to be accessed.

Property	Type	Category	Description
User name	String	standard	Defines the SAP user name. This property defines the SAP user.
User password	String	standard	Defines the SAP user password. This property defines the SAP user password. This password must correspond to user name defined in the **User name** property.

SAP TRANSACTION 

OBJECT DESCRIPTION

Defines a SAP transaction.

A *SAP transaction* allows Convertigo to call a SAP BAPI function in the SAP application which is accessed by the parent *SAP connector*.

The BAPI function call takes as parameters the transaction's variables, with their dynamic value at runtime. The BAPI function response is automatically transformed to XML data and returned in transaction's output response.

The schema of the *SAP transaction* are directly extracted from BAPI definition at transaction creation.

Note: Do not confuse Convertigo *SAP transaction* with SAP Transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/ sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/ sequences and run hidden ones if he knows their execution URL.

Property	Type	Category	Description
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
BAPI name	String	standard	Defines the BAPI function to call for this transaction. The **BAPI name** property allows to define the name of the BAPI function to call on the target SAP application. Any BAPI name can be used, in accordance with the target SAP application and available BAPIs (depending on the parent *SAP connector* configuration).

Property	Type	Category	Description
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.

SAP LOGON TRANSACTION 

OBJECT DESCRIPTION

Defines a SAP logon transaction.

A *SAP logon transaction* allows the developer to dynamically change SAP user credentials in the parent *SAP connector* for current execution context.

A *SAP logon transaction* contains three pre-declared variables:

- `jcoUser`: this variable allows to override the **User name** property of the parent *SAP connector* for current context,
- `jcoPassword`: this variable allows to override the **User password** property of the parent *SAP connector* for current context,
- `jcoClient`: this variable allows to override the **Client** property of the parent *SAP connector* for current context.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/ sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/ sequences and run hidden ones if he knows their execution URL.

Property	Type	Category	Description
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.

2.5 SQL

2.5.1 Main objects

SQL CONNECTOR 

OBJECT DESCRIPTION

Establishes connections with an SQL database.

The *SQL connector* enables Convertigo to connect to any database and execute requests.

The access to the target database is configured using the **Driver** property and the **Database URL** property. Credentials may be defined in the **User name** and the **User password** properties, if required.

The *SQL connector* includes a connection pooling process that allows opening a certain number of connections, defined by the **Max. connections** property, that are always ready to execute the requests.

The pooled connections are by default kept opened after the transaction execution. This behavior can be inverted using the **Keep connection alive** property.

The pooled connections can also be automatically tested before execution using the **Test connection** property.

Idle connections can be detected and automatically reset using the **Idle connection search delay** property.

The connection pool can be disabled using the **Enable connection pool** property.

A JDNI mode can also be selected, in the **Driver** property. In this case, the connection to the database is made without using any JDBC driver nor connection pool.

Using JNDI mode, the connection to the database can be configured using a `context.xml` file. This file is located in the `<workspace_folder>/studio` folder in Studio, or has to be created in the `<convertigo_webapp>/META-INF/` folder in Server.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Connection testing query	String	expert	Defines the SQL query to execute to test the connection to the database (optional). The *SQL connector* connection pool needs to check the validity of opened connections. This optional property defines the SQL query to execute on a new opened connection to check the connection's validity. If this property is left blank, Convertigo uses a default SQL query to retrieve the list of the database system tables, depending on the chosen **Driver**: • `sun.jdbc.odbc.JdbcOdbcDriver`: `SELECT 1 AS dbcp_connection_test`. • `com.ibm.as400.access.AS400JDBCDriver`: `SELECT * FROM SYSIBM.SQLSCHEMAS FETCH FIRST 1 ROWS ONLY`. • `com.mysql.jdbc.Driver`: `SELECT * FROM INFORMATION_SCHEMA.TABLES LIMIT 1`. • `net.sourceforge.jtds.jdbc.Driver`: `SELECT TOP 1 * FROM INFORMATION_SCHEMA.TABLES`. • `org.hsqldb.jdbcDriver`: `SELECT TOP 1 * FROM INFORMATION_SCHEMA.SYSTEM_TABLES`. • `com.ibm.db2.jcc.DB2Driver`: `SELECT * FROM SYSCAT.TABLES FETCH FIRST 1 ROWS`. • `oracle.jdbc.driver.OracleDriver`: `SELECT * FROM ALL_TABLES WHERE ROWNUM <= 1`. • `org.mariadb.jdbc.Driver` needs a URL of the form: `SELECT * FROM INFORMATION_SCHEMA.TABLES LIMIT 1`. • `JNDI`: `SELECT 1 AS dbcp_connection_test`.

Property	Type	Category	Description
Database URL	String	standard	Defines the database URL. This property defines the URL needed to connect to the database using the driver class. The database URL syntax depends on the driver class selected in **Driver** property: • `sun.jdbc.odbc.JdbcOdbcDriver` needs a URL of the form: `jdbc:odbc:<DSN(datasource_name)>`. • `com.ibm.as400.access.AS400JDBCDriver` needs a URL of the form: `jdbc:as400://<server_name>:<port(optional)>/<default_schema>;<properties(optional)>`. • `com.mysql.jdbc.Driver` needs a URL of the form: `jdbc:mysql://<server_name>:<port(optional)>/<database_name>`. • `net.sourceforge.jtds.jdbc.Driver` needs a URL of the form: `jdbc:jtds:sqlserver://<server_name>:<port(optional)>/<database_name>`. • `org.hsqldb.jdbcDriver` needs a URL of the form: `jdbc:hsqldb:file:/<file_path>/<database_name>`. • `com.ibm.db2.jcc.DB2Driver` needs a URL of the form: `jdbc:db2://<server_name>:<port(optional)>/<database_name>`. • `oracle.jdbc.driver.OracleDriver` needs a URL of the form: `jdbc:oracle:<drivertype>:<username/password(optional)>@//<host>:<port(optional)>/<service>`, see http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-faq-090281.html#05_03 for Oracle official documentation. • `org.mariadb.jdbc.Driver` needs a URL of the form: `jdbc:mysql://<server_name>:<port(optional)>/<database_name>`. • `JNDI` needs a JNDI resource name: `jdbc/<resource_ref_name>`.

Property	Type	Category	Description
Driver	String	standard	Defines the `JDBC` driver class to use. The following drivers can be selected: • `sun.jdbc.odbc.JdbcOdbcDriver`: JDBC-ODBC bridge for accessing ODBC databases, for example Microsoft Access. • `com.ibm.as400.access.AS400JDBCDriver`: IBM AS400 database. • `com.mysql.jdbc.Driver`: MySQL database. • `net.sourceforge.jtds.jdbc.Driver`: Microsoft SQL Server database. • `org.hsqldb.jdbcDriver`: HSQLDB database (one is included in the Studio for demos and samples). • `com.ibm.db2.jcc.DB2Driver`: IBM DB2 Server database. • `oracle.jdbc.driver.OracleDriver`: ORACLE database. • `org.mariadb.jdbc.Driver`: MariaDB database, community-developed fork of MySQL. • `JNDI`: JNDI mode, not using any JDBC driver nor the connection pooling process. When using JNDI mode, the connection to the database can be configured using a `context.xml` file. This file is located in the `<workspace_folder>/studio` folder in Studio, or has to be created in the `<convertigo_webapp>/META-INF/` folder in Server. Note: You can refer to appendix *SQL drivers and related jar files* in the Operating Guide for more information about driver classes and related jar files.
Enable connection pool	boolean	expert	Defines whether the connection pool is used or not to access target database. The *SQL connector* connection pool allows to automatically pre-connect a pool of connections to the target database. This property allows the programmer to enable or disable the connection pool. If set to `true`, the connection pool is enabled and connections are retrieved from the pool. If set to `false`, the connection pool is disabled and connections are created on-demand. Default value is `true`.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.
Idle connection search delay	long	expert	Defines the number of seconds to wait between searches for idle connections in the connection pool (in seconds). The *SQL connector* connection pool can automatically search for idle connections and remove them from the pool so they are re-started. This property allows the programmer to set a time delay between two searches. Default value is `60` seconds. To disable the idle connection search, set this value to `0`.

Property	Type	Category	Description
Keep connection alive	boolean	expert	Defines whether the connection to the database should be maintained after a transaction execution. Due to the use of connection pool, the *SQL connector* prevents database connections from being closed after each transaction execution or context end. This property set to `false` will force the closure of a connection after a transaction execution or a context end.
Max. connections	int	standard	Defines the maximum number of connections allowed in the connection pool to access the target database. The *SQL connector* connection pool opens in parallel all connections to the target database. This property defines the maximum number of co-existing connections allowed for this connector to connect to the target database. For example, HSQLDB database included in Studio only allows one connection.
Test connection	boolean	expert	Defines whether the connection pool should test or not the connection before providing it. The *SQL connector* connection pool can test the connection before providing it to the requesting transaction for execution. This property allows the programmer to enable or disable this automatic check. Default value is `false`, i.e. the automatic test of each connection is disabled.
User name	String	standard	Defines the user name needed for connecting to the database. This user must exist in the target database and have sufficient authorizations to performs requests executed by transactions.
User password	String	standard	Defines the user password needed for connecting to the database. This password must correspond to user name defined in the **User name** property.

SQL TRANSACTION 

OBJECT DESCRIPTION

Defines an SQL transaction.

An *SQL transaction* allows Convertigo to execute a request on an SQL database, which is accessed by the parent *SQL connector*.

An *SQL transaction* is always associated with an SQL query, or several SQL queries, defined in the **Query** property. The query/queries can be dynamically configured using the transaction's variables (see **Query** property description and *Variable* objects documentation).

In case of single query transaction, the SQL query is executed in auto-commit mode (no matter what is set in **Auto-commit** property). The resultset is output in transaction's XML response in an `sql_output` element. The data organization in the `sql_output` element depends on the **Output mode** property.

In case of multiple queries, the several queries are sequentially executed, using auto-commit mode defined in **Auto-commit** property. Each query's resultset is output in transaction's XML response in `sql_output` elements. The data organization in `sql_output` elements depends on the **Output mode** property and is the same for all queries of the transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/ sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/ sequences and run hidden ones if he knows their execution URL.
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/ sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.

Property	Type	Category	Description
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Auto-commit	int	expert	Defines the commit mode, to be automatic or not. The **Auto-commit** property can take one of the following values: • `enabled, after each query`: auto-commit is done after executing each query from the **Query** property, • `enabled, once at the end`: auto-commit is done after executing all queries from the **Query** property, • `disabled, manual commit`: the developer should program himself the commits to the database thanks to `COMMIT` statement. Default value is `enabled, after each query`, enabling auto-commit after each query execution. Notes: • This property cannot be used when the **Query** property contains one single query. In this case, the auto-commit is always applied. • Not all databases support "grouped transactions" and "rollback". To use this property, be sure that your target database supports these features. Otherwise, the auto-commit is always applied.

Property	Type	Category	Description
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default XML column tag name	String	standard	Defines the default column tag name to output in XML response, depending on the **Output mode** property. This property allows to configure a tag name for each column of result, depending on the **Output mode** property. Default value is `column`.
Default XML row tag name	String	standard	Defines the default row tag name to output in XML response, depending on the **Output mode** property. This property allows to configure a tag name for each line of result, depending on the **Output mode** property. Default value is `row`.
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.
Optional max number of results	String	expert	Defines the maximum number of results returned by the SQL query. Setting this property automatically adds a `LIMIT xx` at the end of the query before its execution on the database. It allows limiting automatically the number of results of every execution of one query. Notes: • Beware that this property should be used only when the target SQL database allows the `LIMIT` keyword in the queries. To know which type of database is accessed by the transaction, refer to the **Driver** property of parent *SQL connector*. • Beware that this property cannot be used when the **Query** property contains several queries.

Property	Type	Category	Description
Output mode	int	expert	Defines how the resulting XML is generated from each SQL query result. The **Output mode** property allows to change the structure of generated XML for a same query result. It can be set to one of the following values: • `RAW`: generates a `row` element for each result line, selected columns are added as attributes named after the column names. Note that as the columns are XML attributes, they are sorted by alphabetical order under the `row` element. • `AUTO`: generates a mix of complex elements and elements with attributes (interesting for multi-table requests). • `ELEMENT`: generates XML elements named after table names, selected columns are added as child XML elements named after column name. In case of multi-table requests, one row is a complex element which depth depends on the number of selected tables. Note that as the columns are XML elements, they are sorted in the order requested in the query and as the resultset has returned data. • `ELEMENT_WITH_ATTRIBUTES`: generates a structure similar to the `ELEMENT` output mode but with `row` and `column` tag names. Each element (row or column) contains a `name` attribute with the actual name of the selected table or column (useful when table or column names contain symbols not allowed in XML tag names, or start by a number, etc.). • `FLAT_ELEMENT`: generates a structure similar to the `RAW` output mode but with elements instead of attributes. Each result line generates a `row` element, selected columns are added as child XML elements named after the column names. Note that as the columns are XML elements, they are sorted in the order requested in the query and as the resultset has returned data.

Property	Type	Category	Description
Query	String	standard	Defines the SQL query/queries to execute on the target database. The **Query** property allows to define an SQL query or several SQL queries to be executed on the target database. Any SQL query must be written in accordance with the target database tables and available functions and keywords (depending on the parent *SQL connector* configuration). The SQL query/queries can be parameterized with the transaction's variables, to be dynamically configured at runtime with variable values. To use a transaction variable in an SQL query, use one of the following syntaxes in the query: • `{variable_name}`: the simple-brace notation - variable name surrounded by curly braces - protects the SQL query from SQL injections, i.e. only the first value represented by the variable is used, discarding any further content (SQL injection). It works so that the variable only contains a value. This notation can only be used inside the `WHERE` clause. If you need to use a variable anywhere else inside the SQL query, you should use the double-brace notation. • `{{variable_name}}`: the double-brace notation - variable name surrounded by double curly braces - does not protect the SQL query from SQL injections, i.e. the variable content can contain any content. If the variable value contains a piece of SQL query, it will not prevent the SQL query execution. This may be useful when a whole `WHERE` clause is computed outside the transaction (at sequence level for example) and passed as a variable to the SQL transaction. It should also be used when variables need to be used in the query outside of the `WHERE` clause. In the case of multiple SQL queries, they must be separated by semicolons ';'. Notes: • In the case of single query, you cannot use the **Auto-commit** property. • In the case of multiple SQL queries, you cannot use the **Optional max number of results** property in this case.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.
XML grouping	boolean	expert	Defines whether the resulting XML should be grouped by elements. Default value is `false`. Setting it to `true` enables the grouping of XML elements in the transaction's XML response. The behavior can be different depending on the **Output mode** property value.

2.6 CICS

2.6.1 Main objects

CICS CONNECTOR 

OBJECT DESCRIPTION

Establishes connections with a CICS application.

A CICS Transaction Gateway can host several servers running different applications. In a CICS application, the display and business logics are usually managed in distinct programs.

Programs exchange input and output data through a memory pool called COMMAREA (COMMON AREA). The COMMAREA is usually mapped by a definition of COBOL data included in communicating programs. This definition can be stored in two ways:

- built in the source code of the CICS program,
- stored in a separate file, called `copybook`, that is copied when compiling.

From a Convertigo point of view, a *CICS connector* represents a gateway server.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.
Mainframe	String	standard	Defines the mainframe name (or its IP address).
Port	int	standard	Defines the server port number.
Server	String	standard	Defines the server name (or its IP address).
User id	String	standard	Defines the user identification for connecting to the CICS server.
User password	String	standard	Defines the user password for connecting to the CICS server.

CICS TRANSACTION 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

2.7 Web services

2.7.1 Main objects

HTTP CONNECTOR 

OBJECT DESCRIPTION

Establishes HTTP connections.

HTTP connections are needed by Convertigo for connecting to and communicating with required HTTP servers. The *HTTP connector* is used to consume web services such as REST, SOAP or JSON, as well as getting data using HTTP protocol.

To call a REST or SOAP web service, *XML HTTP Transactions* have to be used as these web services are XML-based. To consume a JSON web service, *JSON HTTP Transactions* have to be used as it performs the conversion from the JSON data to the transaction output XML. To retrieve any other data in HTTP protocol (non XML-based REST web service, image or file getting, etc.), standard *HTTP Transaction* has to be used as its response is text-based.

Note: *HTTP connector* supports OAuth authentication. To enable OAuth, you simply need to provide four variables to any kind of *HTTP transaction*: `__header_oAuthKey`, `__header_oAuthSecret`, `__header_oAuthToken` and `__header_oAuthTokenSecret`. For more information about OAuth in *HTTP connector*, refer to the following article in our Technical Blog: *http://www.convertigo.com/en/how-to/technical-blog/entry/using-oauth-with-convertigo-http-connector.html*

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Authentication type	Authentication Mode	expert	Defines the authentication type between basic and NTLM authentications. This property allows to define which type of authentication has to be used for the HTTP request. Default value is `Basic`. If **Basic/NTLM authentication user** and **Basic/NTLM authentication password** properties are not filled, no authentication is performed. Notes: • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.
Basic/NTLM authentication password	String	expert	Defines the user's password for basic or NTLM authentication. This property value is used as user password for basic or NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.

Property	Type	Category	Description
Basic/NTLM authentication user	String	expert	Defines the user name for basic or NTLM authentication. This property value is used as user name for basic or NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default URL charset encoding	String	expert	Defines the default charset encoding to use for the transactions variable values sent as parameters in HTTP requests. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is `UTF-8`.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.

Property	Type	Category	Description
HTTP headers forwarding policy	XMLVector	expert	Defines the HTTP headers to forward and the policy to use to forward them. This property allows forwarding HTTP headers from the client browser to the target application. This property allows to define a list of HTTP headers to forward and, for each header, the forwarding policy to use. For each header, two columns have to be set: • **Header name**: defines the name of the header to forward, • **Forwarding policy**: defines how to replace the header value when forwarding it. This second property can take the following values: • `Merge`: If the forwarded header exists, its value is merged with existing one. If the forwarded header doesn't exist, it is added. • `Ignore`: If the forwarded header exists, its value is not replaced, it is ignored. If the forwarded header doesn't exist, it is added. • `Replace`: Replaces all headers without any condition by forwarded values. Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Is HTTPS	boolean	standard	Defines whether the connection is secured (HTTPS). If set to `true`, the connection is SSL-based. Make sure the target SSL port (usually 443) is correctly set.
NTLM authentication domain	String	expert	Defines the NTLM authentication domain in case of NTLM authentication. This property value is used as user domain for NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, this property does not need to be filled.
Port	int	standard	Defines the server port.
Root path	String	standard	Defines the root path. This is the first URI requested by the HTTP connector. Any other URI in the project is relative to this URI.
Server	String	standard	Defines the server name (or its IP address). This property defines the DNS name or IP address of the target application server.
Trust all certificates	boolean	standard	Defines whether trusted certificates must be checked. In SSL mode, the server sends existing certificates to Convertigo. In most cases, set this setting to `true` to automatically trust all server certificates. If set to `false` , target server certificates must be installed in Convertigo.

EXAMPLES

Let's consider a company named Global Company delivering a Web service for its Human

Resources department.This is a sample Web service accessible in REST and SOAP modes (developed thanks to Convertigo).

In this example, we want to define an *HTTP connector*, in a new project named `sample_refManual_http`, for connecting to this sample Web service in REST.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTTP connector objects examples*** *in the* ***New Project*** *wizard.*

An *HTTP connector* object is created with the following properties:

```
HTTP connector [
  is HTTPs=true
  server=demo.convertigo.net
  port=443
  root path=/cems/projects/globalCompany_HR_WS
  trust all certificates=true
  carioca authentication=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

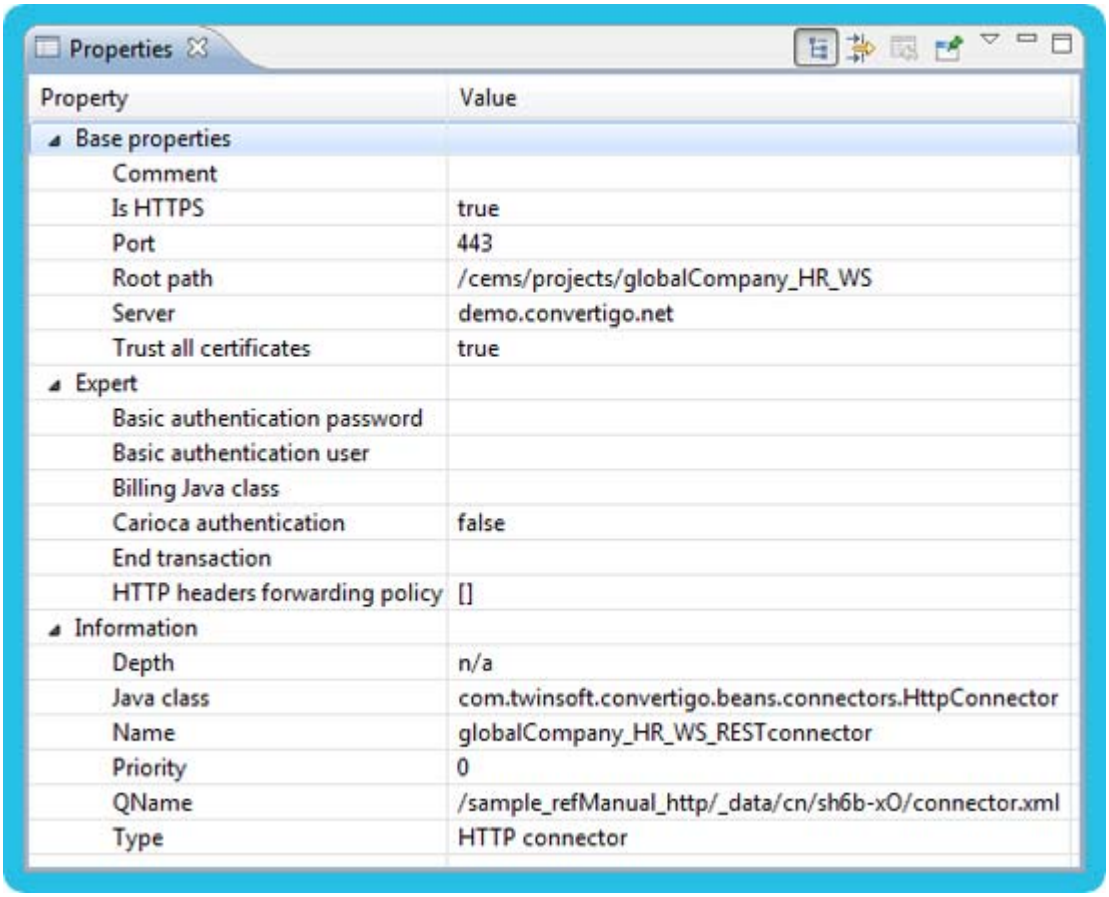


Figure 2 - 182: HTTP connector - Configuration example

The connector is created in the **Connectors** folder of the project, including various other

objects, such as transactions. It appears as follows in the **Projects** view:

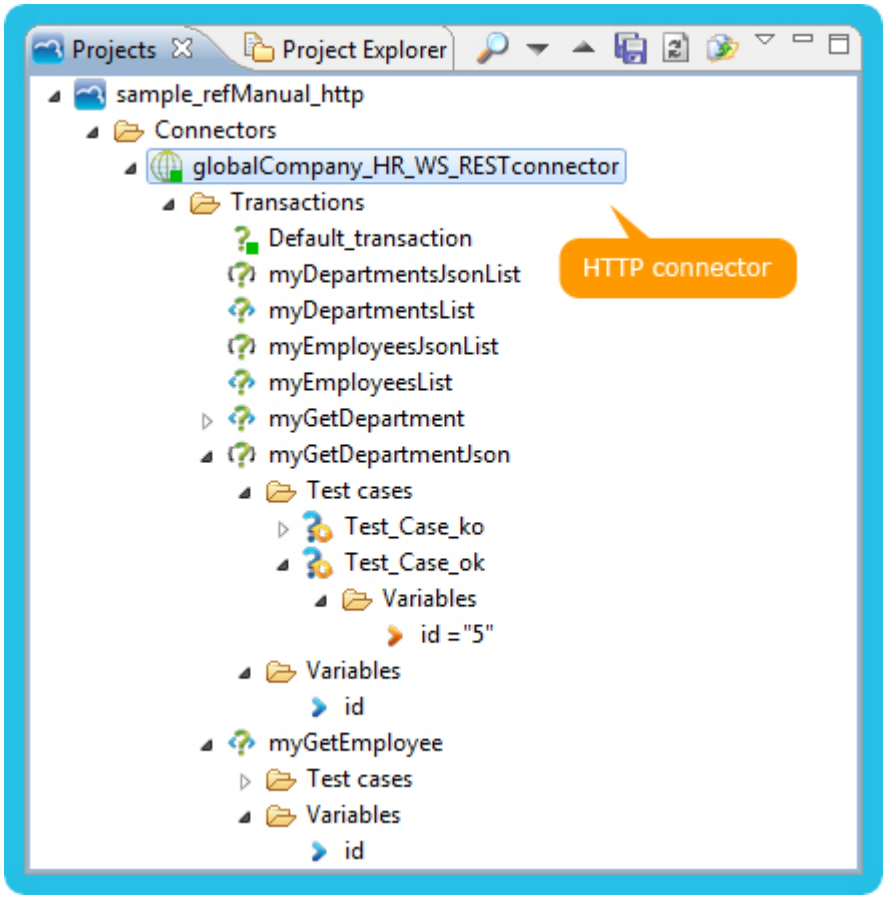


Figure 2 - 183: HTTP connector - Object in Projects view

The **HTTP connector** editor displaying HTTP data and generated XML appears as follows:

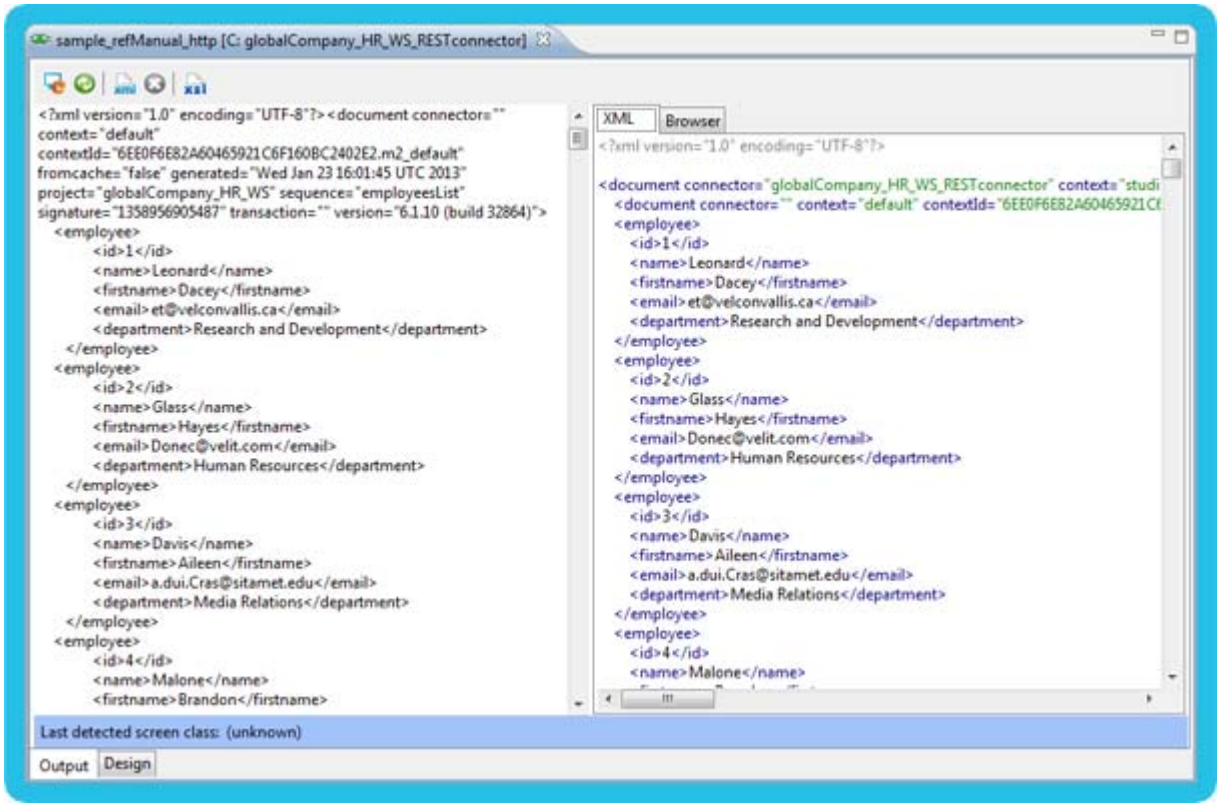


Figure 2 - 184: HTTP connector - Connector editor in Studio

PROXY HTTP CONNECTOR 

OBJECT DESCRIPTION

Defines a proxy-based HTTP connector.

The *Proxy HTTP connector* is similar to an *HTTP connector* and in addition simulates a proxy.

Note: This connector is used for very specific applications and is not intended to be used by Convertigo standard users.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Authentication type	Authentication Mode	expert	Defines the authentication type between basic and NTLM authentications. This property allows to define which type of authentication has to be used for the HTTP request. Default value is `Basic`. If **Basic/NTLM authentication user** and **Basic/NTLM authentication password** properties are not filled, no authentication is performed. Notes: • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.
Basic/NTLM authentication password	String	expert	Defines the user's password for basic or NTLM authentication. This property value is used as user password for basic or NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.
Basic/NTLM authentication user	String	expert	Defines the user name for basic or NTLM authentication. This property value is used as user name for basic or NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.

Property	Type	Category	Description
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default URL charset encoding	String	expert	Defines the default charset encoding to use for the transactions variable values sent as parameters in HTTP requests. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is `UTF-8`.
Dynamic content files	String	standard	Defines files exposing dynamic content (should not be cached). This property allows to specify the files that should not be cached by the *Proxy HTTP connector*.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.

Property	Type	Category	Description
HTTP headers forwarding policy	XMLVector	expert	Defines the HTTP headers to forward and the policy to use to forward them. This property allows forwarding HTTP headers from the client browser to the target application. This property allows to define a list of HTTP headers to forward and, for each header, the forwarding policy to use. For each header, two columns have to be set: • **Header name**: defines the name of the header to forward, • **Forwarding policy**: defines how to replace the header value when forwarding it. This second property can take the following values: • `Merge`: If the forwarded header exists, its value is merged with existing one. If the forwarded header doesn't exist, it is added. • `Ignore`: If the forwarded header exists, its value is not replaced, it is ignored. If the forwarded header doesn't exist, it is added. • `Replace`: Replaces all headers without any condition by forwarded values. Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Is HTTPS	boolean	standard	Defines whether the connection is secured (HTTPS). If set to `true`, the connection is SSL-based. Make sure the target SSL port (usually 443) is correctly set.
NTLM authentication domain	String	expert	Defines the NTLM authentication domain in case of NTLM authentication. This property value is used as user domain for NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, this property does not need to be filled.
Port	int	standard	Defines the server port.
Removable headers	XMLVector	standard	Defines HTTP headers that should be removed from client request. This property allows to specify a list of HTTP headers that should not be forwarded by the *Proxy HTTP connector* to the target application. Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Replacement strings	XMLVector	standard	Defines strings to be replaced on the fly. As simulating a proxy, the *Proxy HTTP connector* replaces "on-the-fly" strings in the HTTP flow. This property allows to specify the strings to replace and their replacements. Note: A new string can be added to the list using the blue keyboard icon. The strings defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.

Property	Type	Category	Description
Root path	String	standard	Defines the root path. This is the first URI requested by the HTTP connector. Any other URI in the project is relative to this URI.
Server	String	standard	Defines the server name (or its IP address). This property defines the DNS name or IP address of the target application server.
Trust all certificates	boolean	standard	Defines whether trusted certificates must be checked. In SSL mode, the server sends existing certificates to Convertigo. In most cases, set this setting to `true` to automatically trust all server certificates. If set to `false` , target server certificates must be installed in Convertigo.

HTTP TRANSACTION 

OBJECT DESCRIPTION

Defines an HTTP transaction.

An *HTTP transaction* is a Convertigo transaction based on HTTP requests. It allows to perform an HTTP request and get the response back.

Unlike *XML HTTP transaction* or *JSON HTTP transaction*, simple *HTTP transaction* receives text-based HTTP responses. It is used to retrieve any data in HTTP protocol (non XML-based REST web service, image or file getting, etc.).

Note: *HTTP connector* supports OAuth authentication. To enable OAuth, you simply need to provide four variables to any kind of *HTTP transaction*: `__header_oAuthKey`, `__header_oAuthSecret`, `__header_oAuthToken` and `__header_oAuthTokenSecret`. For more information about OAuth in *HTTP connector*, refer to the following article in our Technical Blog: *http://www.convertigo.com/en/how-to/technical-blog/entry/using-oauth-with-convertigo-http-connector.html*

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/sequences and run hidden ones if he knows their execution URL.

Property	Type	Category	Description
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
HTTP data encoding	int	standard	Defines the data encoding to use for encoding the data received through the HTTP connector. HTTP data retrieved through the *HTTP connector* can be one of of the following types: • `string` type: string data are not encoded, • `binary` type: binary data are encoded in `Base64`.
HTTP headers	XMLVector	expert	Defines HTTP headers to be sent. This property allows to define the request Header Fields to be sent with the request to the target web service method. For each header, two columns have to be set: • **Variable**: HTTP header name (ex: `Content-Type`). • **Value**: HTTP header value (ex: `application/x-www-from-urlencoded`). Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
HTTP info	boolean	expert	Defines whether to include HTTP information in output XML. HTTP information can be added to the transaction's output XML, such as the request URL, HTTP status code and the HTTP request and response headers. You can also have the raw HTTP data in case of error. The **HTTP info** property allows to define whether these information have to be inserted in the transaction's output XML (value set to `true`) or not (value set to `false`). Default value is `false`.
HTTP info tagname	String	expert	Defines the tagname of the element containing the HTTP info in output XML. When the **HTTP info** property defines to insert the HTTP information in the transaction's output XML, the **HTTP info tagname** property allows the programmer to define the tagname of the element containing these information. Default value is `HttpInfo`.
HTTP verb	int	standard	Defines the HTTP verb to use for this HTTP request: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `TRACE`, `OPTIONS` or `CONNECT`. For more information about HTTP verbs, you can visit the following RFC page: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*.
Handles cookies	boolean	expert	Defines whether cookies must be handled. If set to `true` (default value), the transaction maintains cookies in Convertigo's context. Default value should not be changed unless you specifically want the transaction to ignore cookies while browsing.

Property	Type	Category	Description
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.
Request template	String	expert	Defines the request body template file path. HTTP request sent by the transaction can contain data in its body. This data is based on a user-defined template file, which can be: • an XML file describing the content of the HTTP request body, possibly including transaction input variables in the data structure, • an XSL file used to transform the variable-based transaction input XML to generate the content of the HTTP request body. This property allows to define the path of the template file, it is either: • a local file, by default relative to the project's directory, or to the project's current subfolder, • a local file relative to the Convertigo webapp common `templates` directory, • an absolute path. If the template file is an XML file, it can contain transaction variables identified with a specific syntax in the XML and dynamically replaced at runtime with received variable values. The syntax to use in the XML template file to refer to a transaction variable is the following: • `$(<variableHttpName>)`: this simple notation starts with a $ character and then includes between brackets the **HTTP name** of the variable. Beware that the **HTTP name** of the variable can be different from the variable name (see *Variable* objects documentation). • `$(<variableHttpName>)concat`: this notation is very similar to the preceding, excepted that the last bracket is followed by the `concat` keyword. It starts by a $ character and includes between brackets the **HTTP name** of the variable, that should be in this case a *Multi-valued variable*. The `concat` keyword implies that all values received in the *Multi-valued variable* must be concatenated before replacing this notation by this computed value in the template XML. • `$(<variableHttpName>)`: this notation is identical to the first notation, but the behavior is different for a *Multi-valued variable*. The tag surrounding this notation in the template XML is duplicated for each value in the *Multi-valued variable*.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
Sub path	String	standard	Defines the end of the path for the HTTP connection. This property allows to define the sub path, relative to the connector root path, to the target web service URI. For example, if the target is: `http://server/MyApp/targetpage.jsp`, the connector server would be: `server`, the connector root path: `/MyApp` and the transaction sub path: `/targetpage.jsp`.
URL charset encoding	String	expert	Defines the charset encoding to use for the variable values sent as parameters in HTTP request. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is blank. If blank, the parent connector's **Default URL charset encoding** property value is used.
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.

XML HTTP TRANSACTION 

OBJECT DESCRIPTION

Defines an XML-based HTTP transaction.

An *XML HTTP transaction* is an HTTP transaction, allowing to perform an HTTP request and get the response back, for which responses are XML-based. It is used to call a REST or SOAP web service which responses are XML-based.

Note: *HTTP connector* supports OAuth authentication. To enable OAuth, you simply need to provide four variables to any kind of *HTTP transaction*: `__header_oAuthKey`, `__header_oAuthSecret`, `__header_oAuthToken` and `__header_oAuthTokenSecret`. For more information about OAuth in *HTTP connector*, refer to the following article in our Technical Blog: *http://www.convertigo.com/en/how-to/technical-blog/entry/using-oauth-with-convertigo-http-connector.html*

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/sequences and run hidden ones if he knows their execution URL.

Property	Type	Category	Description
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.
Assigned element QName	XmlQName	expert	The schema element qualified name of the targeted web service to be referenced in the Convertigo SOAP response element. |
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).

Property	Type	Category	Description
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
HTTP headers	XMLVector	expert	Defines HTTP headers to be sent. This property allows to define the request Header Fields to be sent with the request to the target web service method. For each header, two columns have to be set: • **Variable**: HTTP header name (ex: `Content-Type`). • **Value**: HTTP header value (ex: `application/x-www-from-urlencoded`). Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
HTTP info	boolean	expert	Defines whether to include HTTP information in output XML. HTTP information can be added to the transaction's output XML, such as the request URL, HTTP status code and the HTTP request and response headers. You can also have the raw HTTP data in case of error. The **HTTP info** property allows to define whether these information have to be inserted in the transaction's output XML (value set to `true`) or not (value set to `false`). Default value is `false`.
HTTP info tagname	String	expert	Defines the tagname of the element containing the HTTP info in output XML. When the **HTTP info** property defines to insert the HTTP information in the transaction's output XML, the **HTTP info tagname** property allows the programmer to define the tagname of the element containing these information. Default value is `HttpInfo`.
HTTP verb	int	standard	Defines the HTTP verb to use for this HTTP request: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `TRACE`, `OPTIONS` or `CONNECT`. For more information about HTTP verbs, you can visit the following RFC page: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*.
Handles cookies	boolean	expert	Defines whether cookies must be handled. If set to `true` (default value), the transaction maintains cookies in Convertigo's context. Default value should not be changed unless you specifically want the transaction to ignore cookies while browsing.
Ignore SOAP envelope	boolean	expert	Defines whether the response SOAP envelope elements should be removed from the transaction XML response. If set to `true` (default value), the SOAP envelope elements of the HTTP response are not kept in the transaction output XML. If set to `false`, the SOAP envelope XML elements are kept in the transaction output XML.

Property	Type	Category	Description
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.
Request template	String	expert	Defines the request body template file path. HTTP request sent by the transaction can contain data in its body. This data is based on a user-defined template file, which can be: • an XML file describing the content of the HTTP request body, possibly including transaction input variables in the data structure, • an XSL file used to transform the variable-based transaction input XML to generate the content of the HTTP request body. This property allows to define the path of the template file, it is either: • a local file, by default relative to the project's directory, or to the project's current subfolder, • a local file relative to the Convertigo webapp common `templates` directory, • an absolute path. If the template file is an XML file, it can contain transaction variables identified with a specific syntax in the XML and dynamically replaced at runtime with received variable values. The syntax to use in the XML template file to refer to a transaction variable is the following: • `$(<variableHttpName>)`: this simple notation starts with a $ character and then includes between brackets the **HTTP name** of the variable. Beware that the **HTTP name** of the variable can be different from the variable name (see *Variable* objects documentation). • `$(<variableHttpName>)concat`: this notation is very similar to the preceding, excepted that the last bracket is followed by the `concat` keyword. It starts by a $ character and includes between brackets the **HTTP name** of the variable, that should be in this case a *Multi-valued variable*. The `concat` keyword implies that all values received in the *Multi-valued variable* must be concatenated before replacing this notation by this computed value in the template XML. • `$(<variableHttpName>)`: this notation is identical to the first notation, but the behavior is different for a *Multi-valued variable*. The tag surrounding this notation in the template XML is duplicated for each value in the *Multi-valued variable*.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Schema of response's RPC call	String	expert	Defines the schema of the RPC call response. Only used in case of auto-generated transaction when importing a WSDL. When importing a remote web service thanks to its WSDL to create a project, each transaction allows invoking a method of the target web service. Each method already has a schema defined in the web service. This property allows to reference the name of this schema type to reuse it in transaction output XML schema. We advise not to manually update this property has it is mainly used for automatically created transactions.
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
Sub path	String	standard	Defines the end of the path for the HTTP connection. This property allows to define the sub path, relative to the connector root path, to the target web service URI. For example, if the target is: `http://server/MyApp/targetpage.jsp`, the connector server would be: `server`, the connector root path: `/MyApp` and the transaction sub path: `/targetpage.jsp`.
URL charset encoding	String	expert	Defines the charset encoding to use for the variable values sent as parameters in HTTP request. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is blank. If blank, the parent connector's **Default URL charset encoding** property value is used.

Property	Type	Category	Description
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.
XML response encoding	String	expert	Defines the encoding of the XML returned by the target server. Default value is `ISO-8859-1`. Depending on the target web service, the value has to be updated.

EXAMPLES

Let's consider a company named Global Company delivering a Web service for its Human Resources department.This is a sample Web service accessible in REST and SOAP modes (developed thanks to Convertigo).

We defined an *HTTP connector*, in a project named `sample_refManual_http`, for connecting to this sample Web service in REST (for more information, see *HTTP connector* documentation and examples). In this example, we are focusing on the transactions created to call the methods of this Web service in XML.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTTP connector objects examples*** *in the* ***New Project*** *wizard.*

Several *XML HTTP transaction* objects are created to call the different methods of the target Web service in XML. For example, one transaction, defined to call the `getDepartment` method of the Web service, is created with the following properties:

```
XML HTTP transaction [
  accessibility=Public
  HTTP verb=GET
  sub path=/.xml?__sequence=getDepartment
  handles cookies=true
  request template=
  XML response encoding=ISO-8859-1
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

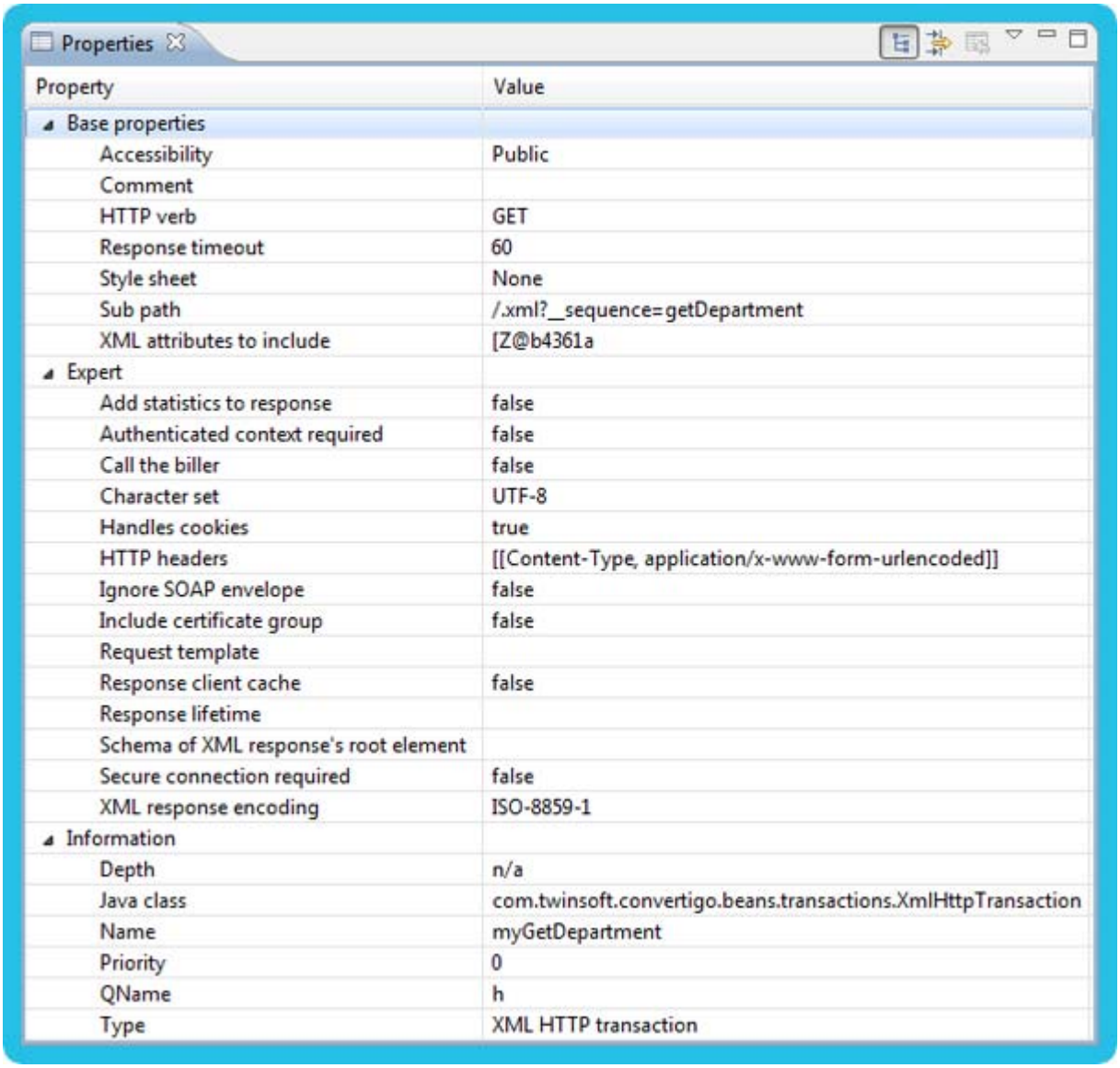


Figure 2 - 185: XML HTTP transaction - Configuration example

The transaction is created in the **Transactions** folder of the connector, including various other objects, such as variables and test cases. It appears as follows in the **Projects** view:

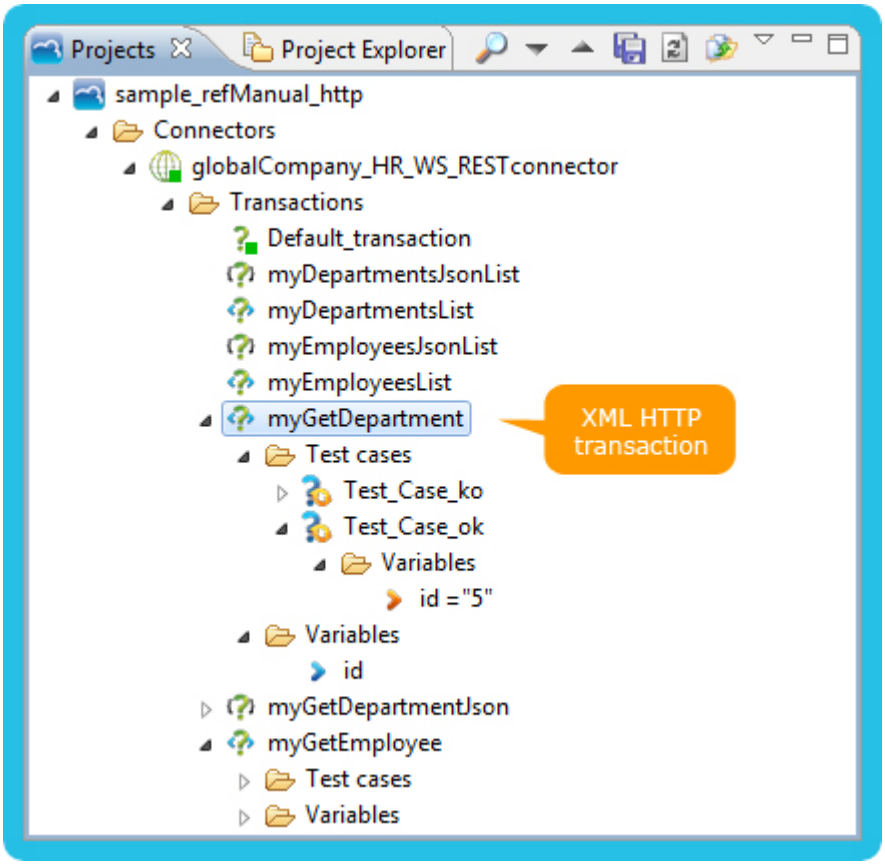


Figure 2 - 186: XML HTTP transaction - Object in Projects view

Executing one of the test cases of the `myGetDepartment` transaction (thanks to the **Run** entry in the test case's contextual menu or by pressing the F5 key) invokes the Web service method defined by the URL in **Sub path** property and gets the response from it. The connector editor shows:

- on the left part, the HTTP response received from the target Web service method,
- on the right part, in the **XML** tab, the transaction output XML which includes the Web service response:

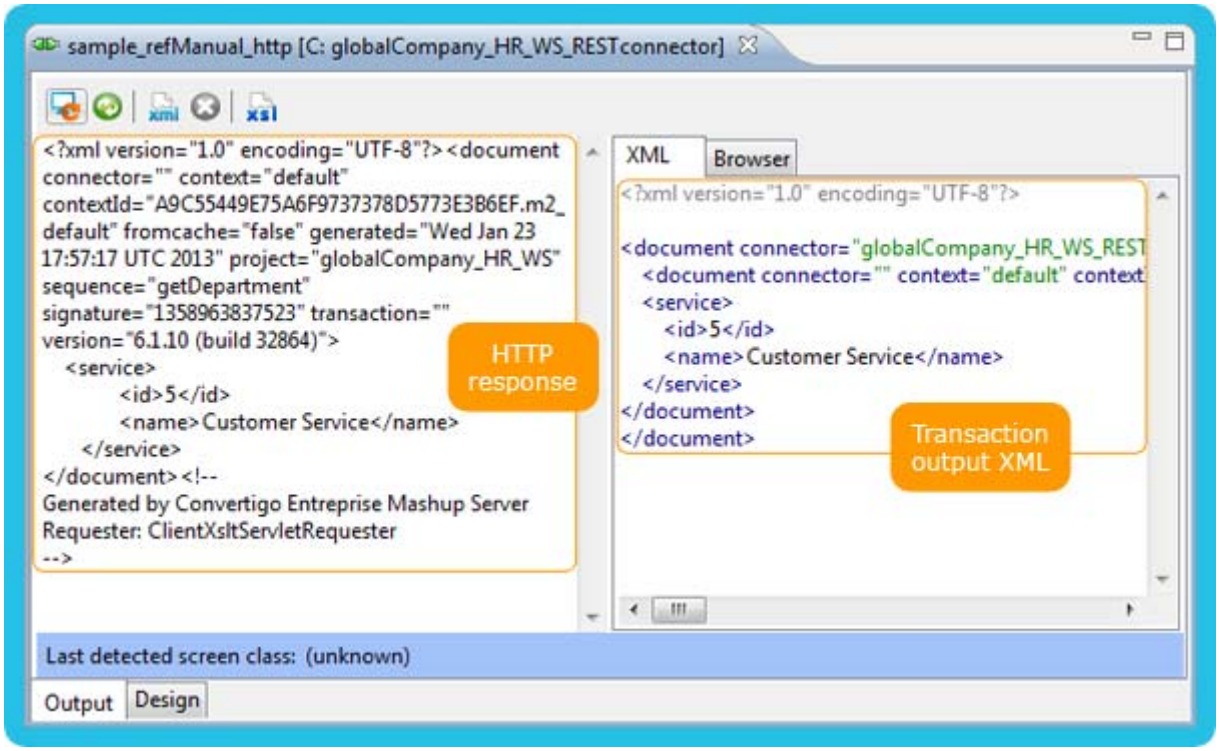


Figure 2 - 187: XML HTTP transaction - Responses in connector editor

JSON HTTP TRANSACTION 

OBJECT DESCRIPTION

Defines a JSON-based HTTP transaction.

A *JSON HTTP transaction* is an HTTP transaction, allowing to perform an HTTP request and get the response back, for which responses are JSON-based. It is used to consume a JSON web service.

Note: *HTTP connector* supports OAuth authentication. To enable OAuth, you simply need to provide four variables to any kind of *HTTP transaction*: `__header_oAuthKey`, `__header_oAuthSecret`, `__header_oAuthToken` and `__header_oAuthTokenSecret`. For more information about OAuth in *HTTP connector*, refer to the following article in our Technical Blog: *http://www.convertigo.com/en/how-to/technical-blog/entry/using-oauth-with-convertigo-http-connector.html*

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/sequences and run hidden ones if he knows their execution URL.

Property	Type	Category	Description
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
HTTP headers	XMLVector	expert	Defines HTTP headers to be sent. This property allows to define the request Header Fields to be sent with the request to the target web service method. For each header, two columns have to be set: • **Variable**: HTTP header name (ex: `Content-Type`). • **Value**: HTTP header value (ex: `application/x-www-from-urlencoded`). Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
HTTP info	boolean	expert	Defines whether to include HTTP information in output XML. HTTP information can be added to the transaction's output XML, such as the request URL, HTTP status code and the HTTP request and response headers. You can also have the raw HTTP data in case of error. The **HTTP info** property allows to define whether these information have to be inserted in the transaction's output XML (value set to `true`) or not (value set to `false`). Default value is `false`.
HTTP info tagname	String	expert	Defines the tagname of the element containing the HTTP info in output XML. When the **HTTP info** property defines to insert the HTTP information in the transaction's output XML, the **HTTP info tagname** property allows the programmer to define the tagname of the element containing these information. Default value is `HttpInfo`.
HTTP verb	int	standard	Defines the HTTP verb to use for this HTTP request: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `TRACE`, `OPTIONS` or `CONNECT`. For more information about HTTP verbs, you can visit the following RFC page: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*.
Handles cookies	boolean	expert	Defines whether cookies must be handled. If set to `true` (default value), the transaction maintains cookies in Convertigo's context. Default value should not be changed unless you specifically want the transaction to ignore cookies while browsing.
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.

Property	Type	Category	Description
Include data type in XML response nodes	boolean	expert	Defines if the JSON data type should be included in converted XML nodes. Default value is `true`: data types are included in the XML nodes of the response as `type` attributes. Changing this property to `false` has as effect to not include the data types in XML nodes: the transaction response XML contains only data.
JSON array translation policy	int	expert	Defines how JSON arrays should be translated to XML. This property allows the Convertigo developer to choose how he wants the JSON arrays to be translated to XML. The need can be different depending on the source JSON web service. Possible options are: • `hierarchical`: full expanded mode, with sub nodes for array items, • `compact`: more compressed format with one node per array item, without encompassing array node. Default value is `hierarchical`.
JSON response encoding	String	expert	Defines the encoding of the JSON returned by the target server. Default value is `UTF-8`. Depending on the target web service, the value has to be updated.

Property	Type	Category	Description
Request template	String	expert	Defines the request body template file path. HTTP request sent by the transaction can contain data in its body. This data is based on a user-defined template file, which can be: • an XML file describing the content of the HTTP request body, possibly including transaction input variables in the data structure, • an XSL file used to transform the variable-based transaction input XML to generate the content of the HTTP request body. This property allows to define the path of the template file, it is either: • a local file, by default relative to the project's directory, or to the project's current subfolder, • a local file relative to the Convertigo webapp common `templates` directory, • an absolute path. If the template file is an XML file, it can contain transaction variables identified with a specific syntax in the XML and dynamically replaced at runtime with received variable values. The syntax to use in the XML template file to refer to a transaction variable is the following: • `$(<variableHttpName>)`: this simple notation starts with a $ character and then includes between brackets the **HTTP name** of the variable. Beware that the **HTTP name** of the variable can be different from the variable name (see *Variable* objects documentation). • `$(<variableHttpName>)concat`: this notation is very similar to the preceding, excepted that the last bracket is followed by the `concat` keyword. It starts by a $ character and includes between brackets the **HTTP name** of the variable, that should be in this case a *Multi-valued variable*. The `concat` keyword implies that all values received in the *Multi-valued variable* must be concatenated before replacing this notation by this computed value in the template XML. • `$(<variableHttpName>)`: this notation is identical to the first notation, but the behavior is different for a *Multi-valued variable*. The tag surrounding this notation in the template XML is duplicated for each value in the *Multi-valued variable*.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
Sub path	String	standard	Defines the end of the path for the HTTP connection. This property allows to define the sub path, relative to the connector root path, to the target web service URI. For example, if the target is: `http://server/MyApp/targetpage.jsp`, the connector server would be: `server`, the connector root path: `/MyApp` and the transaction sub path: `/targetpage.jsp`.
URL charset encoding	String	expert	Defines the charset encoding to use for the variable values sent as parameters in HTTP request. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is blank. If blank, the parent connector's **Default URL charset encoding** property value is used.
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.

EXAMPLES

Let’s consider a company named Global Company delivering a Web service for its Human

Resources department.This is a sample Web service accessible in REST and SOAP modes (developed thanks to Convertigo).

We defined an *HTTP connector*, in a project named `sample_refManual_http`, for connecting to this sample Web service in REST (for more information, see *HTTP connector* documentation and examples). In these examples, we are focusing on the transactions created to call some of this Web service's methods in JSON.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTTP connector objects examples*** *in the* ***New Project*** *wizard.*

Example 1

Several *JSON HTTP transaction* objects are created to call some methods of the target Web service in JSON. For example, one transaction, defined to call the `departmentsList` method of the Web service, is created with the following properties:

```
JSON HTTP transaction [
  accessibility=Public
  HTTP verb=GET
  sub path=/.json?__sequence=departmentsList
  include data type in XML response nodes=false
  JSON array translation policy=hierarchical
  JSON response encoding=UTF-8
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

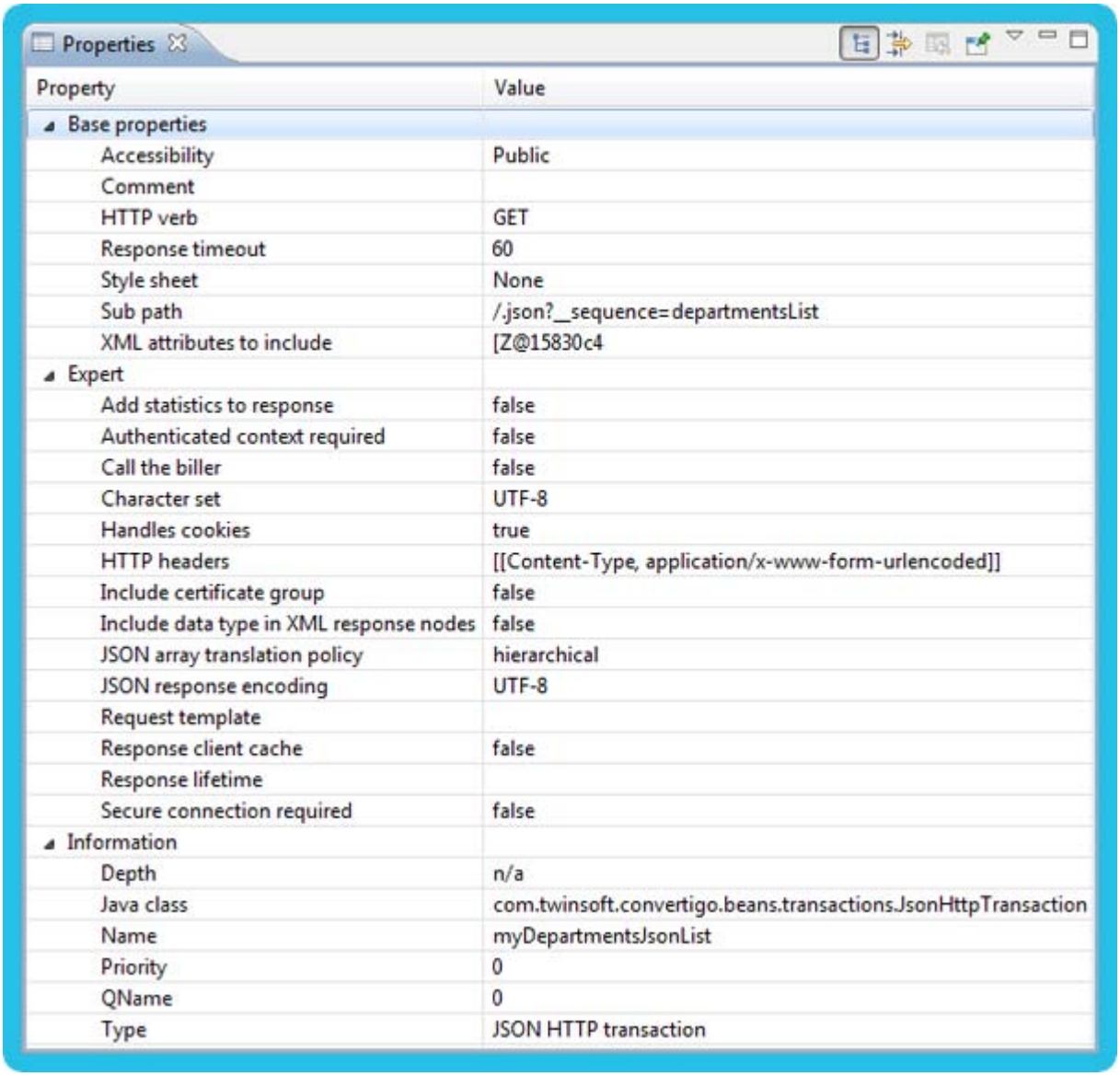


Figure 2 - 188: JSON HTTP transaction - Example 1 - Configuration example

The transaction is created in the **Transactions** folder of the connector, including various other transactions. It appears as follows in the **Projects** view:

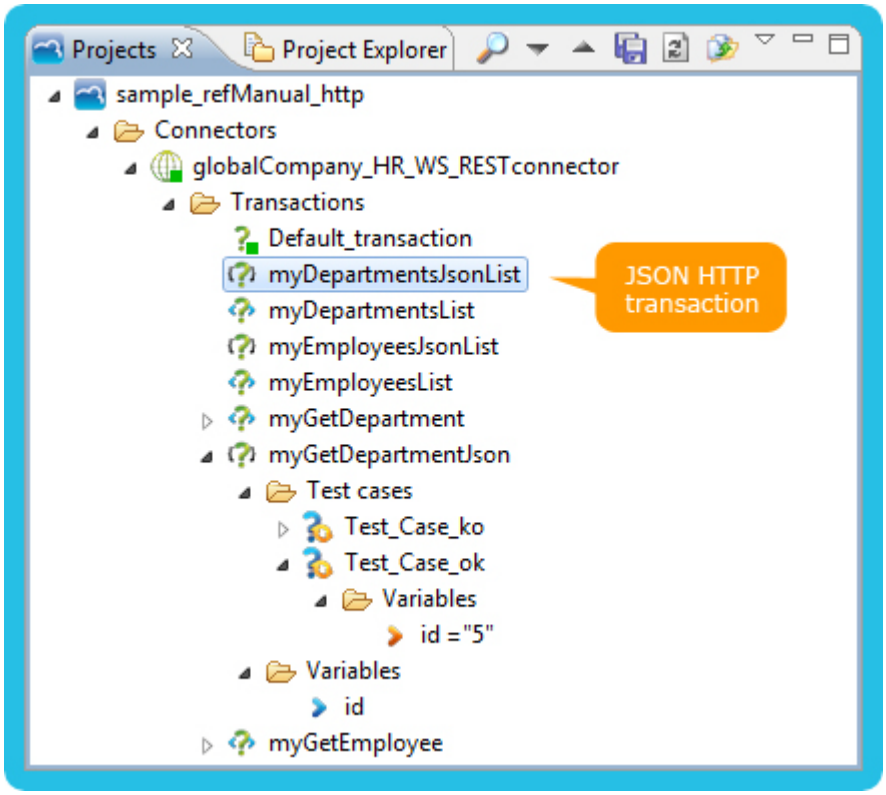


Figure 2 - 189: JSON HTTP transaction - Example 1 - Object in Projects view

Executing the `myDepartmentsJsonList` transaction (thanks to the **Execute** entry in the transaction's contextual menu or by pressing the F5 key) invokes the Web service method defined by the URL in **Sub path** property and gets the response from it. The connector editor shows:

- on the left part, the HTTP response received from the target Web service method,
- on the right part, in the **XML** tab, the transaction output XML generated from the JSON Web service response, using the `hierarchical` array translation policy:

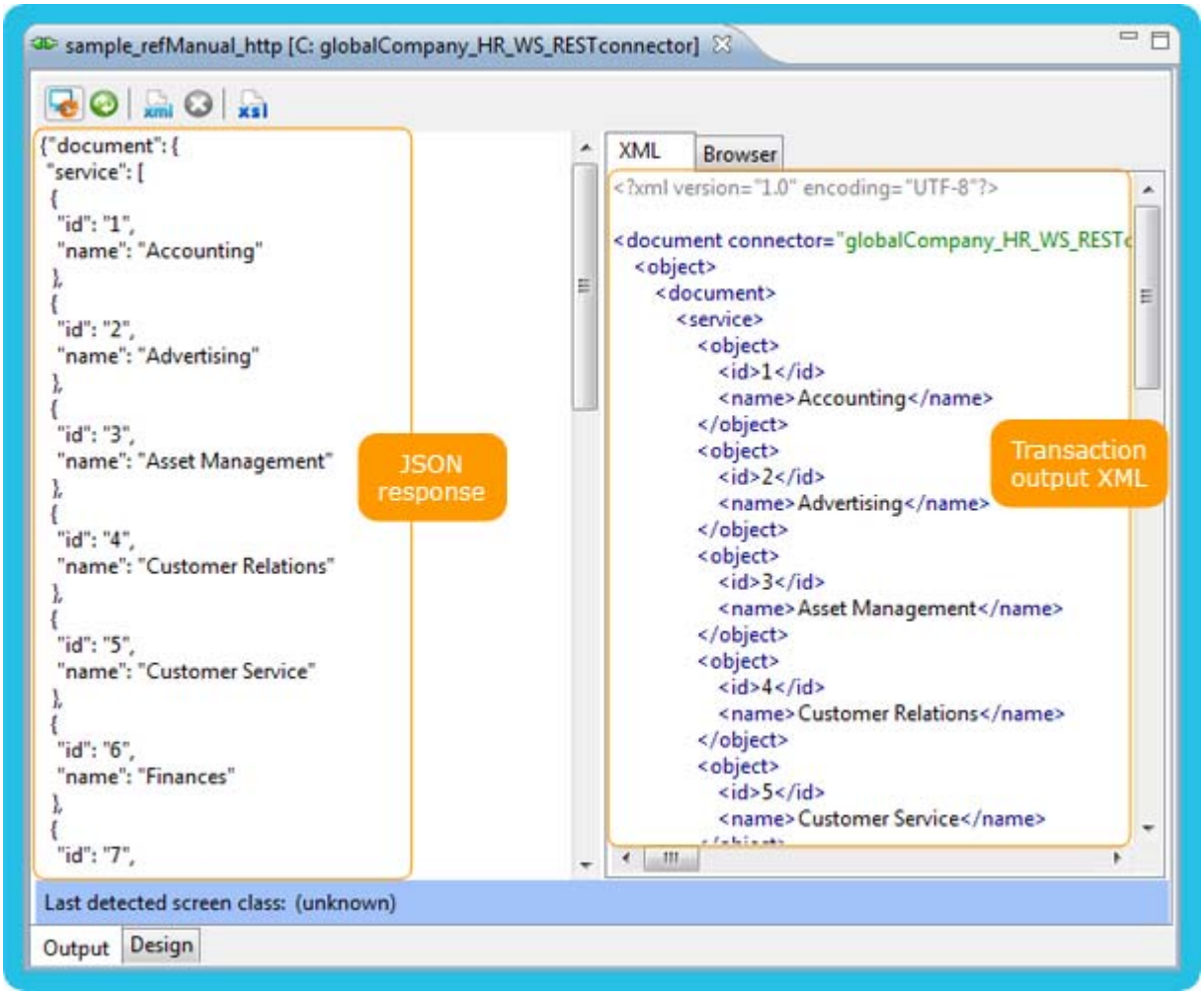


Figure 2 - 190: JSON HTTP transaction - Example 1 - Responses in connector editor

Example 2

Let's consider a second *JSON HTTP transaction* from the same *HTTP connector*, developped to call another method of the target Web service in JSON. This transaction, defined to call the `employeesList` method of the Web service, is created with the following properties:

```
JSON HTTP transaction [
   accessibility=Public
   HTTP verb=GET
   sub path=/.json?__sequence=employeesList
   include data type in XML response nodes=false
   JSON array translation policy=compact
   JSON response encoding=UTF-8
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

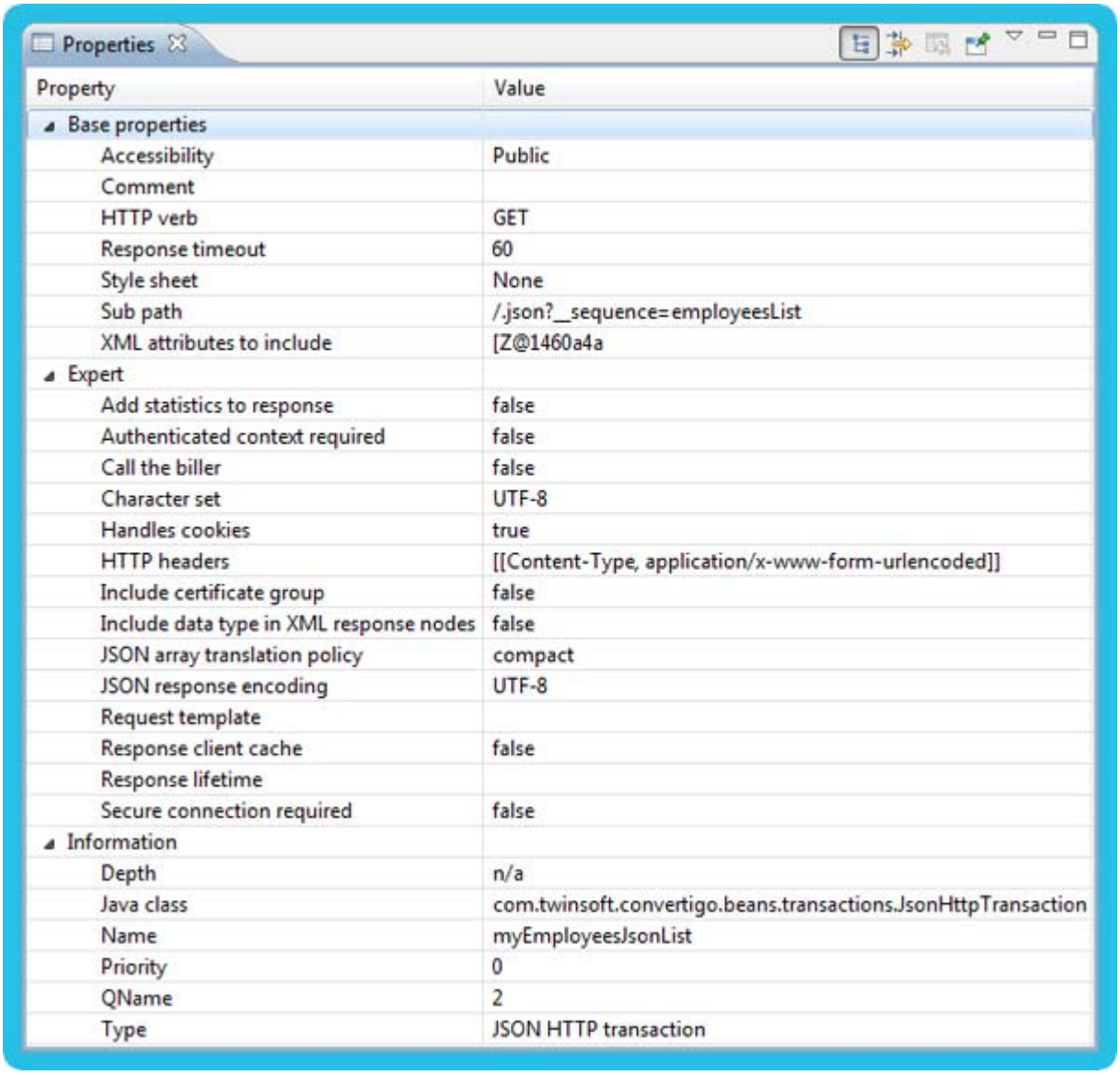


Figure 2 - 191: JSON HTTP transaction - Example 2 - Configuration example

The transaction is created in the **Transactions** folder of the connector, including various other transactions. It appears as follows in the **Projects** view:

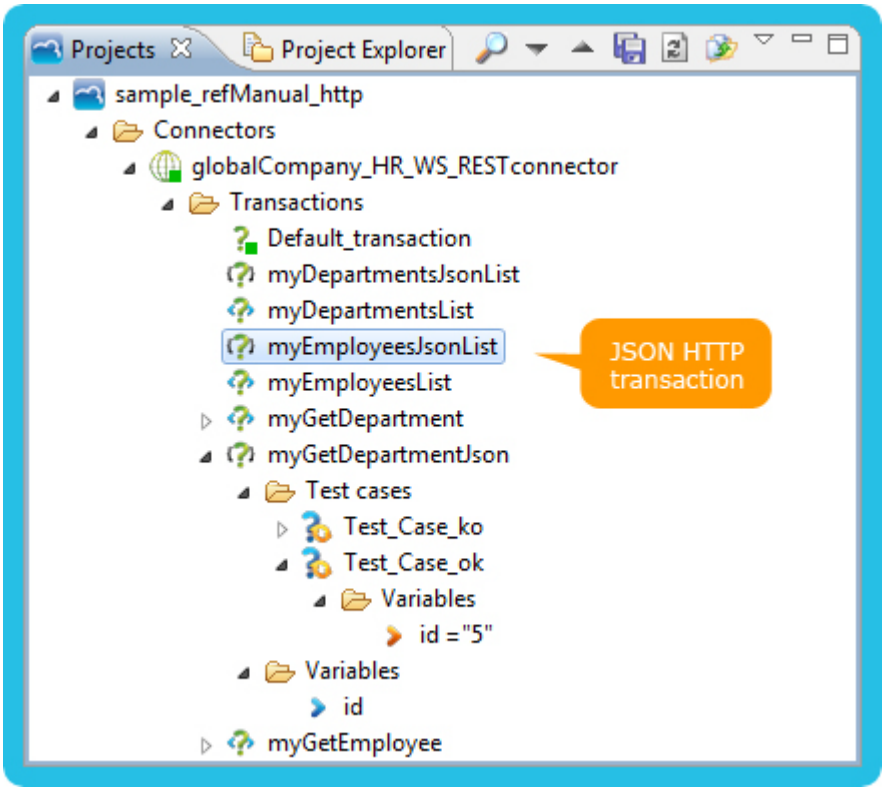


Figure 2 - 192: JSON HTTP transaction - Example 2 - Object in Projects view

Executing the `myEmployeesJsonList` transaction (thanks to the **Execute** entry in the transaction's contextual menu or by pressing the `F5` key) invokes the Web service method defined by the URL in **Sub path** property and gets the response from it. The connector editor shows:

- on the left part, the HTTP response received from the target Web service method,
- on the right part, in the **XML** tab, the transaction output XML generated from the JSON Web service response, using the `compact` array translation policy:

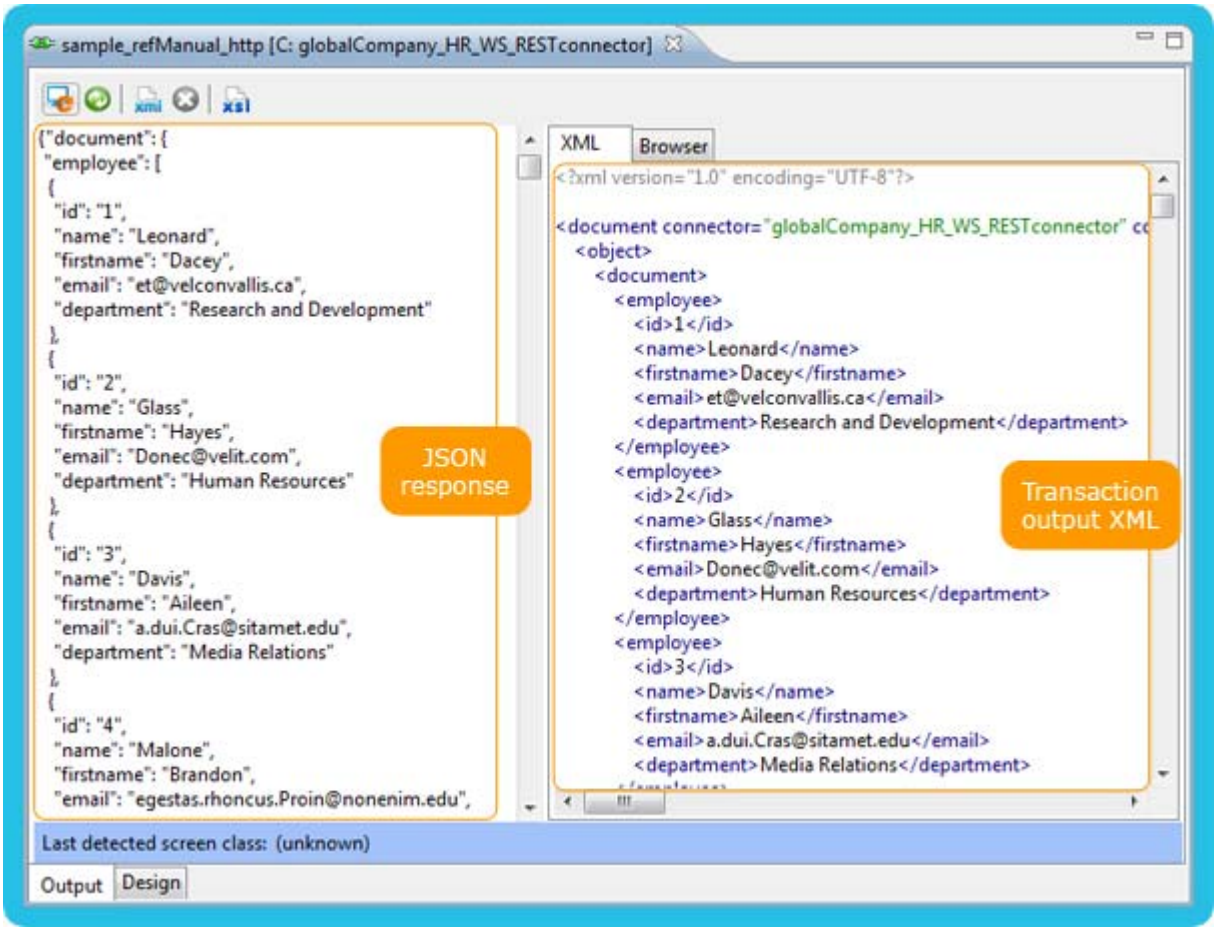


Figure 2 - 193: JSON HTTP transaction - Example 2 - Responses in connector editor

WHAT HAPPENED?

You can see the difference between the first example's transaction, that used the `hierarchical` array translation policy and this one which uses the `compact` translation policy: the array items are named by the array label (here `employee`), whereas in the first example the array items are named `object`, and were surrounded by a parent node named with the array label (which was `service`).

2.8 Web

2.8.1 Main objects

HTML CONNECTOR 

OBJECT DESCRIPTION

Establishes connections with an HTML application.

An *HTML connector* allows Convertigo to connect to a website to perform transactions, that is to say navigate through web pages and either:

- extract data into a proper XML document (CWI),
- clip defined web pages (CWC).

HTML connector is needed by Convertigo to connect to HTML applications. Once connected, all tasks (screen classes detection, data extraction, browsing, etc.) associated with the *HTML connector* can be carried out as defined in the project thanks to several objects:

- Screen classes,
- Criteria,
- Extraction rules,
- HTML transactions,
- Screen classes handlers,
- Statements.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Authentication type	Authentication Mode	expert	Defines the authentication type between basic and NTLM authentications. This property allows to define which type of authentication has to be used for the HTTP request. Default value is `Basic`. If **Basic/NTLM authentication user** and **Basic/NTLM authentication password** properties are not filled, no authentication is performed. Notes: • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.

Property	Type	Category	Description
Basic/NTLM authentication password	String	expert	Defines the user's password for basic or NTLM authentication. This property value is used as user password for basic or NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.
Basic/NTLM authentication user	String	expert	Defines the user name for basic or NTLM authentication. This property value is used as user name for basic or NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, the target application should accept `WWW-Authenticate` header. • If you are using NTLM authentication setting, do not forget to also fill the **NTLM authentication domain** property.
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default URL charset encoding	String	expert	Defines the default charset encoding to use for the transactions variable values sent as parameters in HTTP requests. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is `UTF-8`.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.

Property	Type	Category	Description
HTTP headers forwarding policy	XMLVector	expert	Defines the HTTP headers to forward and the policy to use to forward them. This property allows forwarding HTTP headers from the client browser to the target application. This property allows to define a list of HTTP headers to forward and, for each header, the forwarding policy to use. For each header, two columns have to be set: • **Header name**: defines the name of the header to forward, • **Forwarding policy**: defines how to replace the header value when forwarding it. This second property can take the following values: • `Merge`: If the forwarded header exists, its value is merged with existing one. If the forwarded header doesn't exist, it is added. • `Ignore`: If the forwarded header exists, its value is not replaced, it is ignored. If the forwarded header doesn't exist, it is added. • `Replace`: Replaces all headers without any condition by forwarded values. Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Ignore empty attributes	boolean	expert	Defines whether DOM empty attributes must be parsed or ignored. HTML elements can include empty attributes. This property allows removing these attributes from the page's DOM by setting it to `true`.
Is HTTPS	boolean	standard	Defines whether the connection is secured (HTTPS). If set to `true`, the connection is SSL-based. Make sure the target SSL port (usually 443) is correctly set.
NTLM authentication domain	String	expert	Defines the NTLM authentication domain in case of NTLM authentication. This property value is used as user domain for NTLM authentication. Notes: • The type of authentication is chosen using the **Authentication type** property. • If you are using basic authentication setting, this property does not need to be filled.
Parse mode	ParseMode	expert	Defines the HTML parser version (since Convertigo 4.0 / since Convertigo 4.5). This property takes one of the following values: • `4.0`: uses parsing mode developed since Convertigo 4.0 (original parsing mode), • `4.5`: uses parsing mode developed since Convertigo 4.5 (faster parsing mode, but sometimes not adapted).
Port	int	standard	Defines the server port.
Root path	String	standard	Defines the root path. This is the first URI requested by the HTTP connector. Any other URI in the project is relative to this URI.

Property	Type	Category	Description
Server	String	standard	Defines the server name (or its IP address). This property defines the DNS name or IP address of the target application server.
Trust all certificates	boolean	standard	Defines whether trusted certificates must be checked. In SSL mode, the server sends existing certificates to Convertigo. In most cases, set this setting to `true` to automatically trust all server certificates. If set to `false` , target server certificates must be installed in Convertigo.

EXAMPLES

Example 1

The following is an example of *HTML connector* set in the context of the "*Starting With Convertigo Web Integrator*" *tutorial*.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

This *HTML connector* connects to the Google website:

```
HTML connector [
   is HTTPs=false
   server=www.google.com
   port=80
   root path=/
   trust all certificates=true
   carioca authentication=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

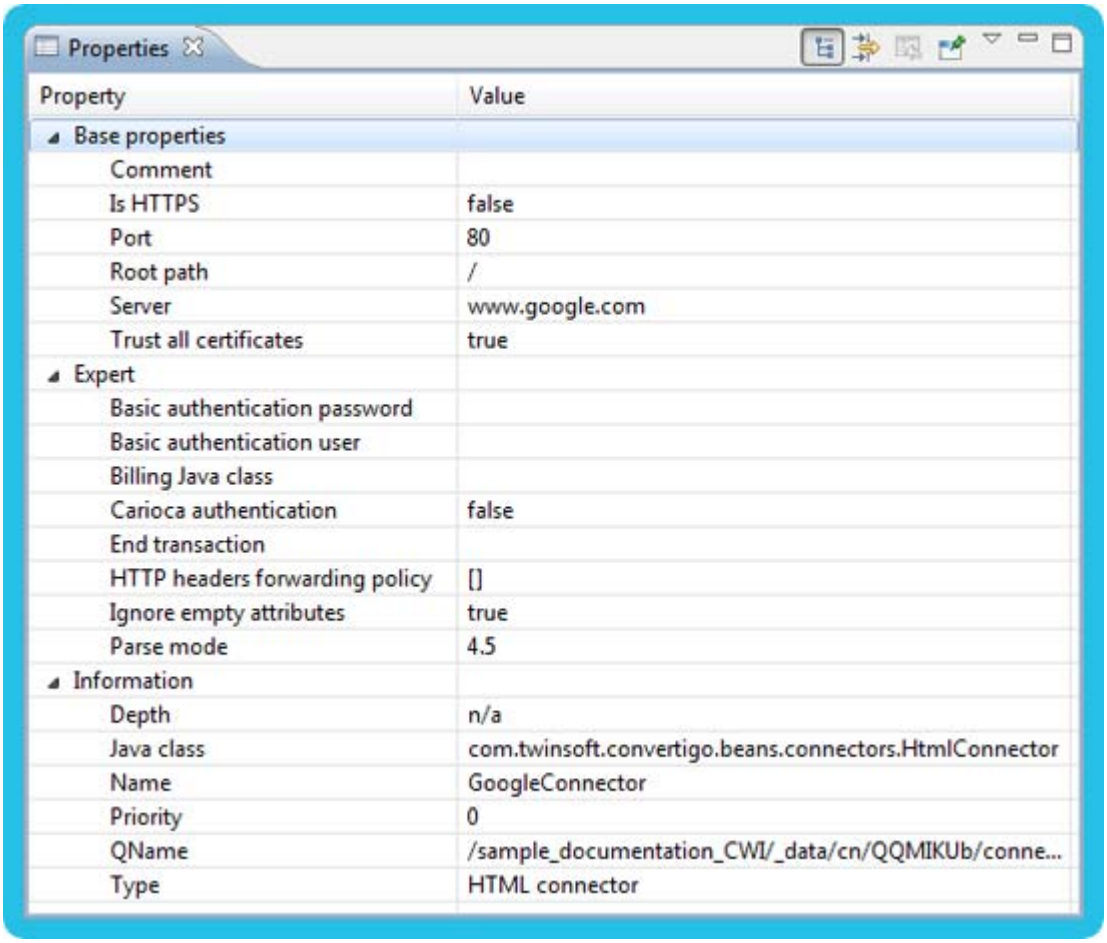


Figure 2 - 194: HTML connector - Example 1 - Configuration example

The connector is created in the **Connectors** folder of the project, including various other objects, such as screen classes and transactions. It appears as follows in the **Projects** view:

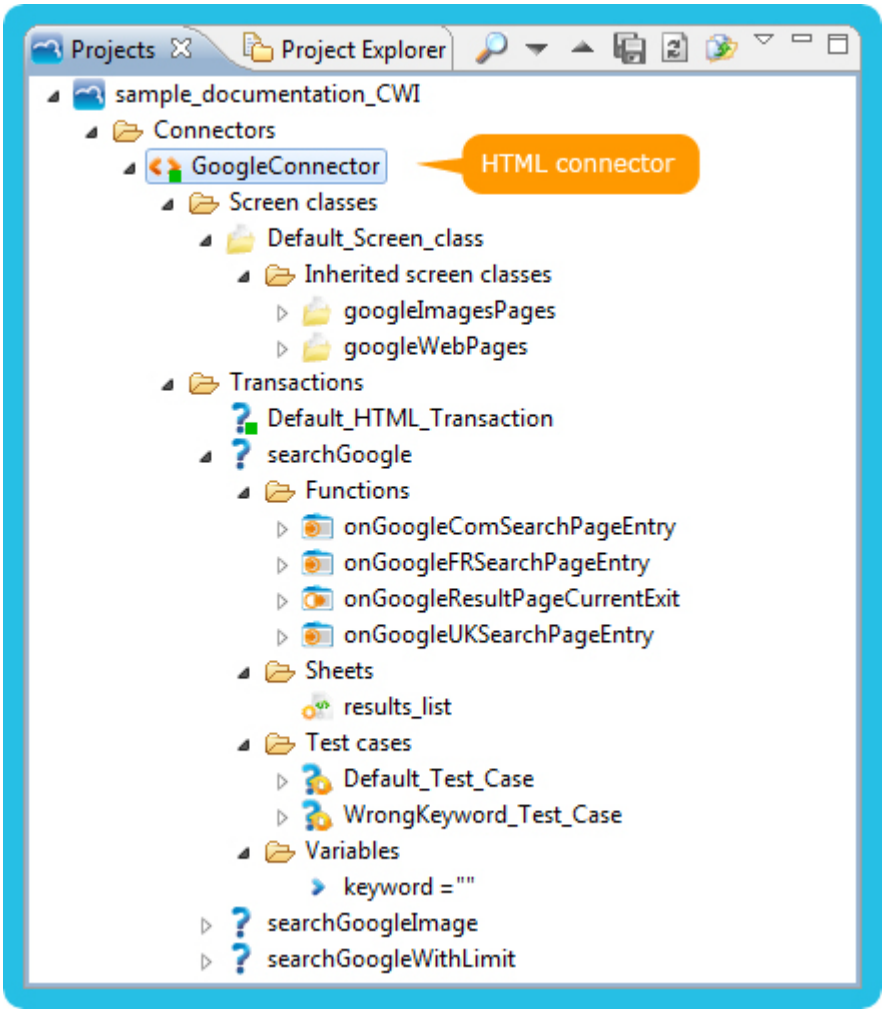


Figure 2 - 195: HTML connector - Example 1 - Object in Projects view

The **HTML connector** editor displaying current HTML page with the corresponding DOM tree, and the XPath evaluator tool, appears as follows:

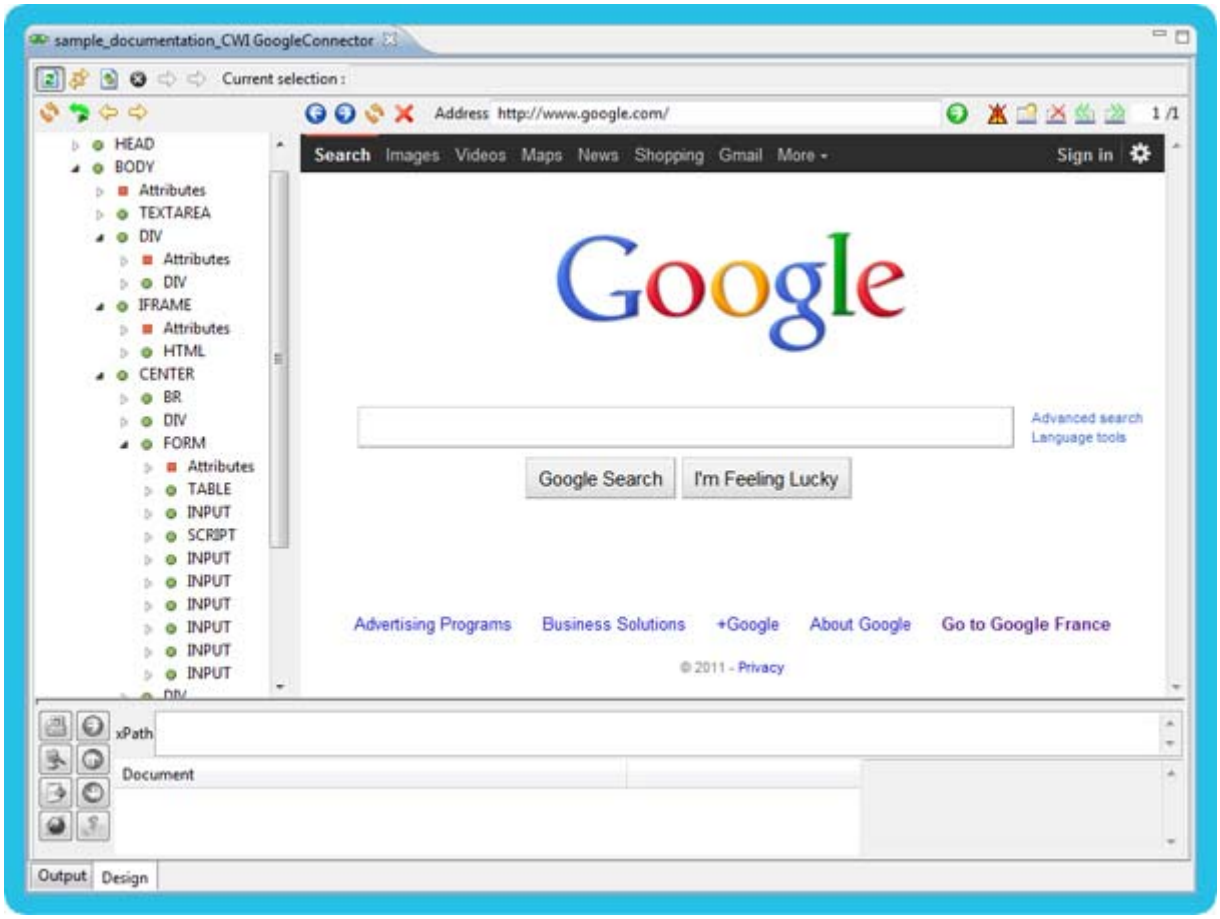


Figure 2 - 196: HTML connector - Example 1 - Web page in editor's Web browser

The default application URI (concatenation of **IsHTTPs**, **Server**, **Port**, and **Root path** properties) HTML page is displayed in the **Web browser** of the **Connector** editor.

Example 2

Let's consider Convertigo website, accessible at the following URL: `http://www.convertigo.com`. In this example, we want to define an *HTML connector*, in a new project named `sample_refManual_webClipper`, allowing to connect to this website.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

This *HTML connector* connects to the Convertigo website:

```
HTML connector [
   is HTTPs=false
   server=www.convertigo.com
   port=80
   root path=/
   trust all certificates=true
   carioca authentication=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

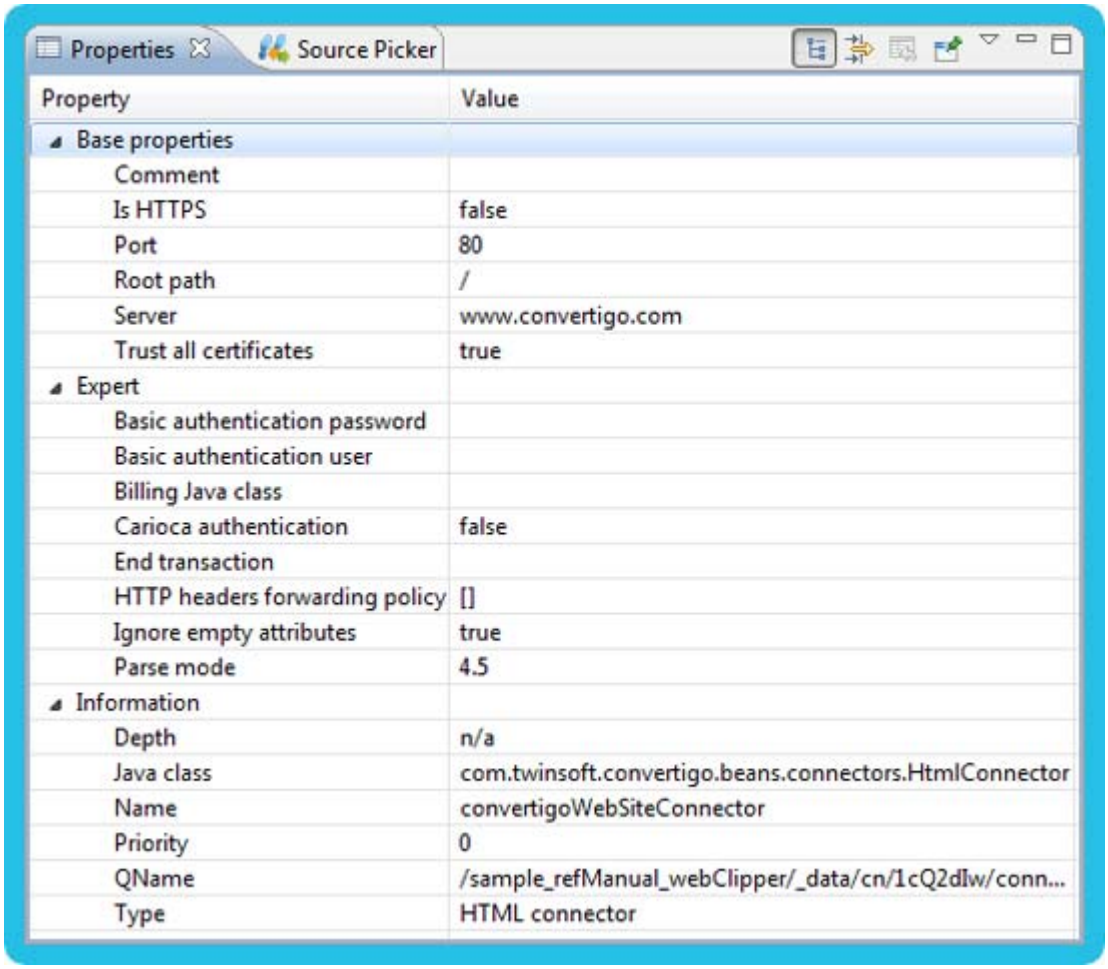


Figure 2 - 197: HTML connector - Example 2 - Configuration example

The connector is created along with the Web Clipping project that contains it. Therefor, it appears in the **Connectors** folder of the project, including various other objects, such as a screen classes structure (specific to Web Clipping projects) and transactions. It appears as follows in the **Projects** view:

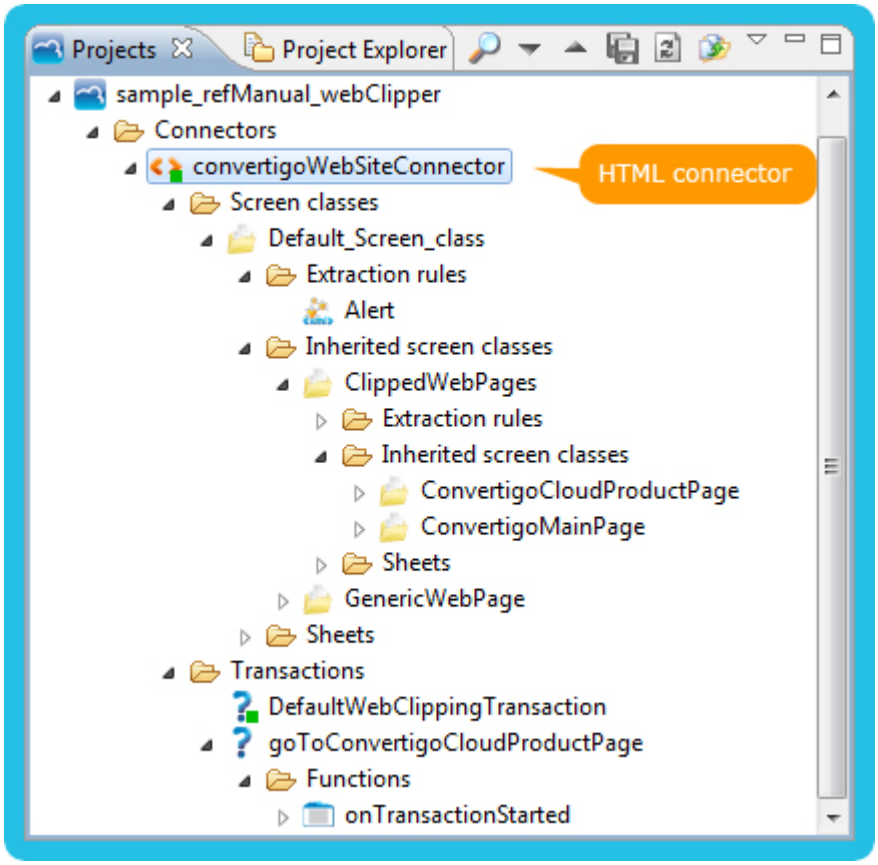


Figure 2 - 198: HTML connector - Example 2 - Object in Projects view

The **HTML connector** editor displaying current HTML page with the corresponding DOM tree, and the XPath evaluator tool, appears as follows:

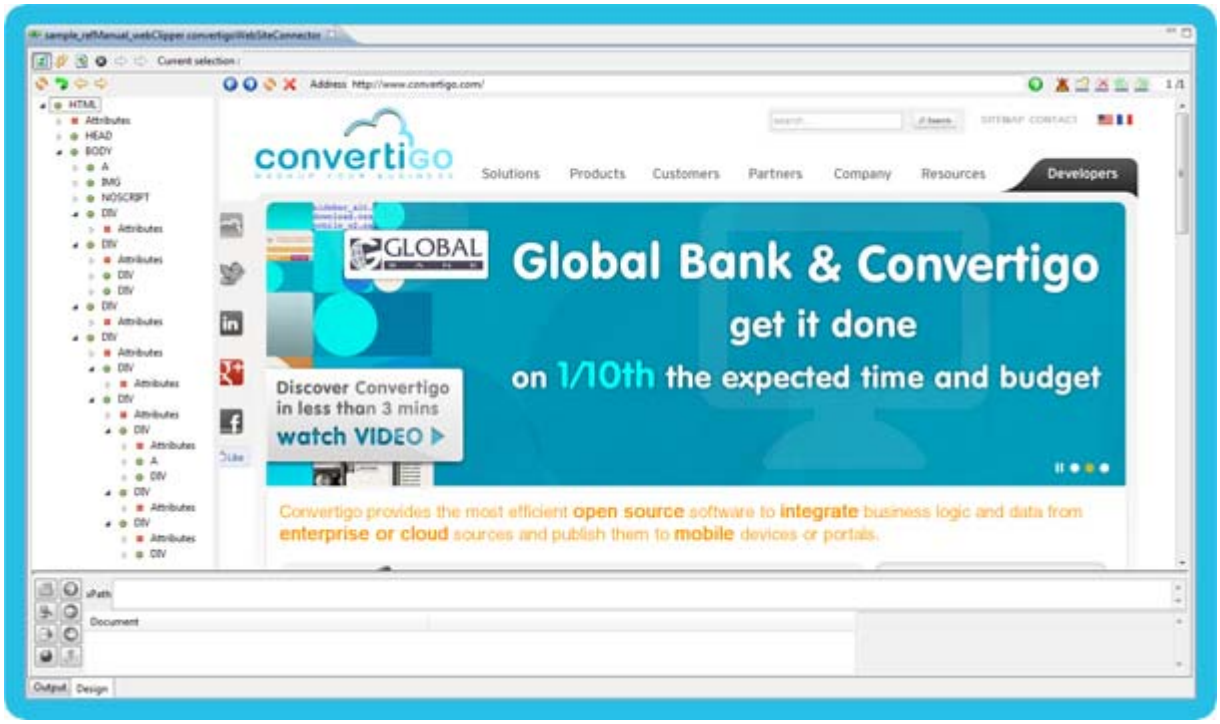


Figure 2 - 199: HTML connector - Example 2 - Connector editor with Web browser, DOM tree and XPath evaluator

The default application URI (concatenation of **IsHTTPs**, **Server**, **Port**, and **Root path**

properties) HTML page is displayed in the **Web browser** of the **Connector** editor.

HTML TRANSACTION 

OBJECT DESCRIPTION

Defines an HTML transaction.

As regards web applications, Convertigo's basic principle is to:

- connect to websites,
- detect defined screen classes and navigate through web pages,
- extract web page data using screen class-specific extraction rules,
- treat extracted data in order to generate an XML document structured as required.

This process defines the notion of *HTML transaction*.

HTML transactions are based on input variables, as other transactions. They execute actions that are programmed using event handlers and statements, they are designed to handle web pages navigation. *HTML Transactions* return data in an XML structure resulting from automated navigation on the target website and extraction rules execution on these pages or, in the case of *Web Clipper* extraction rule, clip required HTML parts of the web pages.

A Convertigo project can contain several *HTML transactions*, each of them being exposed as standard REST or SOAP web service method.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/ sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/ sequences and run hidden ones if he knows their execution URL.
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/ sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.

Property	Type	Category	Description
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
HTTP headers	XMLVector	expert	Defines HTTP headers to be sent. *HTML transaction* can connect automatically to a target web page without using a specific statement (see the **Maintain connector state** property). The **HTTP headers** property allows to define the request Header Fields to be sent with the request to connect to the target web page. For each header, two colums have to be parametered: • **Variable**: HTTP header name (ex: `Content-Type`). • **Value**: HTTP header value (ex: `application/x-www-from-urlencoded`).
HTTP verb	int	standard	Defines the HTTP verb to use for this HTTP request: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `TRACE`, `OPTIONS` or `CONNECT`. For more information about HTTP verbs, you can visit the following RFC page: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*.
Handles cookies	boolean	expert	Defines whether cookies must be handled. If set to `true` (default value), the transaction maintains cookies in Convertigo's context. Default value should not be changed unless you specifically want the transaction to ignore cookies while browsing.
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.
Maintains connector state	boolean	expert	Defines whether the connector state must be maintained between transactions. If set to `false` (default value), each time the transaction is executed, it connects the connector browser to the target web page specified in the **Sub path** property using the **HTTP headers** property for headers, and the **Synchronization** property to synchronize on the accessed page. If set to `true`, the transaction maintains the connector state, i.e. uses the browser in the state in which it has been left when the preceding transaction ended.

Property	Type	Category	Description
Request template	String	expert	Defines the request body template file path. HTTP request sent by the transaction can contain data in its body. This data is based on a user-defined template file, which can be: • an XML file describing the content of the HTTP request body, possibly including transaction input variables in the data structure, • an XSL file used to transform the variable-based transaction input XML to generate the content of the HTTP request body. This property allows to define the path of the template file, it is either: • a local file, by default relative to the project's directory, or to the project's current subfolder, • a local file relative to the Convertigo webapp common `templates` directory, • an absolute path. If the template file is an XML file, it can contain transaction variables identified with a specific syntax in the XML and dynamically replaced at runtime with received variable values. The syntax to use in the XML template file to refer to a transaction variable is the following: • `$(<variableHttpName>)`: this simple notation starts with a $ character and then includes between brackets the **HTTP name** of the variable. Beware that the **HTTP name** of the variable can be different from the variable name (see *Variable* objects documentation). • `$(<variableHttpName>)concat`: this notation is very similar to the preceding, excepted that the last bracket is followed by the `concat` keyword. It starts by a $ character and includes between brackets the **HTTP name** of the variable, that should be in this case a *Multi-valued variable*. The `concat` keyword implies that all values received in the *Multi-valued variable* must be concatenated before replacing this notation by this computed value in the template XML. • `$(<variableHttpName>)`: this notation is identical to the first notation, but the behavior is different for a *Multi-valued variable*. The tag surrounding this notation in the template XML is duplicated for each value in the *Multi-valued variable*.
Response client cache	boolean	expert	Defines whether the transaction/sequence response should be cached by the client. If set to `false`, the response XML is sent to the client along with HTTP headers forcing the client browser not to store it in its local cache. This is the default value, since dynamic responses are usually preferred. If set to `true`, the XML response is sent normally.

Property	Type	Category	Description
Response lifetime	String	expert	Defines the response time-to-live (in seconds) in cache, i.e. the time during which the cached response remains valid or time interval for its renewal. This property enables the cache when filled, disables the cache when left empty. The **Response lifetime** property allows to specify the cache settings for the transaction/ sequence's response. It can be set to the following values: • `<empty>`: Disables the cache for the transaction/sequence. The response will not be cached and each request will execute the complete transaction. It is the default value. • `absolute,<time in secs>`: Enables the cache for the transaction/sequence. The response will be cached for the time specified in seconds. If an other request with the same parameters occurs within this time, the response will be returned from the cache. • `daily,hh:mm:ss`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the current day is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `weekly,hh:mm:ss,w`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `wth` day of week is reached. For Sunday w = 1, for Monday w = 2 ... and for Saturday w = 7. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** • `monthly,hh:mm:ss,d`: Enables the cache for the transaction/sequence. The response will be cached until `hh:mm:ss` of the `dth` day of month is reached. If an other request with the same parameters occurs before this time, the response will be returned from the cache. **A new day starts at 00:00:00.** Notes: • The **Response lifetime** property editor proposes a *Generator tool* that can help you configure the **Response lifetime** setting. • The *Variable* objects contain the **Cache key** property that allows to specify to use this variable as a key to the cache or not. See *Variable* objects documentation for more information.
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.

Property	Type	Category	Description
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Style sheet	int	standard	Defines how the XML returned by the transaction has to be processed by XSLT. This property can take the following values: • `None`: Do not process with XSLT. Usual setting for web services (SOAP or REST) where plain XML data is to be returned. • `From transaction`: Use the XSL style sheet attached to the transaction. When used, make sure a style sheet object is added to the transaction. • `From last detected screen class`: Use XSL style sheet attached to the last detected screen class (in case of a transaction with screen classes). Transactions using sheets from last detected screen class are mainly used in Web Clipping or Legacy Publishing projects.
Sub path	String	standard	Defines the end of the path to use for connecting to the target web page. *HTML transaction* can connect automatically to a target web page without using a specific statement (see the **Maintain connector state** property). The **Sub path** property allows to define the sub path, relative to the connector root path, to connect to the target web page URI. For example, if the target is: `http://server/MyApp/targetpage.jsp`, the connector server would be: `server`, the connector root path: `/MyApp` and the transaction sub path: `/targetpage.jsp`.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the transaction. HTML transactions can connect automatically (without using any specific statement) to a target web page (see the **Maintain connector state** property). This property allows to specify which type of synchronizer is used in the connection process.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

Property	Type	Category	Description
URL charset encoding	String	expert	Defines the charset encoding to use for the variable values sent as parameters in HTTP request. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is blank. If blank, the parent connector's **Default URL charset encoding** property value is used.
XML attributes to include	boolean[]	standard	Defines, when applicable, the XML attributes to be included in the generated XML document. These attributes are: • Definition attributes: `name`, `type`; • Position attributes: `line`, `column`; • Color attributes: `foreground`, `background`; • Decoration attributes: `bold`, `underline`, `reverse`, `blink`; • Optional attributes.

EXAMPLES

The following is an example of *HTML transaction* set in the context of the "*Starting With Convertigo Web Integrator*" *tutorial*.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

The `searchGoogle` transaction is an HTML transaction which purpose is to connect to the `www.google.com` Google search page, send a search request from an input `keyword` variable, and extract result Web pages titles and URLs returned by the Google search engine.

The transaction is thus associated with a variable (the search `keyword`), a style sheet (for presenting data as required using XSL transformation), test cases for testing purpose and various statements and handlers implementing the transaction behavior.

The `searchGoogle` transaction appears as follows:

- in the **Projects** view of the Convertigo Studio:

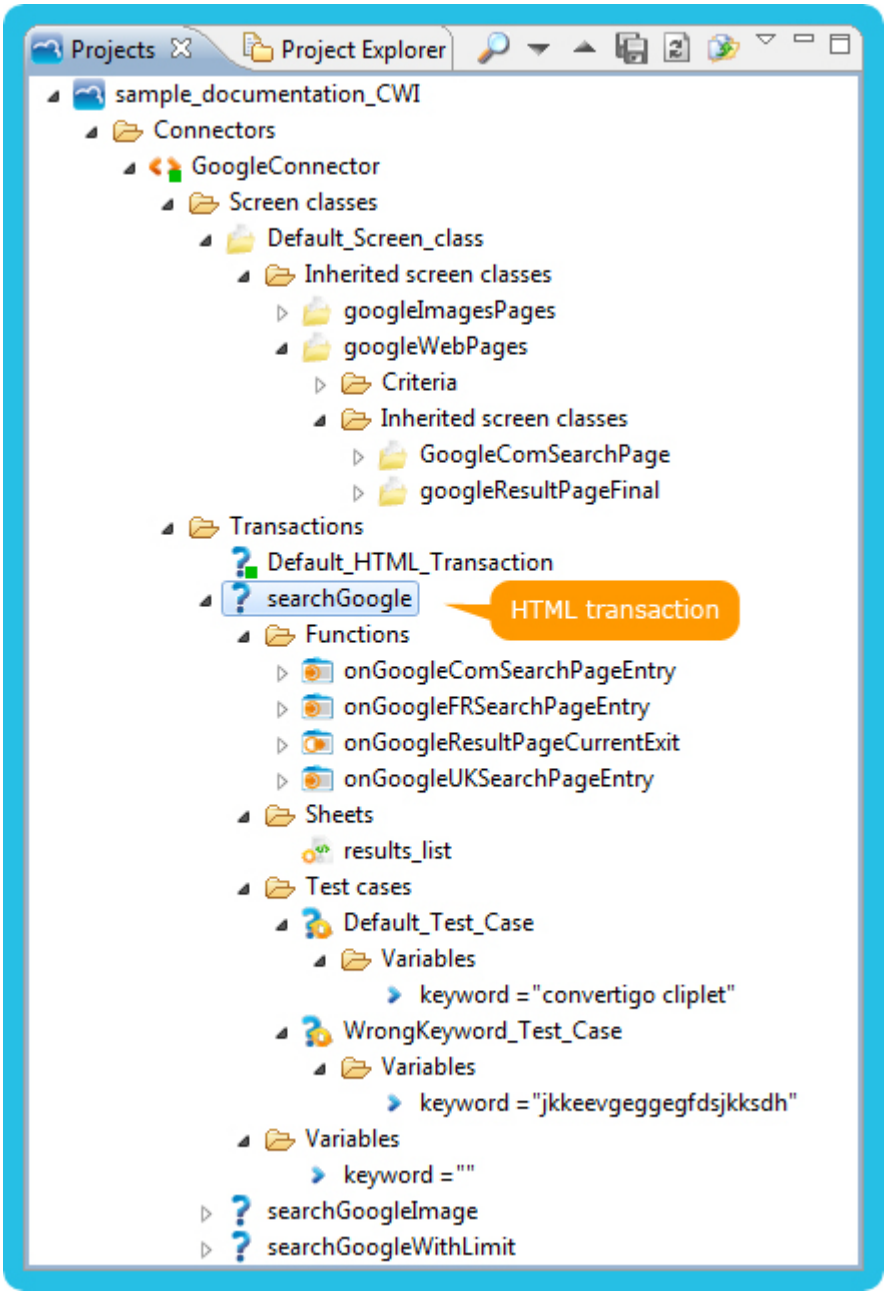


Figure 2 - 200: HTML transaction - Object in Projects view

- in the **Properties** view of the Convertigo Studio:

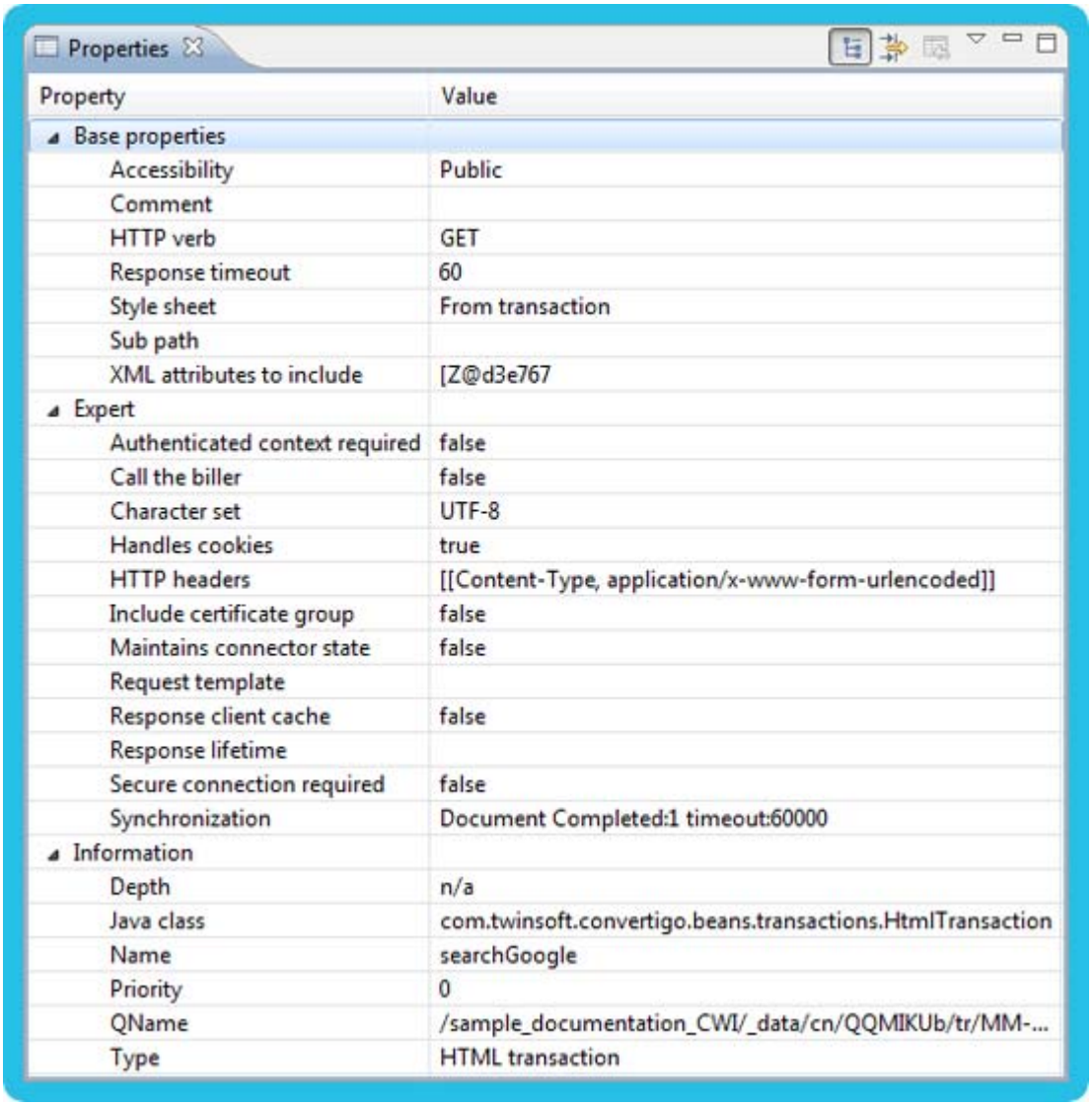


Figure 2 - 201: HTML transaction - Configuration example

The **Maintains connector state** property is set to `false` as the transaction should force the connector to reconnect to the Google search page before starting executing the statements.

The **Synchronization** property set to `Document Completed:1` allows to synchronize the connector after its reconnection, i.e. the execution of the transaction's handlers and statements only starts after the synchonizer has returned. This property is edited in the **Trigger editor**:

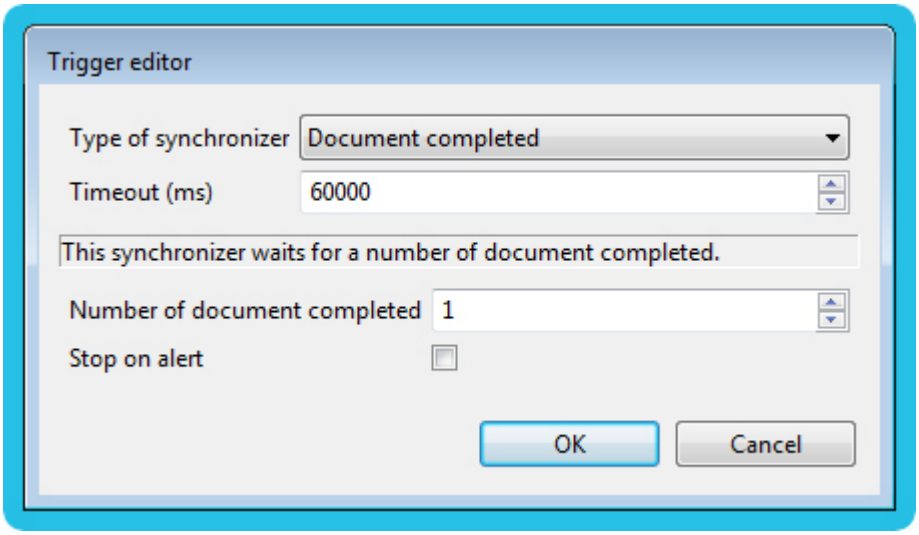


Figure 2 - 202: HTML transaction - Synchronization property edition

The **Style sheet** property is set to `From transaction`, which defines the use of the *Style sheet* object existing in the **Sheets** folder of the transaction.

HTML SCREEN CLASS 

OBJECT DESCRIPTION

Defines a group of screens with common features, in an HTML connector.

By the term "screen" is meant a set of identifiable data which may be rendered to the user or not. It is generally used regardless of the resource accessed by Convertigo (web page, Legacy screen, HTTP stream, etc.).

Thus, in the case of *HTML connector* projects (Web Integrator and Web Clipper), a screen may be defined by the data displayed in an HTML browser, for a web page display.

An *HTML Screen class* is identified by a set of criteria which are dedicated to screen's data detection. When accessing a screen (i.e. a web page) thanks to an *HTML connector*, Convertigo looks for detection criteria defined for screen classes in current connector.

Convertigo considers that the accessed screen belongs to the *HTML screen class* which all criteria match and which have the greatest number of criteria matching. For screen classes that would have the same number of matching criteria, Convertigo considers that the screen belongs to the screen class that has the greatest depth. And if screen classes also have the same depth, Convertigo considers that the screen belongs to the first screen class in alphabetical order.

For Web Integrator and Web Clipper projects (web pages in an *HTML connector*), detection criteria are *XPath* and *URL*. You can see these objects definitions and properties for more information.

An *HTML screen class* can also be associated with extraction rules executed on its detection by Convertigo. Extraction rules define which data are to be extracted from a screen and turned into a proper XML document.

HTML screen classes are pivotal in the execution of transactions, since their detection triggers the execution of screen class handlers (including actions to be performed on detected screens) and extraction rules (extracting data to be turned into XML).

Note: An *HTML screen class* do not define one screen only, but all screens matching the specified criteria. It is up to the Convertigo programmer to set detection criteria.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

EXAMPLES

Example 1

The `sample_documentation_CWI` project set in the context of the "*Starting with Convertigo Web Integrator*" tutorial includes a number of HTML screen classes created to meet the project and associated transaction (`searchGoogle`) needs.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

The purpose of the `searchGoogle` transaction is to connect to the Google search engine (`www.google.com` web page), to send a search request and to extract in XML format results titles and URLs.

Screen classes have therefore been organized into:

- a root screen class matching on any "Web" tab Google page (including search form web page and result list web page),
- inherited screen classes matching on specific pages (either on Google search or on result list web page).

The following table summarizes all the implemented screen classes with their criteria:

Table 2 - 1: `sample_documentation_CWI` project screen classes

Name	Description	Detection criterion
`googleWebPages`	Root screen class detected when accessing any Google Web tab page (including search form or result list page).	"Web" link activated in top left corner of the page.
`googleComSearchPage`	Detected when accessing the Google.com search page.	Root screen class criterion + Google logo.
`googleResultPageFinal`	Detected when accessing the final Google result page.	Root screen class criterion + Results container element.
`googleResultPageCurrent`	Detected when accessing a current Google result page (every page but the last).	Root screen class criterion + Final Result page criterion + "Next" link.

Once created, HTML screen classes appear together with their detection criteria and possible extraction rules in the **Projects** view of the Convertigo Studio:

Figure 2 - 203: HTML Screen class - Project's screen classes and respective criteria

Screen classes appear as follows in the **Properties** view (here, the `googleWebPages` screen class):

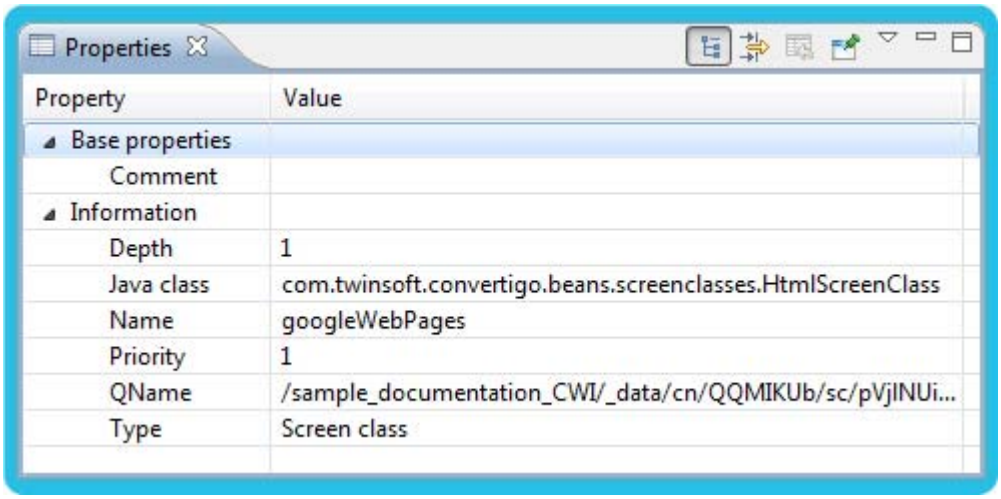


Figure 2 - 204: HTML Screen class - googleWebPages screen class properties

For more information about the different criteria defined to detect these screen classes, see "*Xpath criterion*" documentation and examples.

Example 2

Let's consider the following web page from Convertigo website:

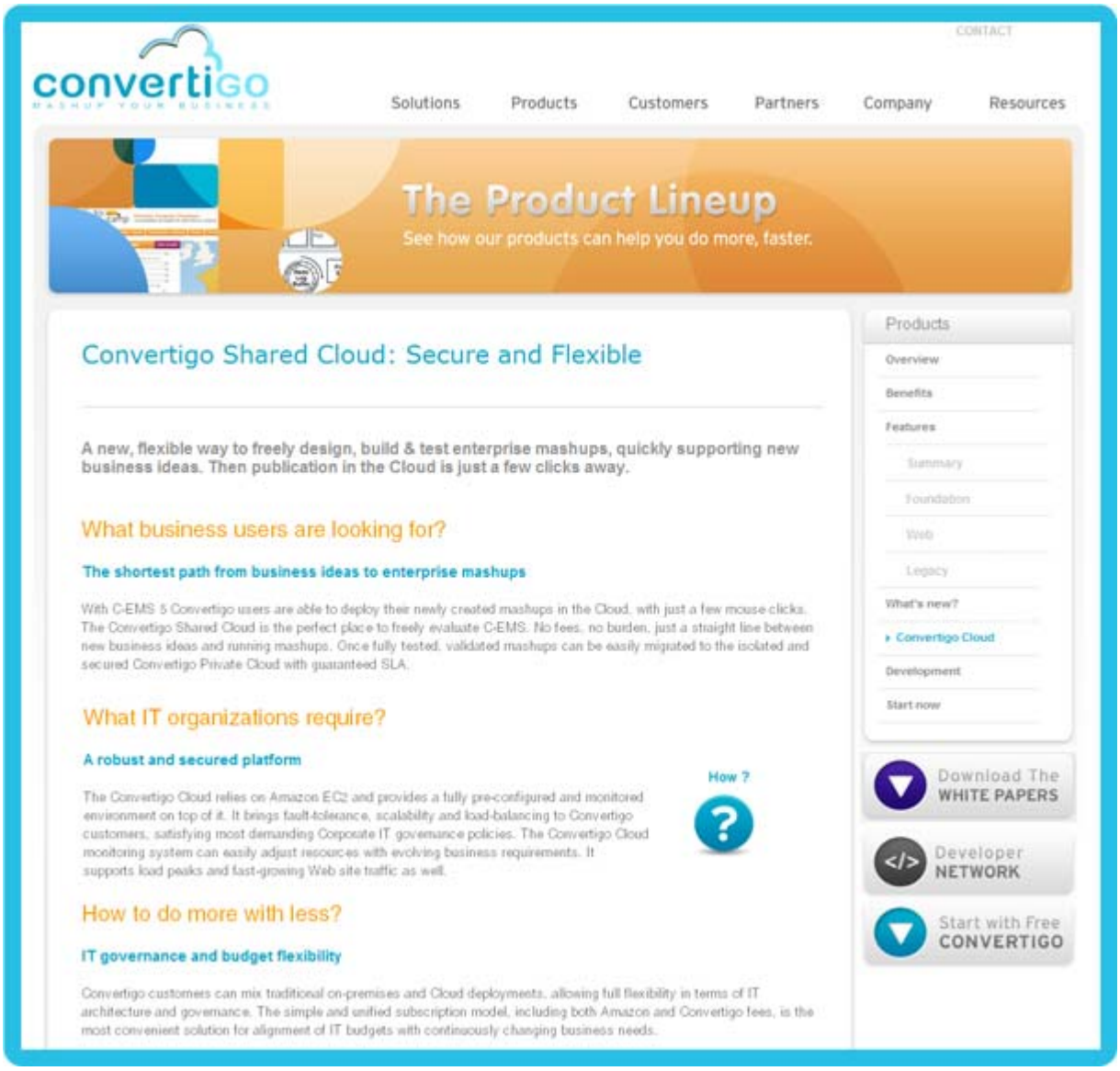


Figure 2 - 205: HTML Screen class - HTML web page

In this example, we want to define a *Screen class* matching this web page on a new project named `sample_refManual_webClipper`.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

A *Screen class* object has no properties to configure, it is defined by its criteria:

```
Screen class [

]
```

The *Screen class* object is created in the **Screen classes** folder of the connector, inherited from the *ClippedWebPages* screen class. Indeed, as this is a Web Clipper project, it includes the predefined screen classes structure: `Default_Screen_Class` with two inherited screen

classes `ClippedWebPages` and `GenericWebPage`. The screen class is created together with its first criterion and appears as follows in the **Projects** view:

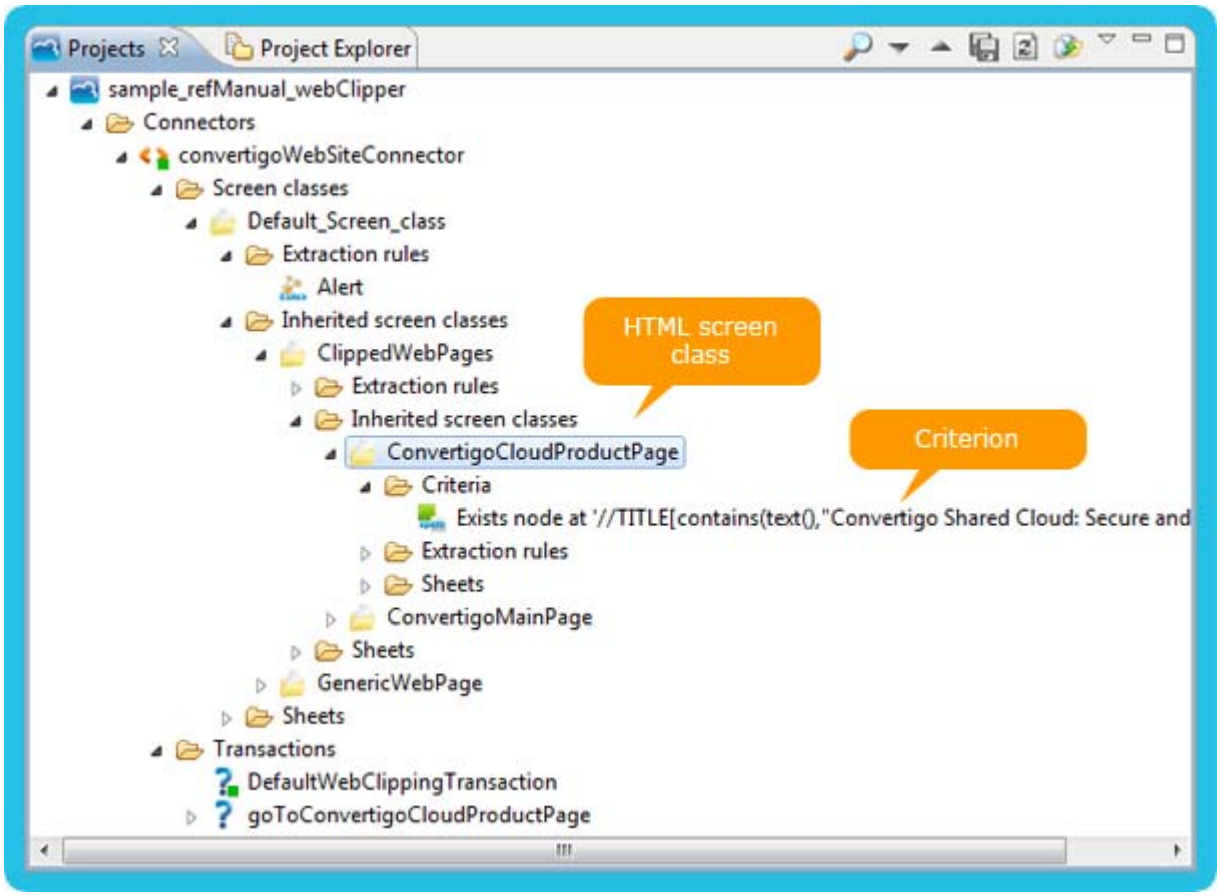


Figure 2 - 206: HTML Screen class - Screen class and first criterion in Projects view

Thanks to its criterion defining the remarkable characteristics of the page, this *Screen class* matches the previous web page. For more information about this criterion, see "*Xpath*" criterion documentation and examples.

2.8.2 Criteria

XPath 

Object Description

Defines a criterion based on an XPath for HTML screen classes.

The *XPath* criterion allows defining an XPath expression leading to one or several elements of a web page that uniquely identify a given screen class (logo, image, link, field, etc.).

Matching condition: The *XPath* criterion matches when the evaluation of the Xpath expression, defined in **XPath** property, on the HTML page's DOM gives a non-empty result.

Note: The *XPath* criterion, together with the *URL* criterion, are the two only possible criteria for detecting a screen class in CWC and CWI projects.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.
XPath	String	standard	Defines the XPath expression of searched nodes. The execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`. In these cases, the criterion matches. The execution of the **XPath** on the web page DOM can result in an empty result. In this case, the criterion doesn't match.

Examples

Example 1

The `sample_documentation_CWI` project set in the context of the "*Starting with Convertigo Web Integrator*" tutorial includes a number of HTML screen classes that are all detected using *XPath* detection criteria.

*You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Documentation samples** > **Web integration** in the **New Project** wizard.*

For example, the `googleWebPages` root screen class representing *any* Google Web page (either search or result) is detected if the first tab ("`Search`" on Google.com) appearing on the top left corner of the page is selected:

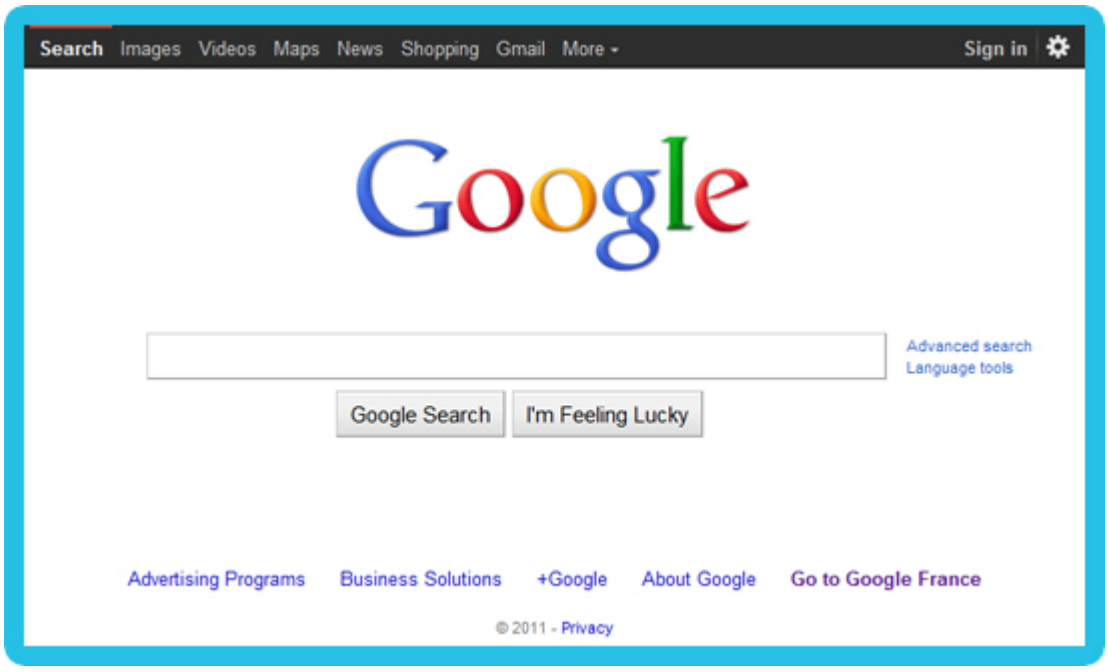


Figure 2 - 207: XPath criterion - Google Web page

The screen class detection criterion is first selected on screen with a right-click, then its XPath is generated from relevant elements in the DOM using the **XPath Evaluator** of the Convertigo Studio:

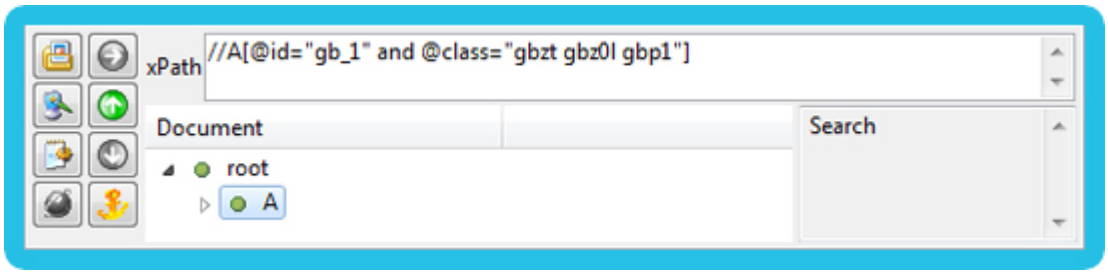


Figure 2 - 208: XPath criterion - Generating XPath in the XPath Evaluator

Then the *XPath* criterion is created with the following parameters:

```
XPath [
  xpath=//A[@id="gb_1" and @class="gbzt gbz0l gbp1"]
  reverse result=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

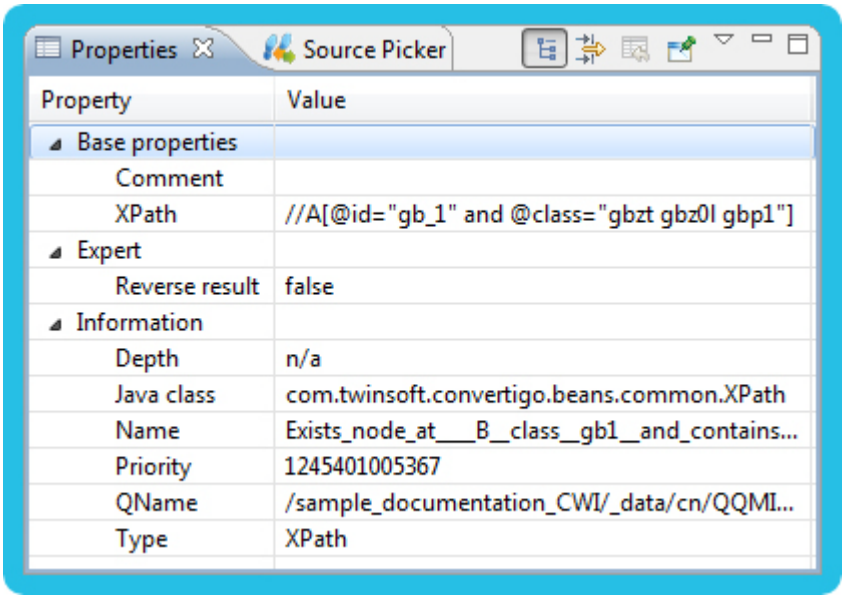


Figure 2 - 209: XPath criterion - Configuration example

The **XPath** property means that any HTML page with:

- a A node with the `id` attribute which is `gb_1`,
- and a `class` attribute which is `gbzt gbz0l gbp1`, meaning that this is the selected tab,

is a Web page detected as belonging to the `googleWebPages` screen class.

This can be checked using the **Show current screen class** function (button in the **Connector View**).

Once it is generated, the *XPath* criterion appears as follows in the screen class **Criteria** folder in the **Projects** view:

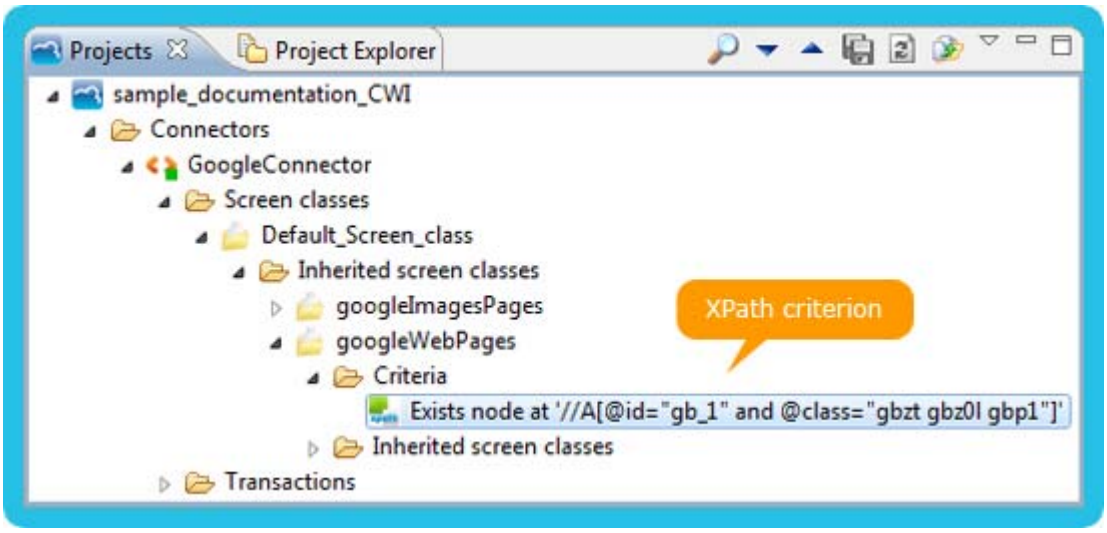


Figure 2 - 210: XPath criterion - googleWebPages screen class criterion in Projects view

Example 2

Let's consider the following web page from Convertigo website:

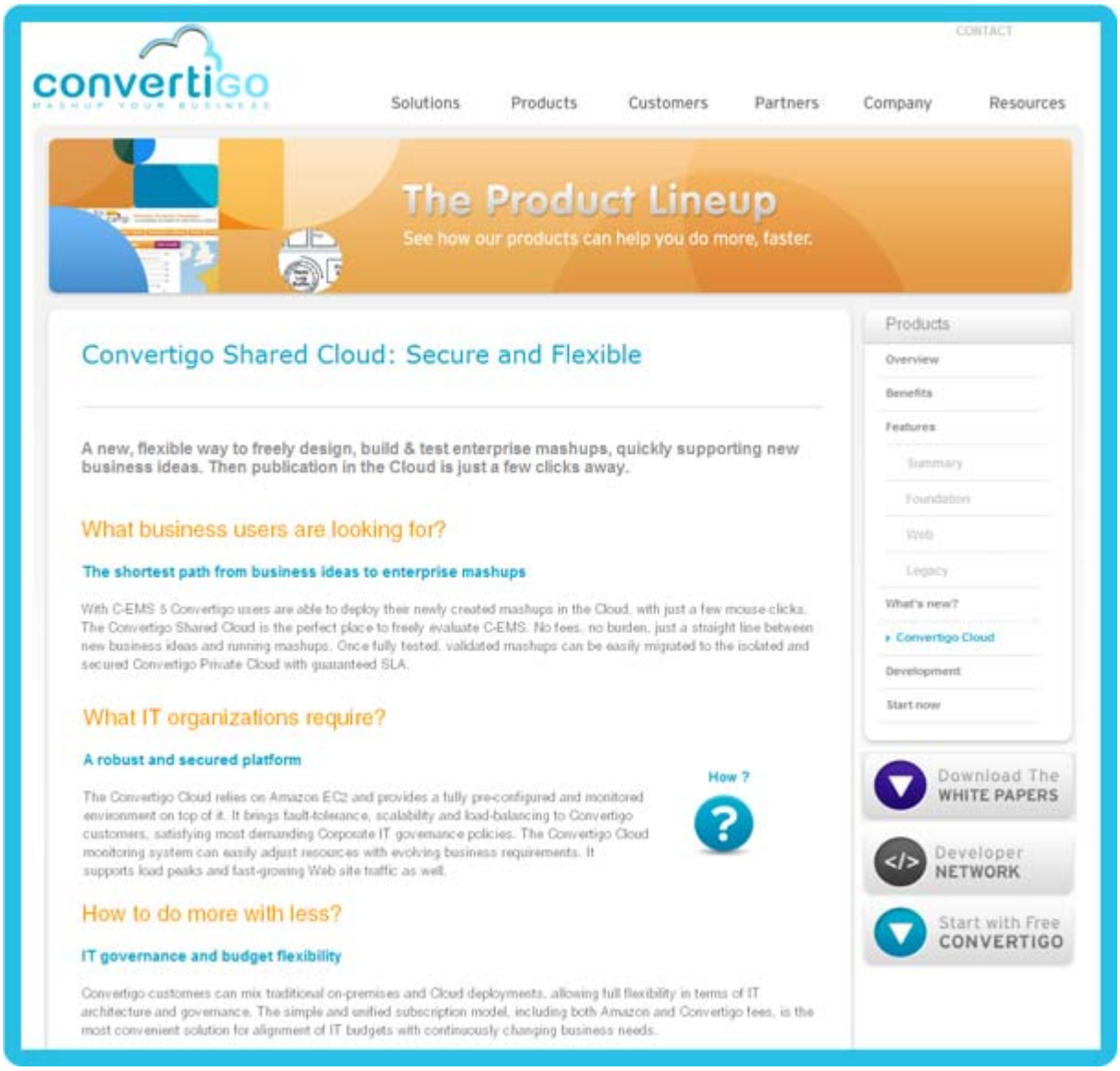


Figure 2 - 211: XPath criterion - HTML web page

A *Screen class* has been defined for matching this web page on a new project named `sample_refManual_webClipper`.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

This screen class is identified thanks to an *XPath* criterion, defined as follows:

```
XPath [
   xpath=//TD[@class="contentheading-products" and contains(text(),
   "Convertigo Shared Cloud: Secure and Flexible")]
   reverse result=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

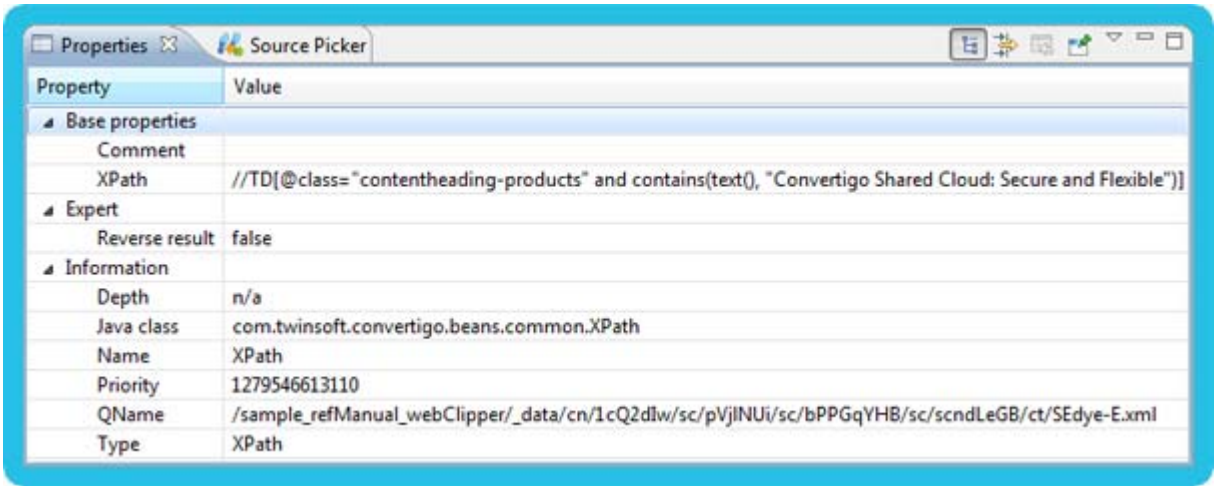


Figure 2 - 212: XPath criterion - Configuration example

The **XPath** property expression means that, if an HTML page contains:

- a table cell with a `class` attribute which is `contentheading-products`,
- and a text containing `Convertigo Shared Cloud: Secure and Flexible`,

then the web page matches this *XPath* criterion.

Once it is generated, the *XPath* criterion appears as follows in the screen class **Criteria** folder in the **Projects** view:

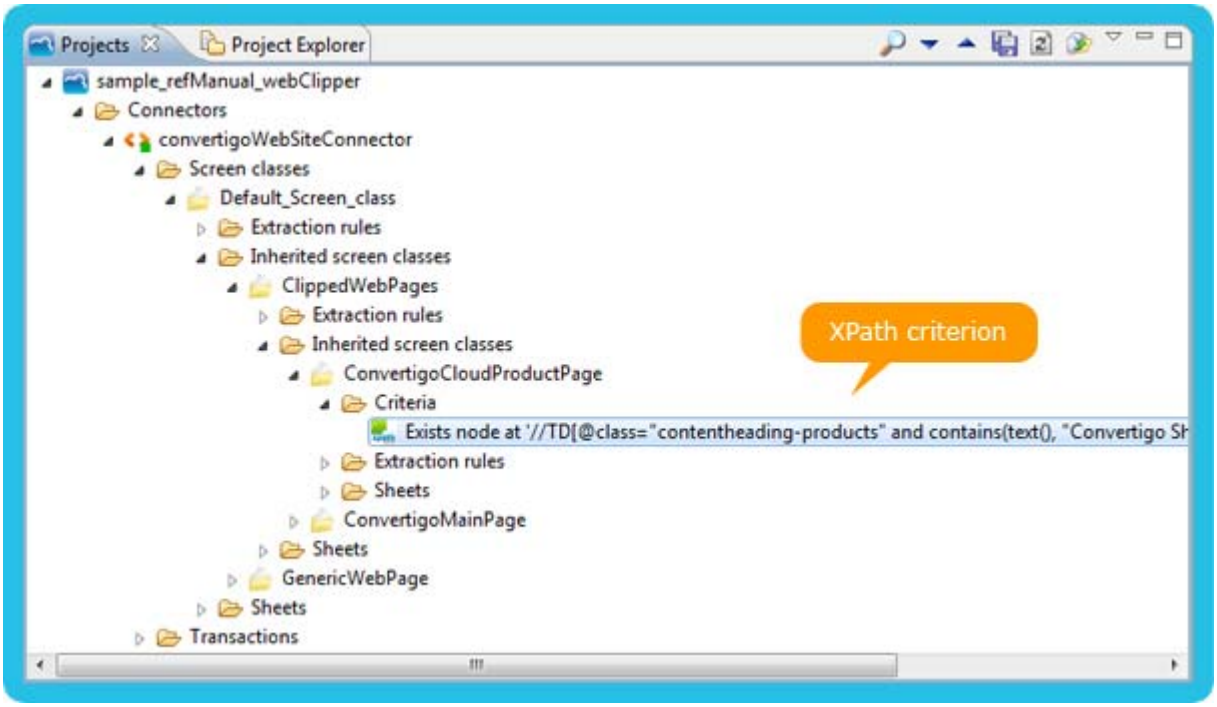


Figure 2 - 213: XPath criterion - Object in Projects view

This criterion matches on the previous web page from Convertigo website. The web page is then detected as belonging to the matching screen class. For more information about the screen class, see "*HTML Screen class*" documentation and examples.

URL (Web) 

Object Description

Defines a criterion based on requested URL for HTML screen classes.

The *URL* criterion allows defining a regular expression that is applied on current page's URL.

Matching condition: The *URL* criterion matches when the regular expression, defined in **Regular expression** property, matches the current page's URL, i.e. if the string pattern described by the regular expression is found in the page's URL.

Note: The *URL* criterion, together with the *XPath* criterion, are the two only possible criteria for detecting a screen class in CWC and CWI projects.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

2.8.3 Extraction rules

WEB CLIPPING EXTRACTION RULES

WEB CLIPPER 

OBJECT DESCRIPTION

Clips fragments of a web page.

The *Web Clipper* extraction rule extracts entire parts of an HTML page. It is applied if the result of the Xpath expression evaluation exists into the HTML page DOM.

The fragment resulting from the Xpath expression evaluation on the HTML page DOM is appended to the HTML transaction output document, with its presentation (images, style sheets, etc.) and behavior (scripts, links, etc.). The URI paths present in the elements of this fragment are changed by Convertigo in order not to point directly towards these URIs.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	XMLVector	configuration	Lists the names of attributes that have to be processed by the rewriting of URI paths. By default, a list of usual attributes is already set: `src`, `href`, `background`, `action`, `cite`, `classid`, `codebase`, `data`, `longdesc`, `usemap`. Note: A new attribute can be added to the list using the blue keyboard icon. The attributes defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Enable HTTP tunnel	HttpTunnel	configuration	Defines whether resources and links are retrieved through Convertigo HTTP tunnel or not ; and if so, whether the Convertigo HTTP tunnel uses cache or not. It can take three values: • `disable`: the resources are not got through Convertigo HTTP tunnel, but directly, • `cache`: the resources are got through COnvertigo HTTP tunnel and the tunnel uses cache, • `no cache`: the resources are got through COnvertigo HTTP tunnel and the tunnel doesn't use cache.
Enable parent extraction	boolean	configuration	Extracts all parents and ancestors of clipped elements for styles inheritance.
Is active	boolean	configuration	Defines whether the extraction rule is active.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Example 1

For the first example, let's consider the following web page from Convertigo website, "The Product Lineup, Convertigo Shared Cloud":

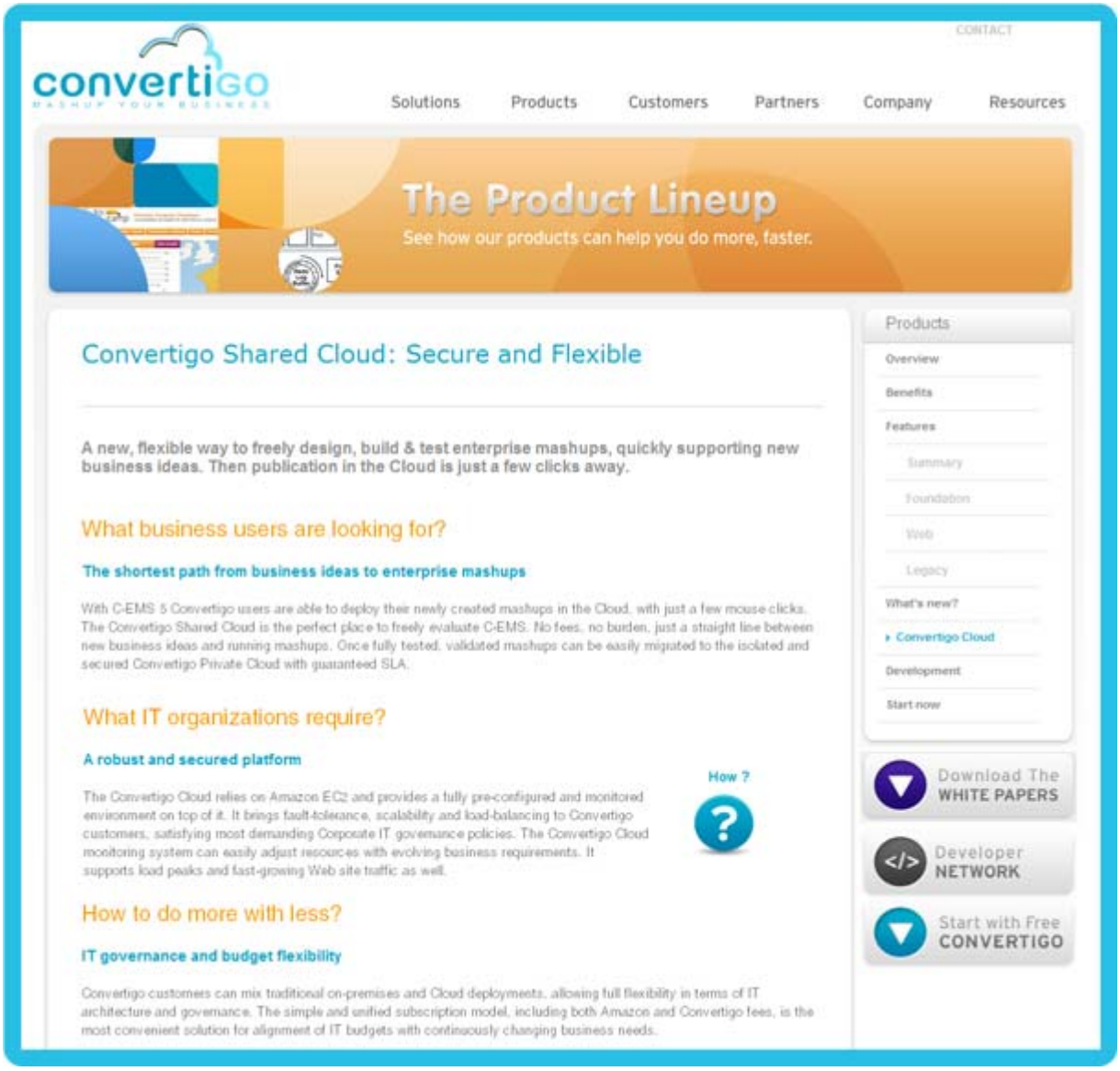


Figure 2 - 214: Web Clipper extraction rule - Example 1 - HTML web page

In this example, we want to extract the titles (in orange) and subtitles (in blue) of the page content thanks to a *Web Clipper* extraction rule, in a new project named `sample_refManual_webClipper`, allowing to connect to this website.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

Once accessing this web page in the connector editor thanks to the `goToConvertigoCloudProductPage` transaction, the Xpath matching both elements is

generated using the **Xpath Evaluator**:

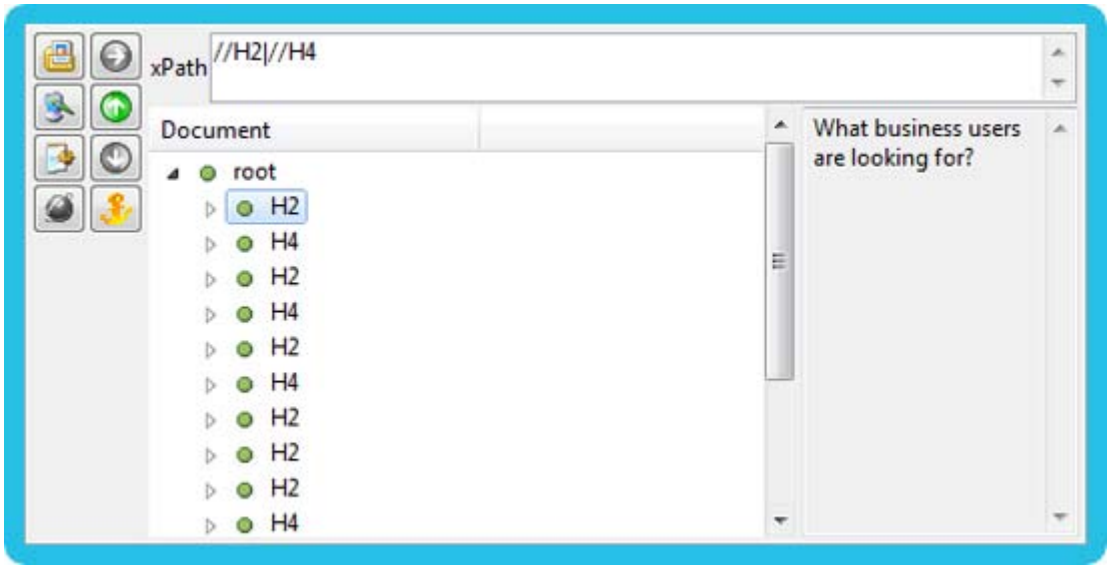


Figure 2 - 215: Web Clipper extraction rule - Example 1 - Generating Xpath in the Xpath Evaluator

Then the rule is created with the following parameters:

```
Web Clipper [
   xpath=//H2|//H4
   attributes=[src, href, background, action, cite, classid, codebase,
   data, longdesc, usemap]
   enable HTTP tunnel=disable
   enable parent extraction=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

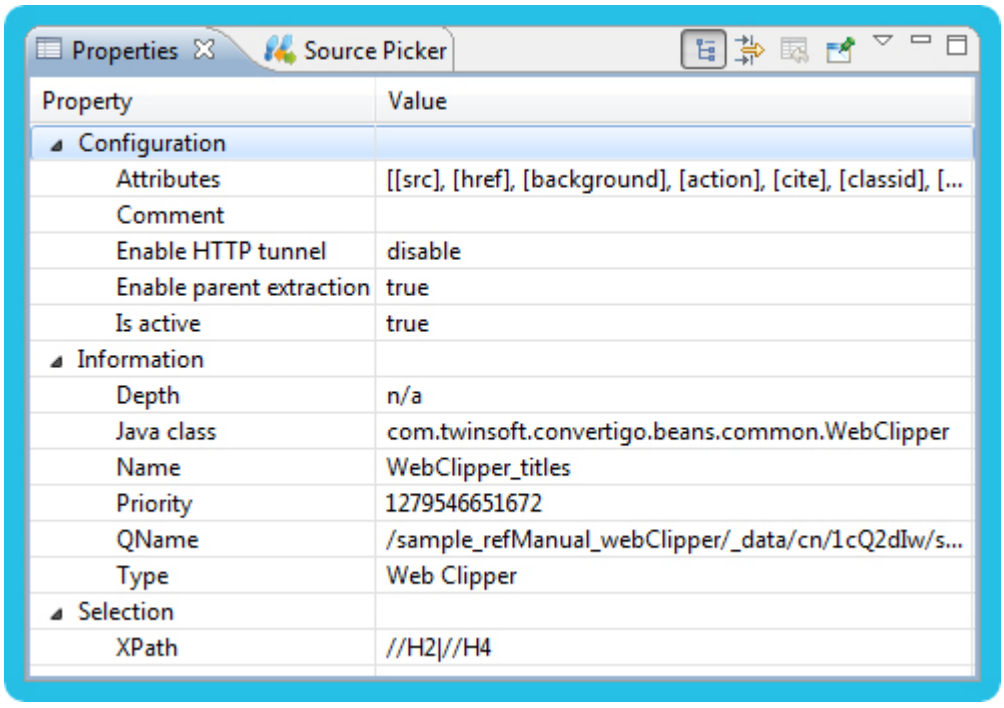


Figure 2 - 216: Web Clipper extraction rule - Example 1 - Configuration example

Attributes property is edited in the **Screen zone** editor:

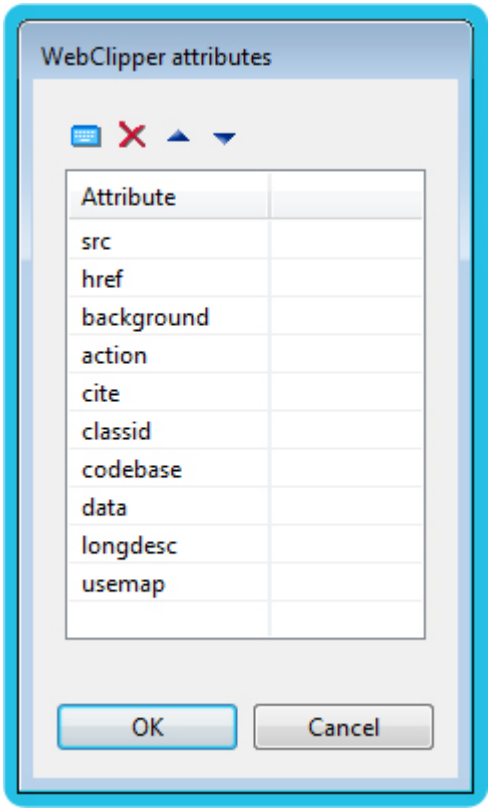


Figure 2 - 217: Web Clipper extraction rule - Example 1 - Attributes property edition

Once it is generated, the *Web Clipper* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

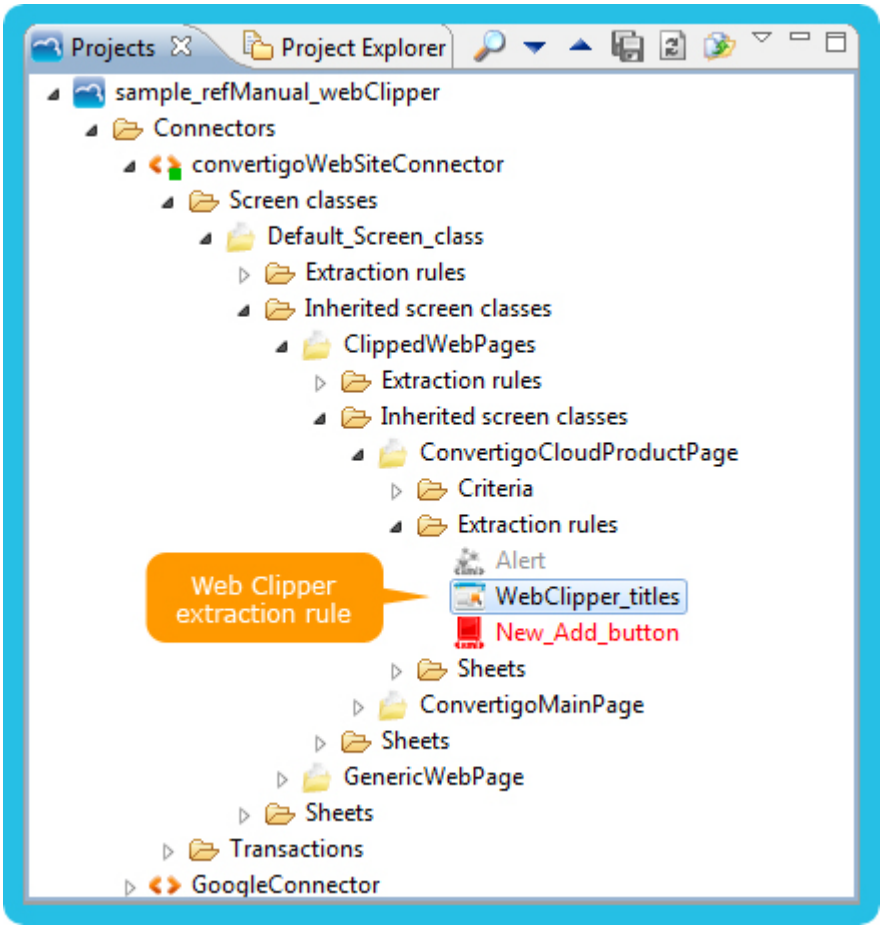


Figure 2 - 218: Web Clipper extraction rule - Example 1 - Object in Projects view

When the rule is applied, the resulting XML includes the HTML elements retrieved by the Xpath expression:

```
<document connector="refManualWebClippingSamplesConnector" context="studio_refManualWebClippingSamples:refManualWebClippingSamplesConnector"
  <HEAD twsid="14">


 <META content="text/html; charset=utf-8" http-equiv="content-type" twsid="15"/>
 <META content="index, follow" name="robots" twsid="16"/>
 <META content="Build mashups, test enterprise mashups, design mashups, C-EMS 5, Convertigo mashup, Convertigo cloud, Amazon EC2, mashables, Conv
 <META content="Convertigo Shared Cloud: Secure and Flexible" name="title" twsid="18"/>
 <META content="Administrator" name="author" twsid="19"/>
 <META content="The Convertigo Cloud is a new, flexible way to freely design, build &amp;amp; test enterprise mashups, quickly supporting new business ideas.
 <META content="Joomla! 1.5 - Open Source Content Management" name="generator" twsid="21"/>
 <TITLE twsid="22">Convertigo Shared Cloud: Secure and Flexible</TITLE>
 <LINK href="http://www.convertigo.com/templates/convertigotemplate_main/favicon.ico" rel="shortcut icon" twsid="23" type="image/x-icon"/>


<META content="text/css" http-equiv="Content-Style-Type" twsid="26"/>
<LINK href="http://www.convertigo.com/templates/convertigotemplate_main/css/template_css.css" rel="stylesheet" twsid="27" type="text/css"/>
<LINK href="http://www.convertigo.com/templates/convertigotemplate_main/css/subpages.css" rel="stylesheet" twsid="28" type="text/css"/>
<LINK href="http://www.convertigo.com/templates/convertigotemplate_main/css/styles.css" rel="stylesheet" twsid="29" type="text/css"/>
<LINK href="http://www.convertigo.com/templates/convertigotemplate_main/css/en/banners.css" rel="stylesheet" twsid="30" type="text/css"/>
<LINK href="http://www.convertigo.com/templates/convertigotemplate_main/css/en/menus.css" rel="stylesheet" twsid="31" type="text/css"/>
</HEAD>
  <H2 twsid="0">What business users are looking for?</H2>
  <H4 twsid="1">The shortest path from business ideas to enterprise mashups</H4>
  <H2 twsid="2">What IT organizations require?</H2>
  <H4 twsid="3">A robust and secured platform</H4>
  <H2 twsid="4">How to do more with less?</H2>
  <H4 twsid="5">IT governance and budget flexibility</H4>
  <H2 twsid="6">How does it work?</H2>
  <H2 twsid="7">Benefits</H2>
  <H2 twsid="8">For Developers</H2>
  <H4 twsid="9">No specific code to write</H4>
  <H2 twsid="10">For IT departments</H2>
  <H4 twsid="11">No new hardware or software to install</H4>
  <H2 twsid="12">For Business Users</H2>
  <H4 twsid="13">Reduced time-to-market</H4>
</document>
```

Figure 2 - 219: Web Clipper extraction rule - Example 1 - Resulting XML with rule

After XSL transformation, thanks to default web clipping project XSL style sheet, the clipped elements are displayed as follows (in a Web browser, possibly through the Test Platform):

Figure 2 - 220: Web Clipper extraction rule - Example 1 - Webized page with rule

We can notice that the presentation attributes (like font colors) seem not completely identical to the original website. Maybe the elements themselves don't have all style properties, but in the context of the full page they do have more style properties. For that reason, we are changing the **Enable parent extraction** property value to set it to `true`.

When the rule is applied, the resulting XML includes the HTML elements retrieved by the Xpath expression as well as their parent and ancestor elements:

```
<BODY id="sec6">
  <DIV id="wrapper">
    <DIV id="container">
      <DIV id="main-content-wrap">
        <DIV id="body-content">
          <DIV class="content-main">
            <TABLE class="contentpaneopen-products">
              <TBODY>
                <TR>
                  <TD valign="top">
                    <H2 twsid="0">What business users are looking for?</H2>
                    <H4 twsid="1">The shortest path from business ideas to enterprise mashups</H4>
                    <TABLE border="0" cellpadding="0" style="width: 100%;">
                      <TBODY>
                        <TR>
                          <TD class="table_left" valign="top">
                            <H2 twsid="2">What IT organizations require?</H2>
                            <H4 twsid="3">A robust and secured platform</H4>
                          </TD>
                        </TR>
                      </TBODY>
                    </TABLE>
                    <H2 twsid="4">How to do more with less?</H2>
                    <H4 twsid="5">IT governance and budget flexibility</H4>
                    <H2 twsid="6">How does it work?</H2>
                    <H2 twsid="7">Benefits</H2>
                    <TABLE border="0" cellpadding="10" style="width: 100%;">
                      <TBODY>
                        <TR>
                          <TD valign="top" width="495">
                            <H2 twsid="8">For Developers</H2>
                            <H4 twsid="9">No specific code to write</H4>
                            <H2 twsid="10">For IT departments</H2>
                            <H4 twsid="11">No new hardware or software to install</H4>
                            <H2 twsid="12">For Business Users</H2>
                            <H4 twsid="13">Reduced time-to-market</H4>
                          </TD>
                        </TR>
                      </TBODY>
                    </TABLE>
                  </TD>
                </TR>
              </TBODY>
            </TABLE>
          </DIV>
        </DIV>
      </DIV>
    </DIV>
  </DIV>
</BODY>
```

Figure 2 - 221: Web Clipper extraction rule - Example 1 - Resulting XML with rule (with parent extraction)

After XSL transformation, thanks to default web clipping project XSL style sheet, the clipped elements are displayed as they are in the original website:

Figure 2 - 222: Web Clipper extraction rule - Example 1 - Webized page with rule (with parent extraction)

Example 2

For the second example, let’s consider the following web page from Google search engine:

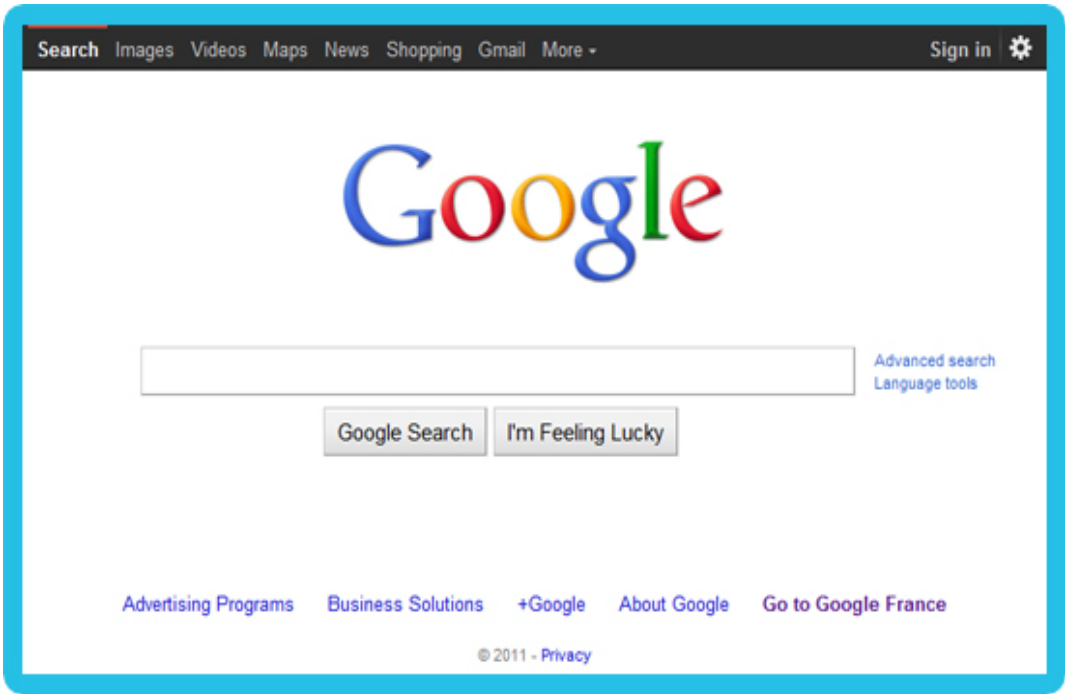


Figure 2 - 223: Web Clipper extraction rule - Example 2 - HTML web page

In this example, we want to extract the Google logo and search form thanks to a *Web Clipper* extraction rule, using Convertigo HTTP tunnel, in a new connector of the `sample_refManual_webClipper` project allowing to connect to this website.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

Once accessing this web page by opening the `GoogleConnector` connector editor, the Xpath matching both elements is generated using the **Xpath Evaluator**:

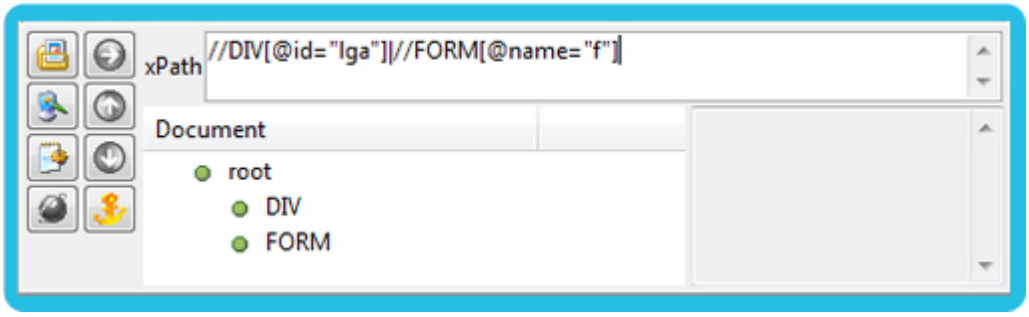


Figure 2 - 224: Web Clipper extraction rule - Example 2 - Generating Xpath in the Xpath Evaluator

Then the rule is created with the following parameters:

```
Web Clipper [
   xpath=//DIV[@id="lga"]|//FORM[@name="f"]
   attributes=[src, href, background, action, cite, classid, codebase,
   data, longdesc, usemap]
   enable HTTP tunnel=no cache
```

```
  enable parent extraction=true
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

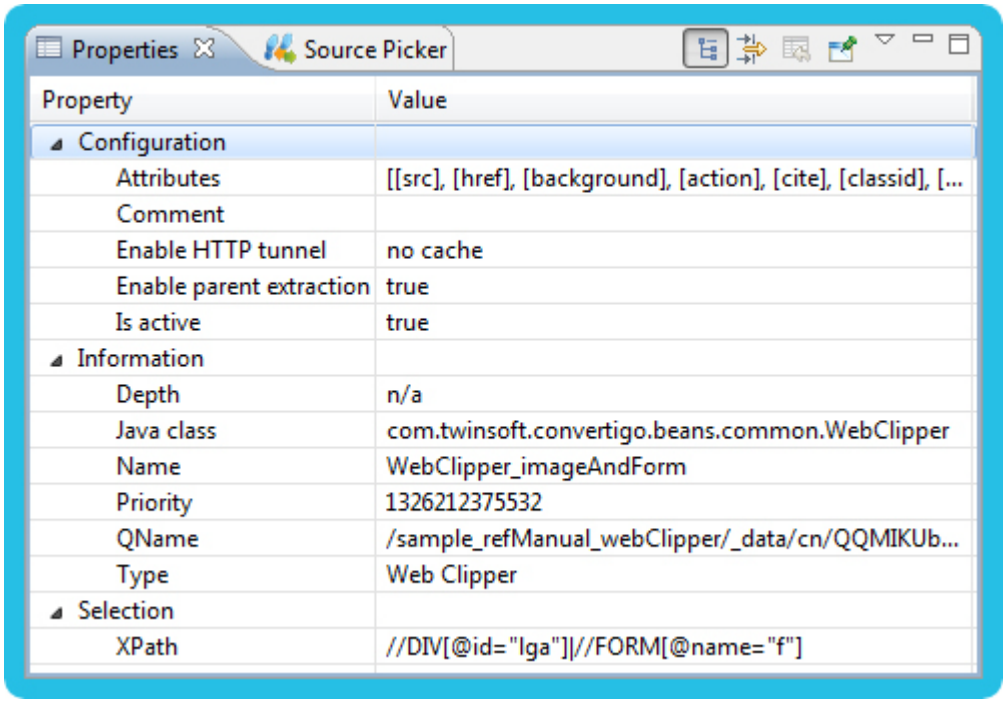


Figure 2 - 225: Web Clipper extraction rule - Example 2 - Configuration example

Once it is generated, the *Web Clipper* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

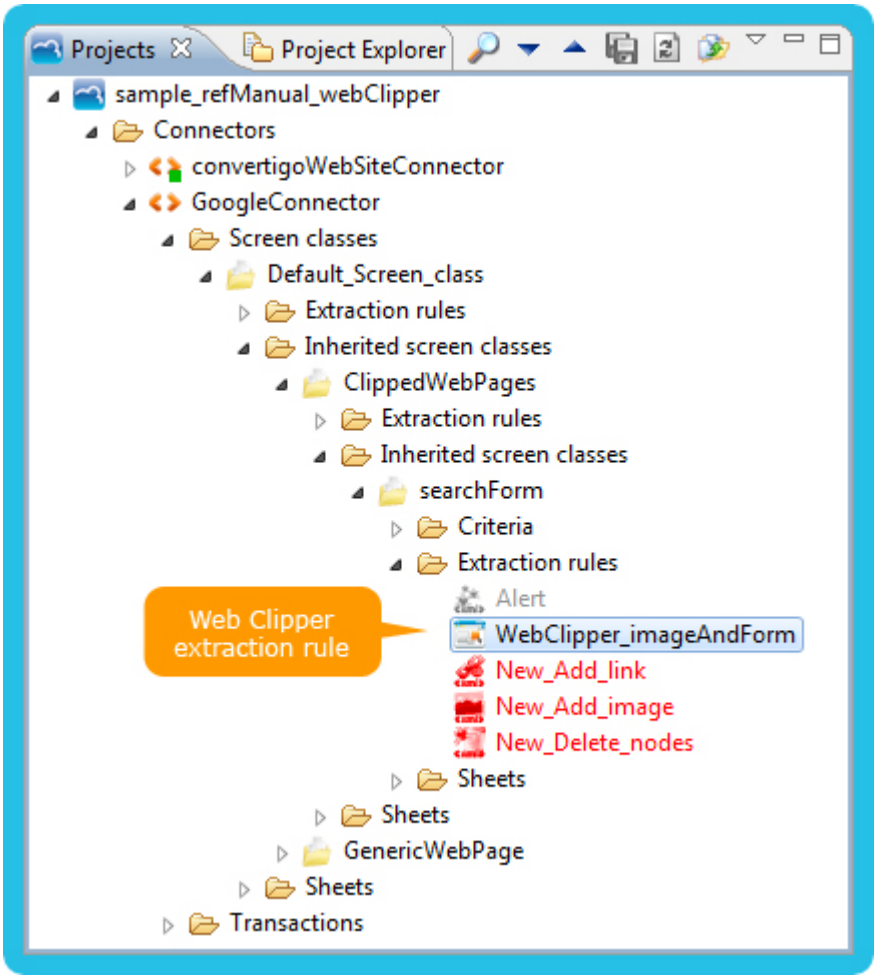


Figure 2 - 226: Web Clipper extraction rule - Example 2 - Object in Projects view

When the rule is executed, the resulting XML includes the HTML elements retrieved by the Xpath expression, with their parents and ancestors (thanks to the **Enable parent extraction** property set to `true`):

```
<document connector="refManualWebClippingSamplesConnector" context="studio_refManualWebClippingSam
  <HEAD twsid="29">
    <META content="text/html; charset=UTF-8" http-equiv="content-type" twsid="30"/>
    <TITLE twsid="31">Google</TITLE>
    <STYLE id="gstyle" twsid="33">body{margin:0}#gog{padding:3px 8px 0}td{line-height:.8em}.gac_m td{line-h
    <STYLE twsid="35" type="text/css">.gac_od{background:white;margin:0;z-index:99;border-top:0;border-left:
  </HEAD>
  <BODY alink="#ff0000" bgcolor="#ffffff" link="#0000cc" onload="document.f.q.focus();if(document.images)n
    <CENTER>
      <DIV id="lga" twsid="0">
        <IMG alt="Google" height="95" id="logo" src="http://localhost:18080/convertigo/webclipper/refManua
        <BR twsid="2"/>
        <BR twsid="3"/>
      </DIV>
      <FORM name="f" twsid="4">
        <TABLE cellpadding="0" cellspacing="0" twsid="5">
          <TBODY twsid="6">
            <TR twsid="7" valign="top">
              <TD twsid="8" width="25%"> </TD>
              <TD align="center" twsid="9">
                <INPUT name="hl" twsid="10" type="hidden" value="en"/>
                <INPUT name="source" twsid="11" type="hidden" value="hp"/>
                <DIV class="ds" style="margin: 4px 0pt; height: 32px;" twsid="12">
                  <INPUT autocomplete="off" class="lst" maxlength="2048" name="q" size="57" style="borde
                </DIV>
                <BR style="line-height: 0pt;" twsid="14"/>
                <SPAN class="ds" twsid="15">
                  <SPAN class="lsbb" twsid="16">
                    <INPUT class="lsb" name="btnG" twsid="17" type="submit" value="Google Search"/>
                  </SPAN>
                </SPAN>
                <SPAN class="ds" twsid="18">
                  <SPAN class="lsbb" twsid="19">
                    <INPUT class="lsb" name="btnI" twsid="20" type="submit" value="I'm Feeling Lucky"/>
                  </SPAN>
                </SPAN>
              </TD>
              <TD align="left" class="sblc" twsid="21" width="25%">
                <A href="#" original_url="/advanced_search?hl=en" twsid="22">Advanced Search</A>
                <A href="#" original_url="/language_tools?hl=en" twsid="23">Language Tools</A>
              </TD>
            </TR>
          </TBODY>
        </TABLE>
        <INPUT name="aq" twsid="24" type="hidden" value="f"/>
        <INPUT name="aqi" twsid="25" type="hidden"/>
        <INPUT name="aql" twsid="26" type="hidden"/>
        <INPUT name="oq" twsid="27" type="hidden"/>
        <INPUT name="gs_rfai" twsid="28" type="hidden"/>
      </FORM>
    </CENTER>
  </BODY>
</document>
```

Figure 2 - 227: Web Clipper extraction rule - Example 2 - Resulting XML with rule

After XSL transformation, thanks to default web clipping project XSL style sheet, the clipped elements are displayed as follows (in a Web browser, possibly through the Test Platform):

Figure 2 - 228: Web Clipper extraction rule - Example 2 - Webized page with rule

The image and styles have been correctly got through Convertigo HTTP tunnel (thanks to the **Enable HTTP tunnel** property set to `no cache`).

ADD LINK 

OBJECT DESCRIPTION

Adds an HTML link under a node.

The *Add link* extraction rule adds an HTML link (A tag with attributes and content) in the XHTML content clipped by a *Web Clipper* extraction rule.

It is part of the set of **Web Clipping** extraction rules adding content into the XHTML content of a previously executed *Web Clipper* extraction rule, such as:

- *Add image*,
- *Add text*,
- *Add button*.

Such extraction rules change the XHTML content resulting from the execution of a *Web Clipper* extraction rule, by adding specific XHTML content based on defined parameters.

Notes:

- The **XPath** property set for such extraction rules must be valid in the XHTML output resulting from the *Web Clipper* extraction rule and possibly modified by previous rules.
- If a list of nodes are matching the defined Xpath, the *Add link* extraction rule will only add one link under the first matching node.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Href	JS expression	configuration	Defines the content of the `href` attribute. The `href` attribute allows to define the link destination, i.e. which web page this link will reach.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Target new window	boolean	configuration	Defines whether the link should open in a new window. If set to `true`, sets the tag `target` attribute to `target="_blank"`.
Text	String	configuration	Defines the text content to add. Depending on the extraction rule, it is: • the added text, for *Add text* rule, • the link displayed text, for *Add link rule*, • the added image label and alternative text, for *Add image* and *Add button* rules.

Property	Type	Category	Description
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the second example of *Web clipper* extraction rule, which clips Google logo and search form on Google search engine page, thanks to objects created in the `sample_refManual_webClipper` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

After the application of the *Web clipper* extraction rule and XSL transformation thanks to default XSL style sheet, the clipped elements are displayed as follows:

Figure 2 - 229: Web Clipper extraction rule - Example 2 - Clipped web page

In this example, we want to add a link on the clipped web page under the two already present links on the right of the page, thanks to an *Add link* extraction rule.

The Xpath matching the links container element in clipped XHTML is generated using the **Xpath Evaluator** of the Convertigo Studio:

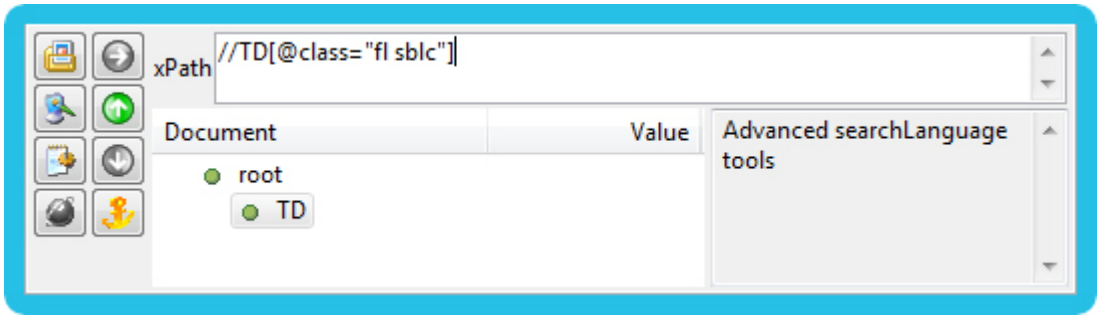


Figure 2 - 230: Add link extraction rule - Generating Xpath in the Xpath Evaluator

This Xpath matches the `TD` element containing both links for the new link to be added under them.

The rule is created with the following parameters:

```
Add link [
  xpath=//TD[@class="fl sblc"]
  href="http://www.convertigo.com/
  target new window=true
  text="Open Convertigo web site"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

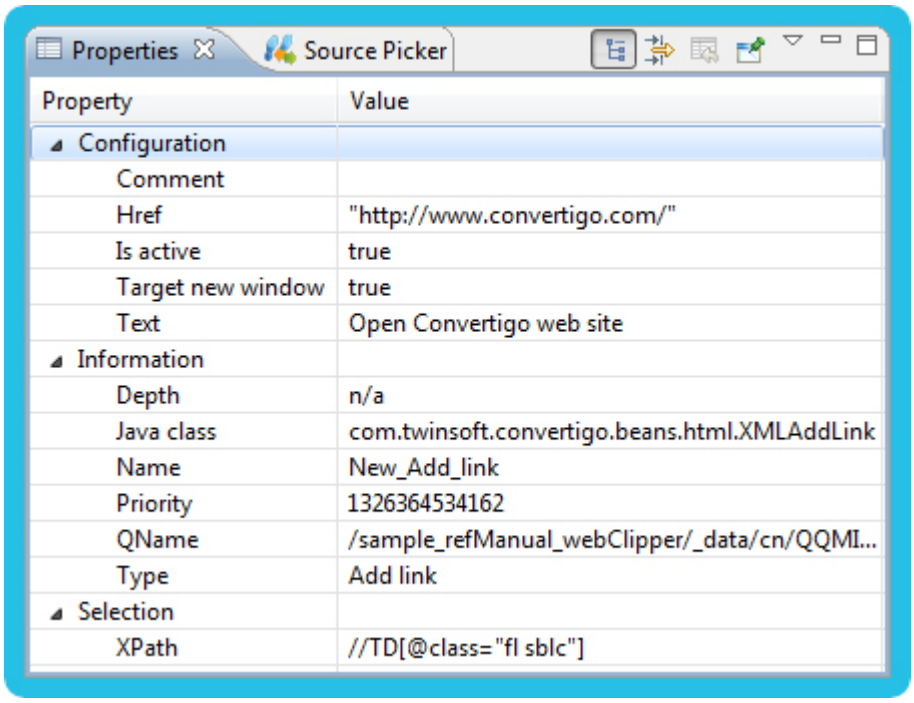


Figure 2 - 231: Add link extraction rule - Configuration example

The **Target new window** property is set to `true` so that the link will open the reached web page in a new tab or new window (depending on the browser).

Once it is generated, the *Add link* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

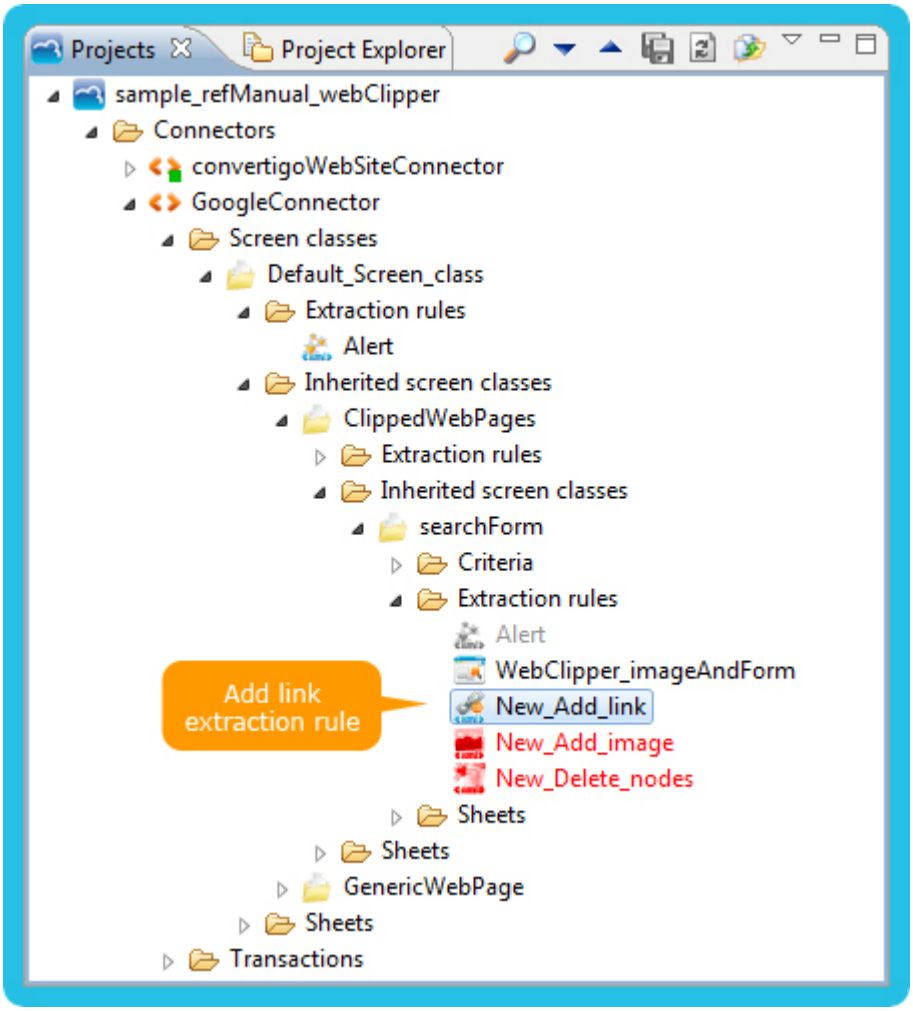


Figure 2 - 232: Add link extraction rule - Object in Projects view

When the rule is executed, the resulting XML includes the HTML elements retrieved by the *Web clipper* extraction rule, plus the added link:

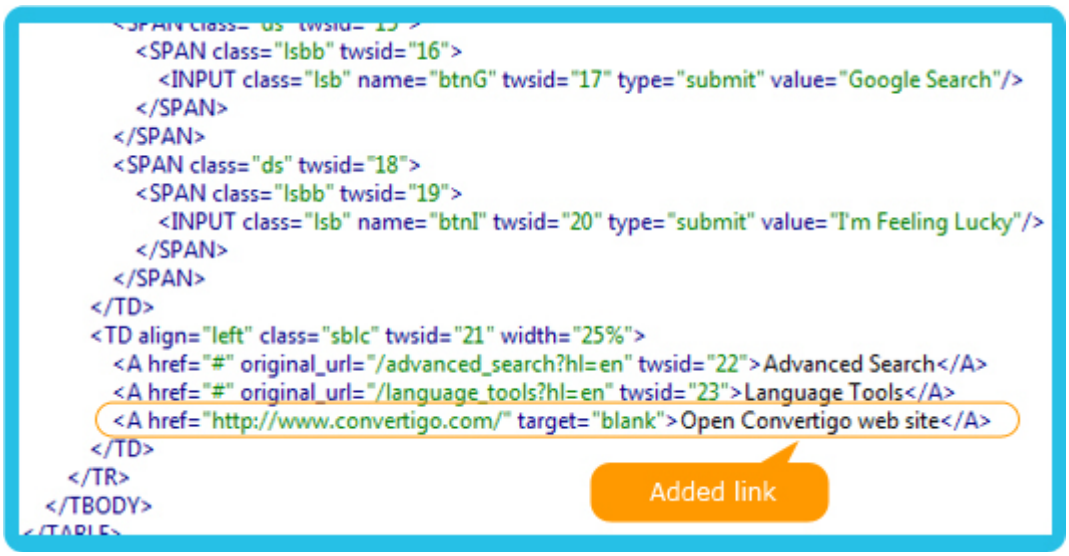


Figure 2 - 233: Add link extraction rule - Resulting XML with rule

After XSL transformation, thanks to default web clipping project XSL style sheet, the link is displayed under the form clipped elements as follows:

Figure 2 - 234: Add link extraction rule - Webized page with rule

We can see the link text corresponding to defined **Text** property. When clicking on the link, a new window (or tab, depending on the browser) opens with Convertigo website page defined in **Href** property.

ADD BUTTON 

OBJECT DESCRIPTION

Adds an HTML button under a node.

The *Add button* extraction rule adds an HTML button, that is to say an image element (`IMG` tag) associated with a link (`A` tag), in the XHTML content clipped by a *Web Clipper* extraction rule.

It is part of the set of **Web Clipping** extraction rules adding content into the XHTML content of a previously executed *Web Clipper* extraction rule, such as:

- *Add link*,
- *Add text*,
- *Add image*.

Such extraction rules change the XHTML content resulting from the execution of a *Web Clipper* extraction rule, by adding specific XHTML content based on defined parameters.

Notes:

- The **XPath** property set for such extraction rules must be valid in the XHTML output resulting from the *Web Clipper* extraction rule and possibly modified by previous rules.
- If a list of nodes are matching the defined Xpath, the *Add button* extraction rule will only add one button under the first matching node.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Href	JS expression	configuration	Defines the content of the `href` attribute. The `href` attribute allows to define the link destination, i.e. which web page this link will reach.
Image URL	String	configuration	Defines the button image URL. The image URL can be either absolute or relative to the project.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Target new window	boolean	configuration	Defines whether the link should open in a new window. If set to `true`, sets the tag `target` attribute to `target="_blank"`.

Property	Type	Category	Description
Text	String	configuration	Defines the text content to add. Depending on the extraction rule, it is: • the added text, for *Add text* rule, • the link displayed text, for *Add link rule*, • the added image label and alternative text, for *Add image* and *Add button* rules.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the first example of *Web clipper* extraction rule, which clips titles and subtitles on "The Product Lineup, Convertigo Shared Cloud" page from Convertigo website, in a project named `sample_refManual_webClipper`, thanks to objects created in the `sample_refManual_webClipper` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

After the application of the *Web clipper* extraction rule and XSL transformation thanks to default XSL style sheet, the clipped elements are displayed as follows:

Figure 2 - 235: Web Clipper extraction rule - Example 1 - Clipped web page

In this example, we want to add a button on the clipped web page next to the first title, thanks to an *Add button* extraction rule.

The Xpath matching all title elements in clipped XHTML is generated using the **Xpath Evaluator** of the Convertigo Studio:

Figure 2 - 236: Add button extraction rule - Generating Xpath in the Xpath Evaluator

This Xpath can match on all title elements, the rule will only add one button on the first

matching node.

The rule is created with the following parameters:

```
Add button [
  xpath=//H2
  href="http://www.convertigo.com/en/overview/convertigo-cloud.html"
  image URL="img/button_img.png"
  target new window=true
  text="open original page"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

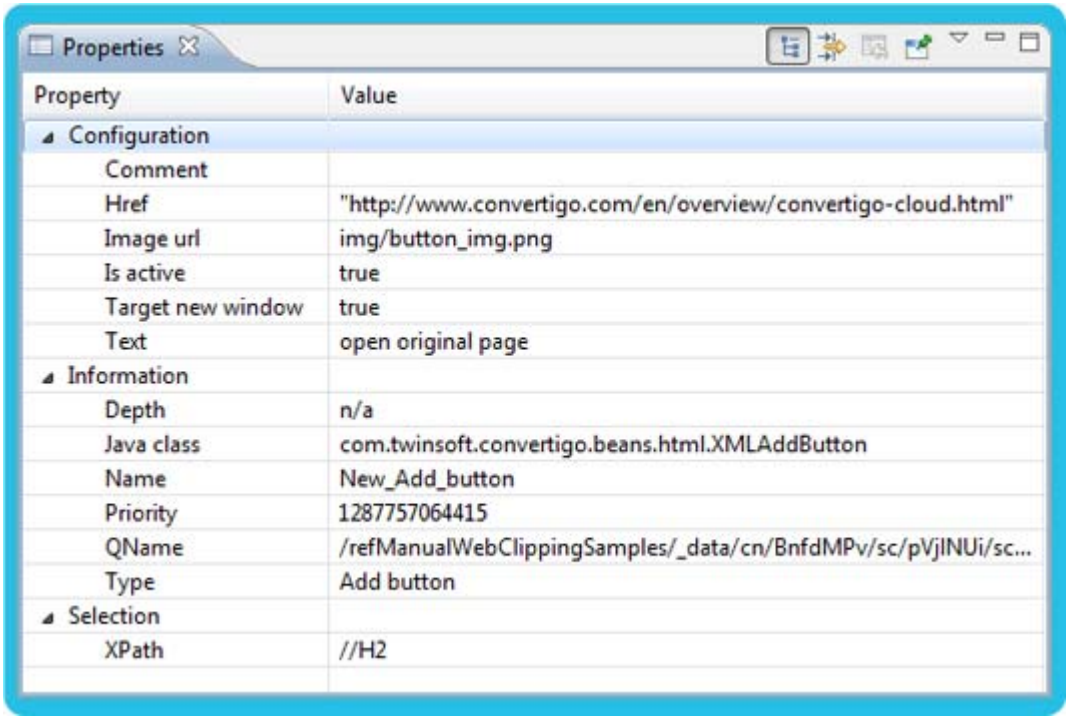


Figure 2 - 237: Add button extraction rule - Configuration example

The **image URL** property is relative to the project directory. It is set with a parent "img/" path, meaning that the image file has to be stored in an `img` folder at the root of the project directory:

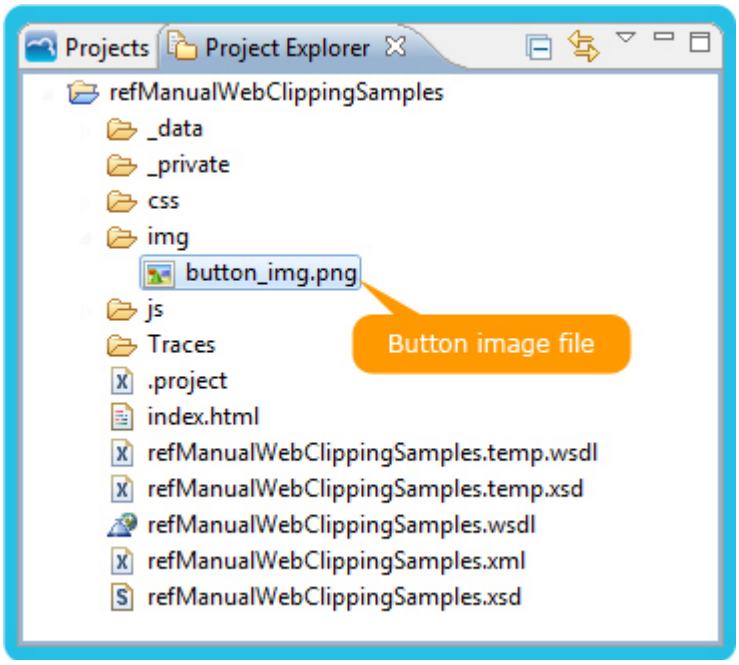


Figure 2 - 238: Add button extraction rule - Storing image file on "img" folder

The **Target new window** property is set to `true` so that the link will open the reached web page in a new tab or new window (depending on the browser).

Once it is generated, the *Add button* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

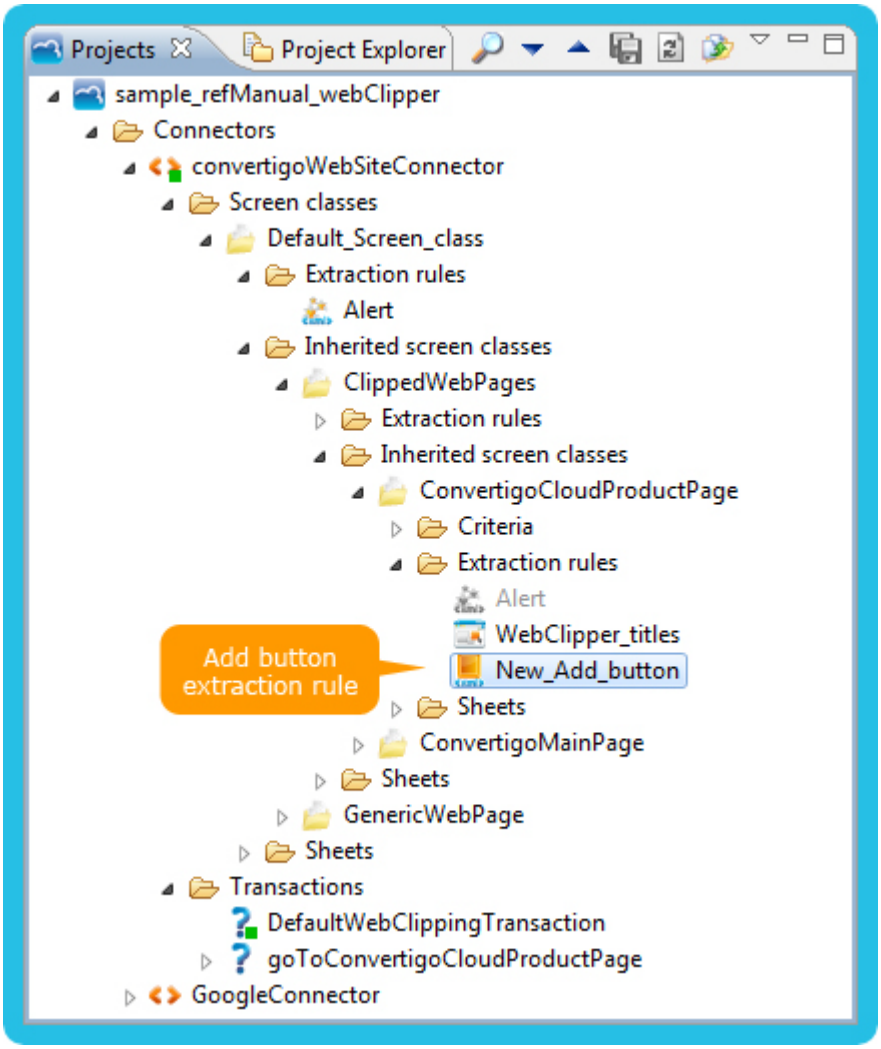


Figure 2 - 239: Add button extraction rule - Object in Projects view

When the rule is executed, the resulting XML includes the HTML elements retrieved by the *Web clipper* extraction rule, plus the added image button:

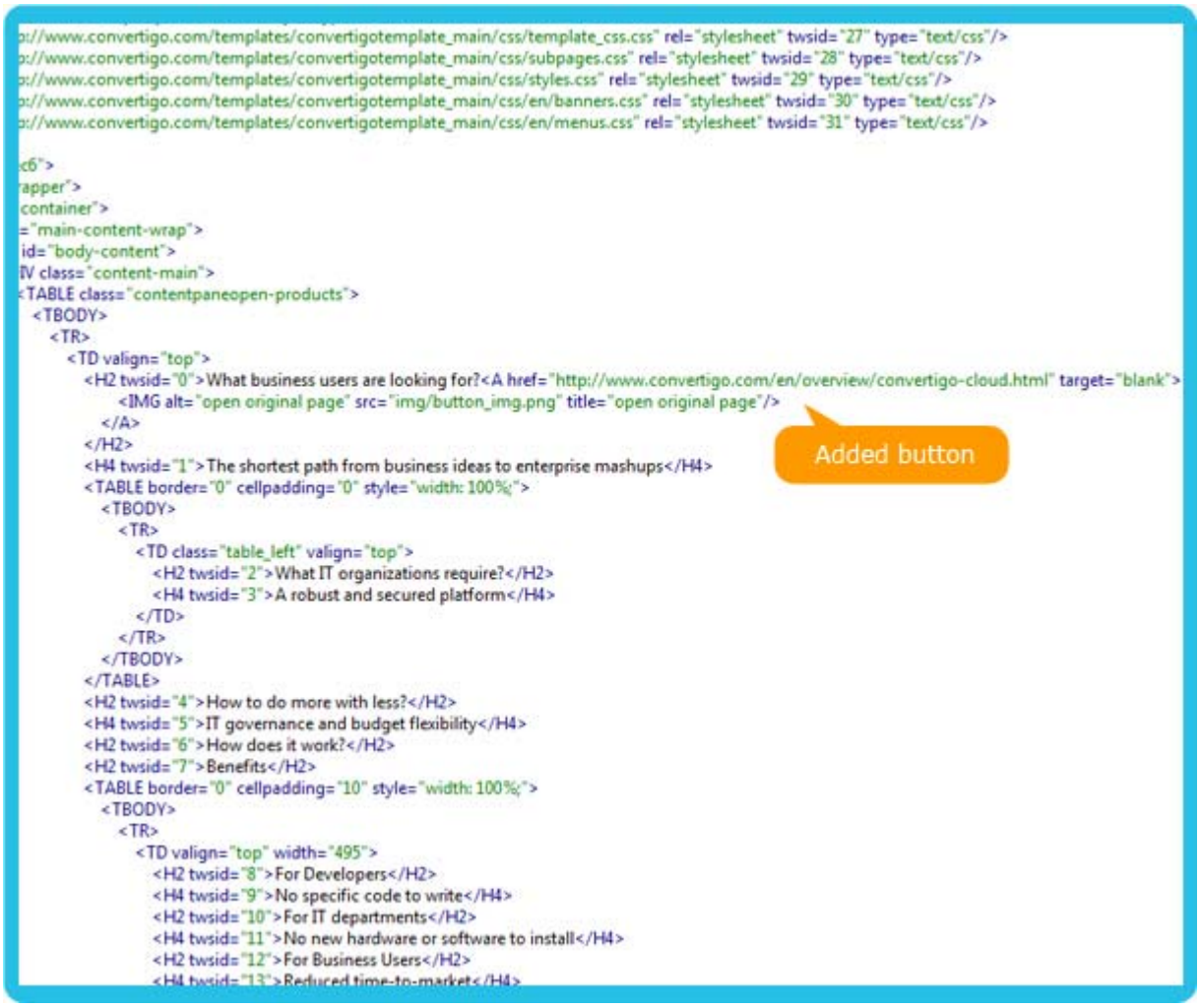


Figure 2 - 240: Add button extraction rule - Resulting XML with rule

After XSL transformation, thanks to default web clipping project XSL style sheet, the button is displayed next to clipped elements as follows:

Figure 2 - 241: Add button extraction rule - Webized page with rule

We can see the tooltip containing text defined in **Text** property. When clicking on the button, a new window (or tab, depending on the browser) opens with Convertigo website page defined in **Href** property.

ADD TEXT 

OBJECT DESCRIPTION

Adds HTML text under a node.

The *Add text* extraction rule adds an HTML text element (P tag with text content) in the XHTML content clipped by a *Web Clipper* extraction rule.

It is part of the set of **Web Clipping** extraction rules adding content into the XHTML content of a previously executed *Web Clipper* extraction rule, such as:

- *Add link*,
- *Add Image*,
- *Add button*.

Such extraction rules change the XHTML content resulting from the execution of a *Web Clipper* extraction rule, by adding specific XHTML content based on defined parameters.

Notes:

- The **XPath** property set for such extraction rules must be valid in the XHTML output resulting from the *Web Clipper* extraction rule and possibly modified by previous rules.
- If a list of nodes are matching the defined Xpath, the *Add text* extraction rule will only add one text under the first matching node.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Text	String	configuration	Defines the text content to add. Depending on the extraction rule, it is: • the added text, for *Add text* rule, • the link displayed text, for *Add link rule*, • the added image label and alternative text, for *Add image* and *Add button* rules.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

ADD IMAGE 

OBJECT DESCRIPTION

Adds an HTML image under a node.

The *Add image* extraction rule adds an HTML image element (`IMG` tag with attributes) in the XHTML content clipped by a *Web Clipper* extraction rule.

It is part of the set of **Web Clipping** extraction rules adding content into the XHTML content of a previously executed *Web Clipper* extraction rule, such as:

- *Add link*,
- *Add text*,
- *Add button*.

Such extraction rules change the XHTML content resulting from the execution of a *Web Clipper* extraction rule, by adding specific XHTML content based on defined parameters.

Notes:

- The **XPath** property set for such extraction rules must be valid in the XHTML output resulting from the *Web Clipper* extraction rule and possibly modified by previous rules.
- If a list of nodes are matching the defined Xpath, the *Add image* extraction rule will only add one image under the first matching node.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Image URL	String	configuration	Defines the image URL. The image URL can be either absolute or relative to the project.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Text	String	configuration	Defines the text content to add. Depending on the extraction rule, it is: • the added text, for *Add text* rule, • the link displayed text, for *Add link rule*, • the added image label and alternative text, for *Add image* and *Add button* rules.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the second example of *Web clipper* extraction rule, which clips Google logo and search form on Google search engine page, thanks to objects created in the `sample_refManual_webClipper` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

After the application of the *Web clipper* extraction rule and XSL transformation thanks to default XSL style sheet, the clipped elements are displayed as follows:

Figure 2 - 242: Web Clipper extraction rule - Example 2 - Clipped web page

In this example, we want to add an image on the clipped web page under the search form buttons, thanks to an *Add image* extraction rule.

The Xpath matching the form element in clipped XHTML is generated using the **Xpath Evaluator** of the Convertigo Studio:

Figure 2 - 243: Add image extraction rule - Generating Xpath in the Xpath Evaluator

This Xpath matches the `FORM` element for the image to be added at the end of the form, after buttons.

The rule is created with the following parameters:

```
Add image [
   xpath=//FORM[@name="f"]
   image URL="logo-convertigo.png"
   text="Convertigo"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

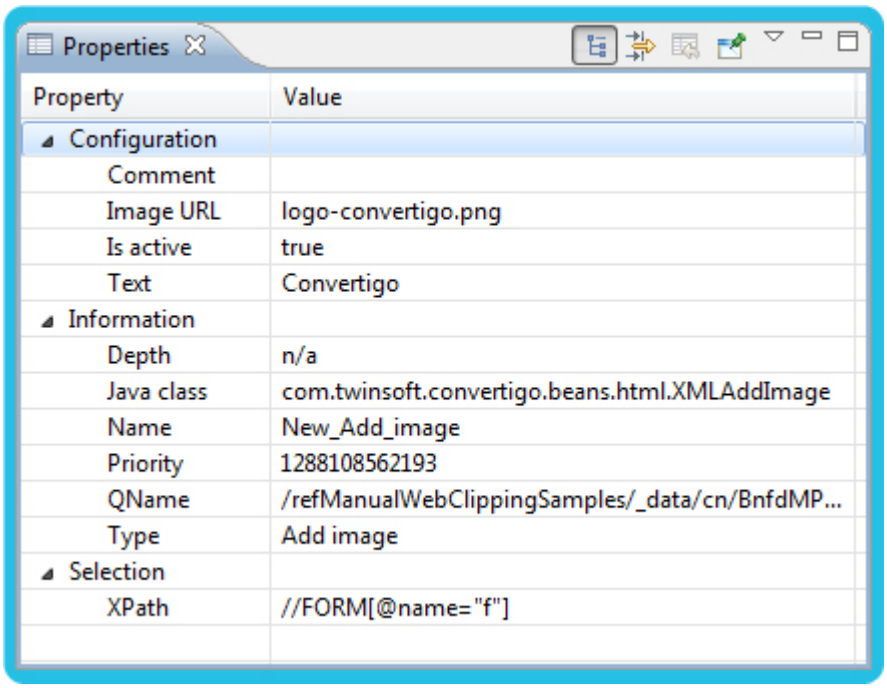


Figure 2 - 244: Add image extraction rule - Configuration example

The **image URL** property is relative to the project directory. It is set with no parent path, meaning that the image file has to be stored at the root of the project directory:

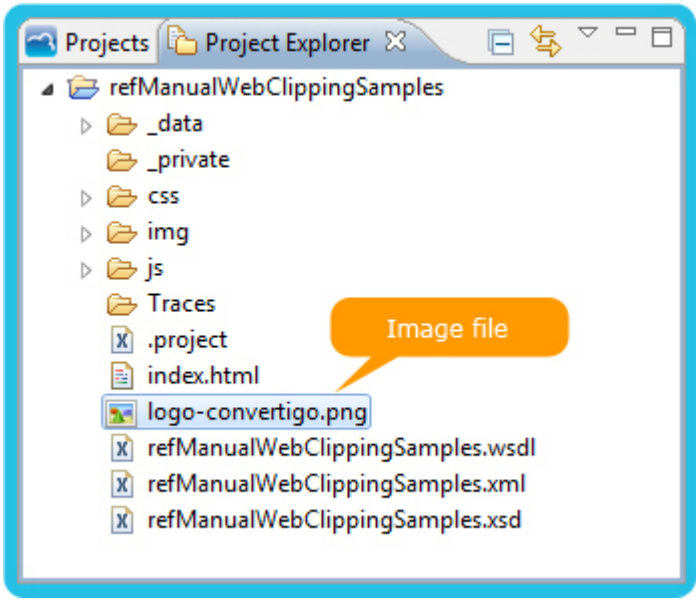


Figure 2 - 245: Add image extraction rule - Storing image file in the project directory

Once it is generated, the *Add image* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

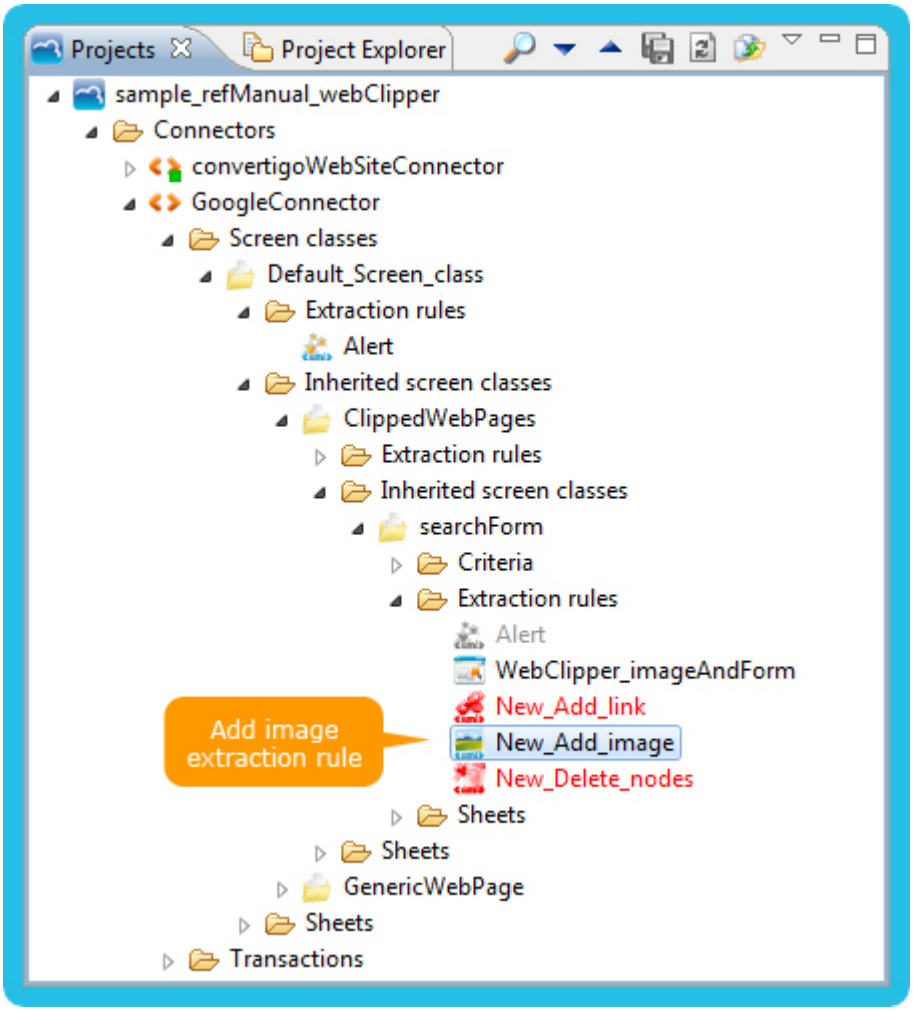


Figure 2 - 246: Add image extraction rule - Object in Projects view

When the rule is executed, the resulting XML includes the HTML elements retrieved by the *Web clipper* extraction rule, plus the added image:

Figure 2 - 247: Add image extraction rule - Resulting XML with rule

After XSL transformation, thanks to default web clipping project XSL style sheet, the image is displayed under the form clipped elements as follows:

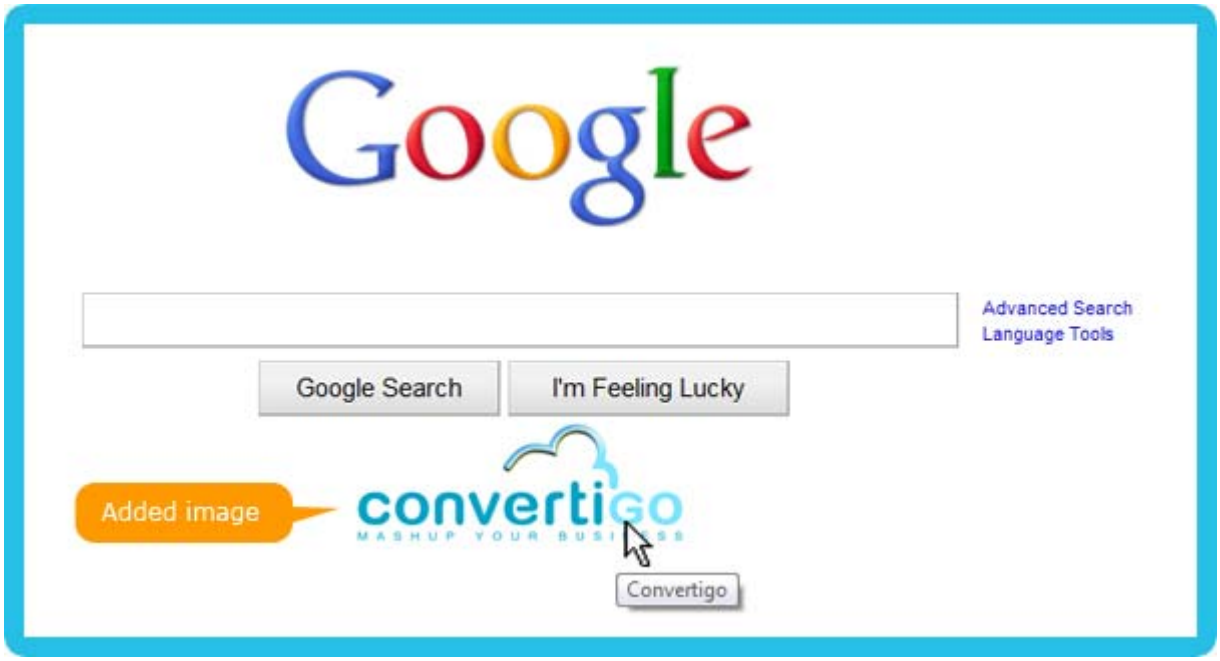


Figure 2 - 248: Add image extraction rule - Webized page with rule

We can see the tooltip containing text defined in **Text** property.

DELETE NODES 

OBJECT DESCRIPTION

Removes identified nodes from the XML output document.

The *Delete nodes* extraction rule deletes all nodes matching the Xpath defined in the **XPath** property set for the rule. Nodes are deleted from the XML output generated by a previously executed *Web Clipper* extraction rule.

When setting a *Web Clipper* extraction rule, it is sometimes impossible to avoid clipping unwanted elements. This rule allows to delete them following their extraction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the second example of *Web clipper* extraction rule, which clips Google logo and search form on Google search engine page, thanks to objects created in the `sample_refManual_webClipper` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***Web Clipper objects examples*** *in the* ***New Project*** *wizard.*

After the application of the *Web clipper* extraction rule and XSL transformation thanks to default XSL style sheet, the clipped elements are displayed as follows:

Figure 2 - 249: Web Clipper extraction rule - Example 2 - Clipped web page

In this example, we want to remove the links on the right of the page from the clipped web page, thanks to a *Delete nodes* extraction rule. We can identify these elements in the previous clipped page XHTML:

Figure 2 - 250: Web Clipper extraction rule - Example 2 - Resulting XML

The Xpath matching the links elements in clipped XHTML is generated using the **Xpath Evaluator** of the Convertigo Studio:

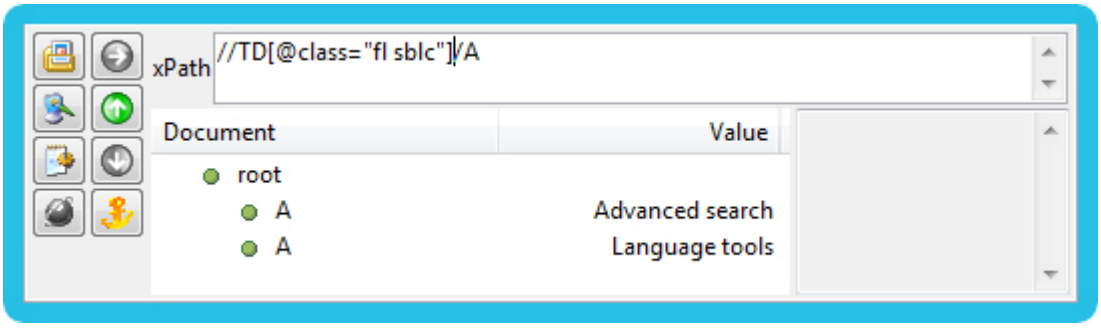


Figure 2 - 251: Delete nodes extraction rule - Generating Xpath in the Xpath Evaluator

This Xpath matches the A elements corresponding to every link, it was chosen for all links to be removed.

The rule is created with the following parameter:

```
Delete nodes [
  xpath=//TD[@class="fl sblc"]/A
]
```

This parameter is edited in the **Properties** view of the Convertigo Studio:

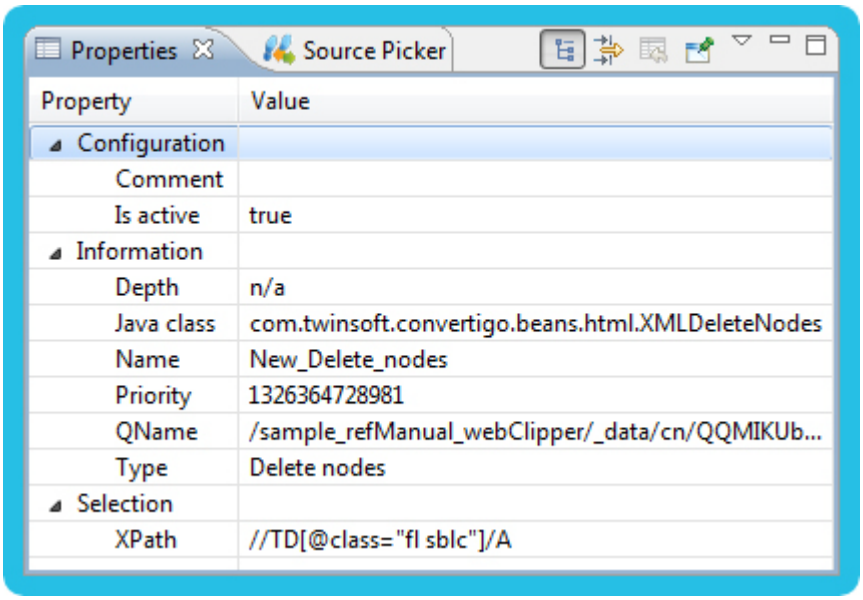


Figure 2 - 252: Delete nodes extraction rule - Configuration example

Once it is generated, the *Delete nodes* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

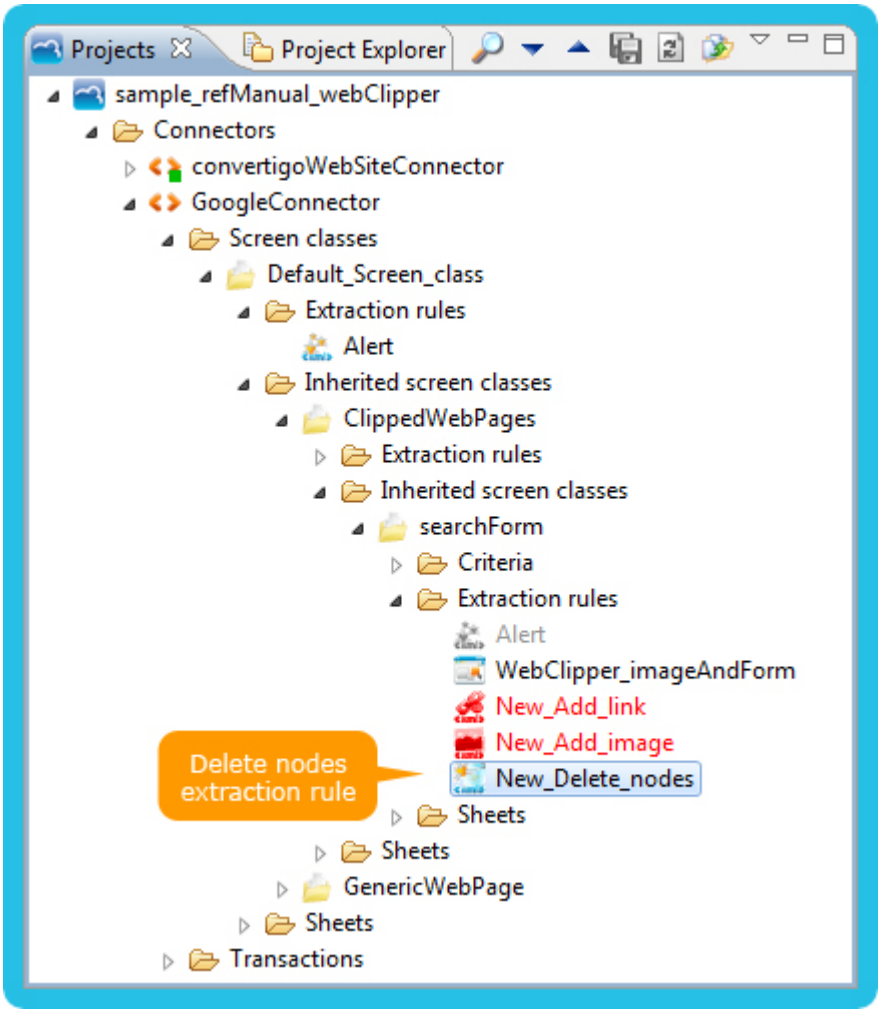


Figure 2 - 253: Delete nodes extraction rule - Object in Projects view

When the rule is executed, the resulting XML includes the HTML elements retrieved by the *Web clipper* extraction rule, unless deleted elements:

Figure 2 - 254: Delete nodes extraction rule - Resulting XML with rule

After XSL transformation, thanks to default web clipping project XSL style sheet, the clipped elements are displayed as follows:

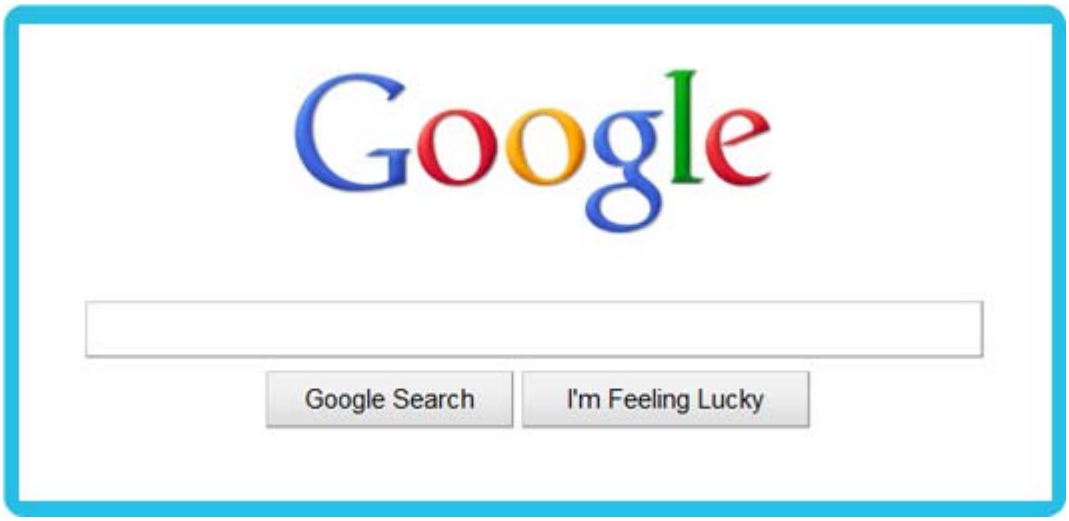


Figure 2 - 255: Delete nodes extraction rule - Webized page with rule

DATA EXTRACTION RULES

NODE

OBJECT DESCRIPTION

Extracts an XML node from a web page.

The *Node* extraction rule extracts an XML node from a web page. It is applied if the result of the Xpath expression evaluation exists into the HTML page DOM.

The first node matching the Xpath expression is extracted: the element is copied then appended to the HTML transaction DOM.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the following web page from Google search engine:

Figure 2 - 256: Node extraction rule - HTML web page

In this example, we want to extract the first node corresponding to available tabs (from the top of the page) thanks to a *Node* extraction rule.

The Xpath is generated using the **Xpath Evaluator** of the Convertigo Studio:

Figure 2 - 257: Node extraction rule - Generating Xpath in the Xpath Evaluator

Then the rule is created with the following parameters:

```
Node [
   xpath=//A[@class="gb1"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

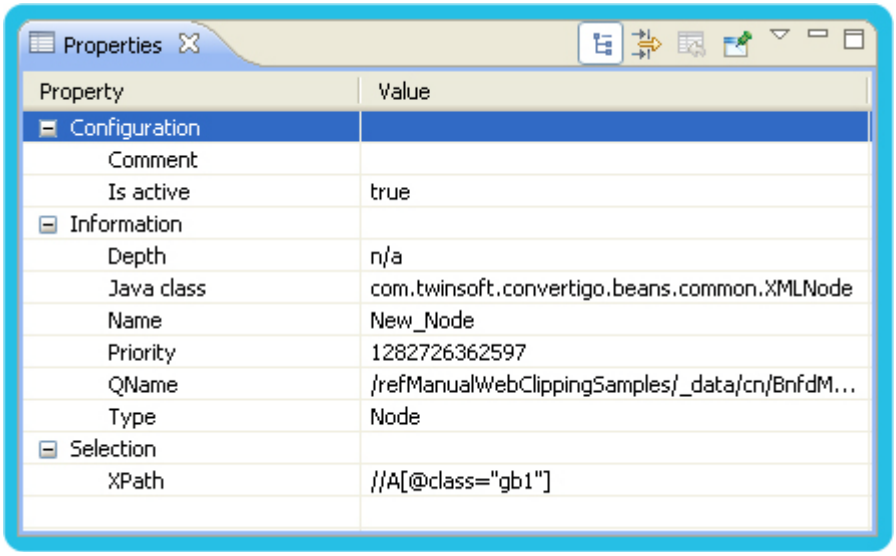


Figure 2 - 258: Node extraction rule - Configuration example

When the rule is executed, the resulting XML includes the first XML node retrieved by the Xpath expression:

```
<document connector="refManualWebClippingSamplesConnector" context="studio_refManualWebClippingSamples
   <A class="gb1" href="http://www.google.fr/imghp?hl=fr&amp;tab=wi" onclick="gbar.qs(this)">Images</A>
</document>
```

Figure 2 - 259: Node extraction rule - Resulting XML with rule

NODE LIST

OBJECT DESCRIPTION

Extracts an XML node list from a web page.

The *Node list* extraction rule extracts a list of XML nodes from a web page. It is applied if the result of the Xpath expression evaluation exists into the HTML page DOM.

The nodes matching the Xpath expression are extracted: the elements are copied then appended to the HTML transaction DOM.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the following web page from Google search engine:

Figure 2 - 260: Node list extraction rule - HTML web page

In this example, we want to extract the nodes corresponding to available tabs (from the top of the page) thanks to a *Node list* extraction rule.

The Xpath is generated using the **Xpath Evaluator** of the Convertigo Studio:

Figure 2 - 261: Node list extraction rule - Generating Xpath in the Xpath Evaluator

Then the rule is created with the following parameters:

```
Node list [
   xpath=//A[@class="gb1"]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

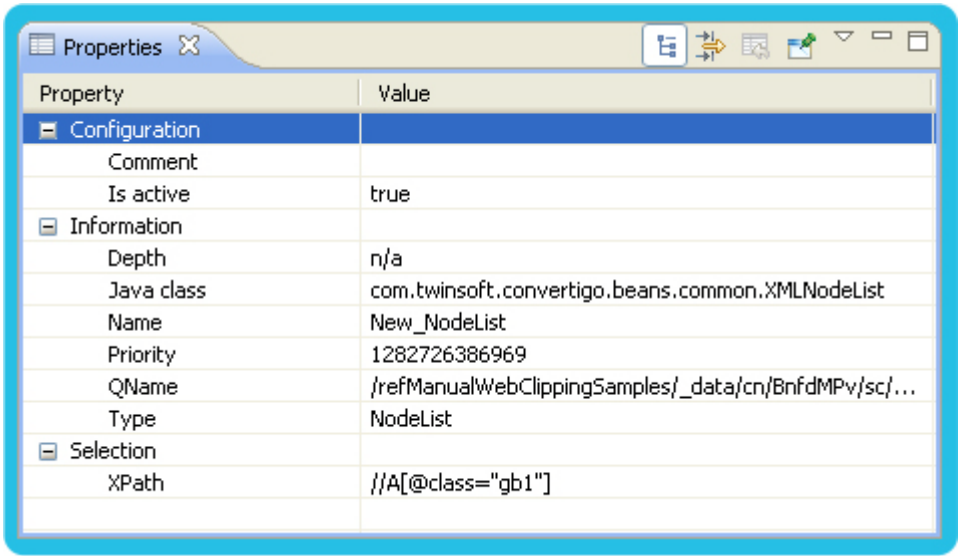


Figure 2 - 262: Node list extraction rule - Configuration example

When the rule is executed, the resulting XML includes the XML nodes retrieved by the Xpath expression:

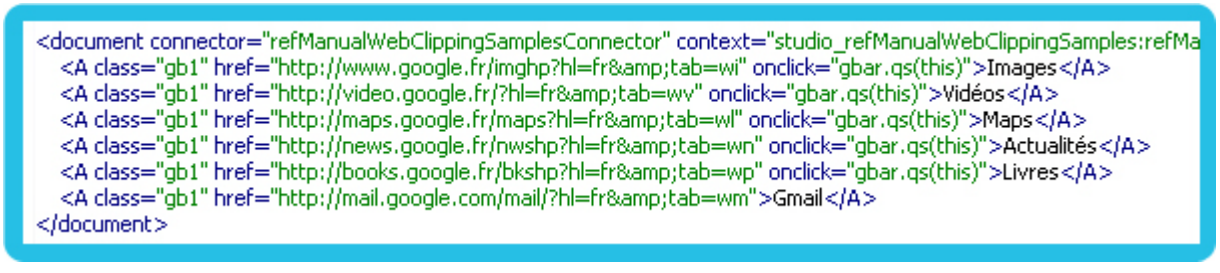


Figure 2 - 263: Node list extraction rule - Resulting XML with rule

RECORD (WEB) 

OBJECT DESCRIPTION

Extracts data from a web page in an XML record.

The *Record* extraction rule helps you extract a set of data from HTML text parts with identical and recurring presentation in a web page.

Extracted data are organized into a simple XML structure made of:

- a parent base element "corresponding to" the base recurring HTML elements containing data to extract, e.g a `<RECORD>`
- child elements "corresponding to" HTML text parts containing data, e.g `<DATAT1>, <DATAT2>, etc.`

The rule is applied if the result of the record Xpath expression evaluation exists in the HTML page DOM.

The resulting record elements are appended to the HTML transaction output DOM as follows:

```
<record_tagname referer="referer_url">
  <data1_tagname>extracted text from data1 xpath</data1_tagname>
  <data2_tagname>extracted text from data2 xpath</data2_tagname>
</record_tagname>
<record_tagname referer="referer_url">
  <data1_tagname>extracted text from data1 xpath</data1_tagname>
  <data2_tagname>extracted text from data2 xpath</data2_tagname>
</record_tagname>
```

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Description	XMLVector	selection	Describes how to extract data into record child text elements. The record is structured as a recurring element containing data, which are defined through **Description** property. This property is a list of child elements descriptions, also named columns descriptions. Each column description is composed of the following fields: • Name: Tag name of the child element (the default name is `data`). • Extract children: Indicates whether text extraction should recurse on child elements of the elements found thanks to the Xpath (by default it is set to `false`). As it needs more CPU if set to "true", it is then recommended to customize your XPath (using `//text()` function for example). • XPath: XPath expression selecting child element data. It is often defined relatively to parent *Record* extraction rule Xpath expression using the following syntax: `./`.
Display referer	boolean	configuration	Defines whether the referer URL is displayed in the output XML element. If this property is set to `true`, the referer URL is added as an attribute, named `referer`, to the XML element added by the extraction rule.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Tag name	String	configuration	Defines the record tag name in resulting DOM (default tag name is `XMLRecord`).
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

If we consider the following web page from Google search engine:

Figure 2 - 264: Record extraction rule - Google results web page

We can notice that Google results are recurrent elements on this Web page. Using a *Record* extraction rule, we want to extract two pieces of data for each Google result: the title and the URL pointed by the link.

This extraction rule will extract sets of data from this web page by:

- extracting each Google result title by concatenating the texts contained in the title HTML element with the texts contained in its child HTML elements,
- and inserting this text into a `title` element,
- extracting the text contained in the hypertext link on each Google title result,
- and inserting it into an `url` element.

The *Record* extraction rule is created with the following parameters:

```
Record [
   xpath="//LI[@class="g"]"
   tagname="result"
   display referer=false
   description={
      column [name="title" extract children=true xpath="./H3"],
      column [name="url" extract children=false
      xpath="/H3/A[@class="l"]/@href"]
   }
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

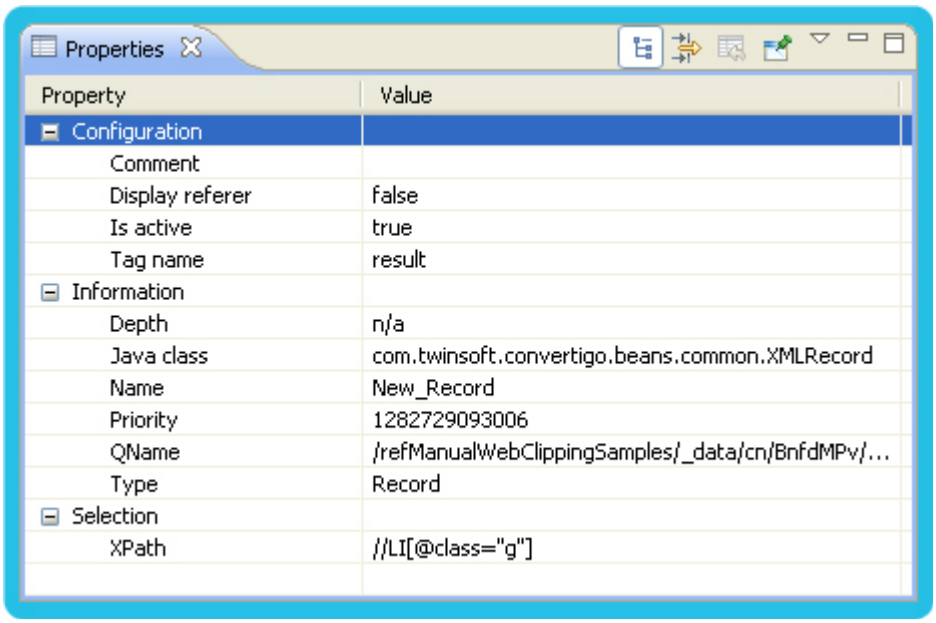


Figure 2 - 265: Record extraction rule - Configuration example

The **Description** property elements (columns) are defined in the **Projects** view of the Convertigo Studio:

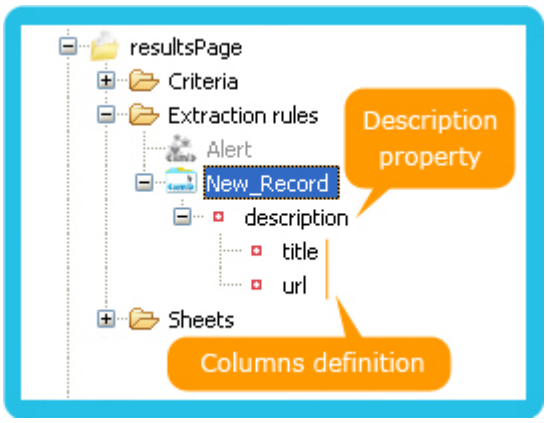


Figure 2 - 266: Record extraction rule - Description property: columns definition

And each column properties are edited in the **Properties** view of the Convertigo Studio:

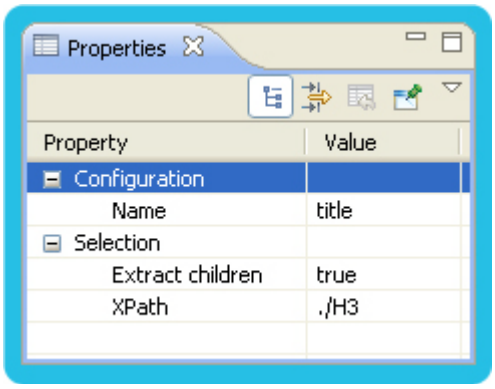


Figure 2 - 267: Record extraction rule - Column properties configuration example

When the rule is executed, the resulting XML includes the record elements and their child data:

```
<document connector="refManualWebClippingSamplesConnector" context="studio_refManualWebClippingSamples:refManualV
  <result>
    <title>Convertigo Web Clipper (CWC)</title>
    <url>https://greenhouse.lotus.com/plugins/plugincatalog.nsf/assetDetails.xsp?action=openDocument&amp;documentId=
  </result>
  <result>
    <title>C-EMS Web Clipper</title>
    <url>http://www.convertigo.com/en/products/modules/182-c-ems-web-clipper.html</url>
  </result>
  <result>
    <title>Convertigo Documentation</title>
    <url>http://www.convertigo.com/en/docs.html</url>
  </result>
  <result>
    <title>Starting with Convertigo Web Integrator</title>
    <url>http://download.convertigo.com/webrepository/doc/CWI/PDF/startingWithConvertigoWebIntegrator.pdf</url>
  </result>
  <result>
    <title>Starting with Convertigo Web Clipper</title>
    <url>http://download.convertigo.com/webrepository/doc/CWC/PDF/startingWithConvertigoWebClipper.pdf</url>
  </result>
  <result>
    <title>Heloworld.com - Helo World</title>
    <url>http://whois.domaintools.com/heloworld.com</url>
  </result>
  <result>
    <title>Proxyph.com - Proxy Ph</title>
    <url>http://whois.domaintools.com/proxyph.com</url>
  </result>
  <result>
    <title>C-EMS Web Clipper : C-EMS Web Clipper « découpe » des parties de ...</title>
    <url>http://www.quelsoft.com/fiche/c-ems-web-clipper-c20-618-1160.html</url>
  </result>
  <result>
    <title>Le mashup frappe à la porte de l'entreprise</title>
    <url>http://www.01net.com/editorial/361012/le-mashup-frappe-a-la-porte-de-lentreprise/</url>
  </result>
  <result>
    <title>Swiv - Flux RSS Placement produit - Veille d'information</title>
    <url>http://swiv.eu/fr.htm?s=placement+produit</url>
  </result>
</document>
```

Figure 2 - 268: Record extraction rule - Resulting XML with rule

TABLE (WEB) 

OBJECT DESCRIPTION

Extracts data from a web page in an XML table.

The *Table* extraction rule helps you extract data into a table structure.

Extracted data are organized into an XML table structure made of:

- a parent base element "corresponding to" the base HTML element containing data to extract. In most cases, this element is an HTML `table` element.
- child elements "corresponding to" recurring row HTML part. In most cases, if root is an HTML `table` element, each `tr` element is assumed to be a row of data.
- rows child elements "corresponding to" recurring column HTML part. In most cases, within a `tr` row, each `td` element is assumed to be a cell.

The rule is applied if the result of the table XPath expression evaluation exists in the HTML page DOM.

Based on this root, the child elements are also defined by Xpath expressions. Each Xpath expression may be relative to its parent element Xpath expression, using the following syntax: "`./`".

By default, a row XPath expression is `.//TR`. You can add restrictions in the XPath expression, for example `.//TR[position() > 1]`, meaning that each `tr` element within the `table` is a row except the first one.

Columns are defined relatively to their parent row. By default, a column Xpath expression is `.//TD`.

The resulting table element is appended to the HTML transaction DOM as follows:

```
<table_tagname referer="referer_url">
  <row_tagname>
   <column1_tagname>extracted text from xpath</column1_tagname>
   <column2_tagname>extracted text from xpath</column2_tagname>
  </row_tagname>
  <row_tagname>
   <column1_tagname>extracted text from xpath</column1_tagname>
   <column2_tagname>extracted text from xpath</column2_tagname>
  </row_tagname>
</table_tagname>
```

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accumulate data in same table	boolean	configuration	Accumulates every data from several screens in the same table XML element.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Description	XMLVector	selection	Describes the table structure in which data are extracted. The table is structured as a root element containing row and column child elements, which are defined through **Description** property. This property is a list of child elements descriptions, also named rows descriptions. Each row is described using two properties: • Row tag name: Row element tag name in resulting DOM (the default tag name is `row`). All resulting nodes described by the following row XPath are tagged after this name. • Row XPath: XPath expression selecting row elements . It is often defined relatively to parent *Table* extraction rule XPath expression. The XPath can result in several nodes (ex `.//TR`) meaning that several rows are being extracted.Each row description contains a list of child elements descriptions, also named columns descriptions. Each column description is composed of the following fields: • Column tag name: Columns element tag name in resulting DOM (the default name is `column`). • Column XPath: XPath expression selecting column elements (data to extract). It is often defined relatively to parent row Xpath expression using the following syntax: "`./`". • Extract children: Indicates whether text extraction should recurse on child elements of the elements found thanks to the Xpath (by default it is set to `true`). As it needs more CPU if set to "true", it is then recommended to customize your XPath (using `//text()` function for example) and set this property to `false`.
Display referer	boolean	configuration	Defines whether the referer URL is displayed in the output XML element. If this property is set to `true`, the referer URL is added as an attribute, named `referer`, to the XML element added by the extraction rule.
Flip table orientation	boolean	configuration	Flips the table orientation, turning lines into columns and columns into lines.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Tag name	String	configuration	Defines the table tag name in the resulting XML (default tag name is `XMLTable`).
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

This section contains a generic and a specific description example, as well as an example of table flipping.

Generic Description Example

If we consider the following web page from SalesForce website, famous CRM SaaS application:

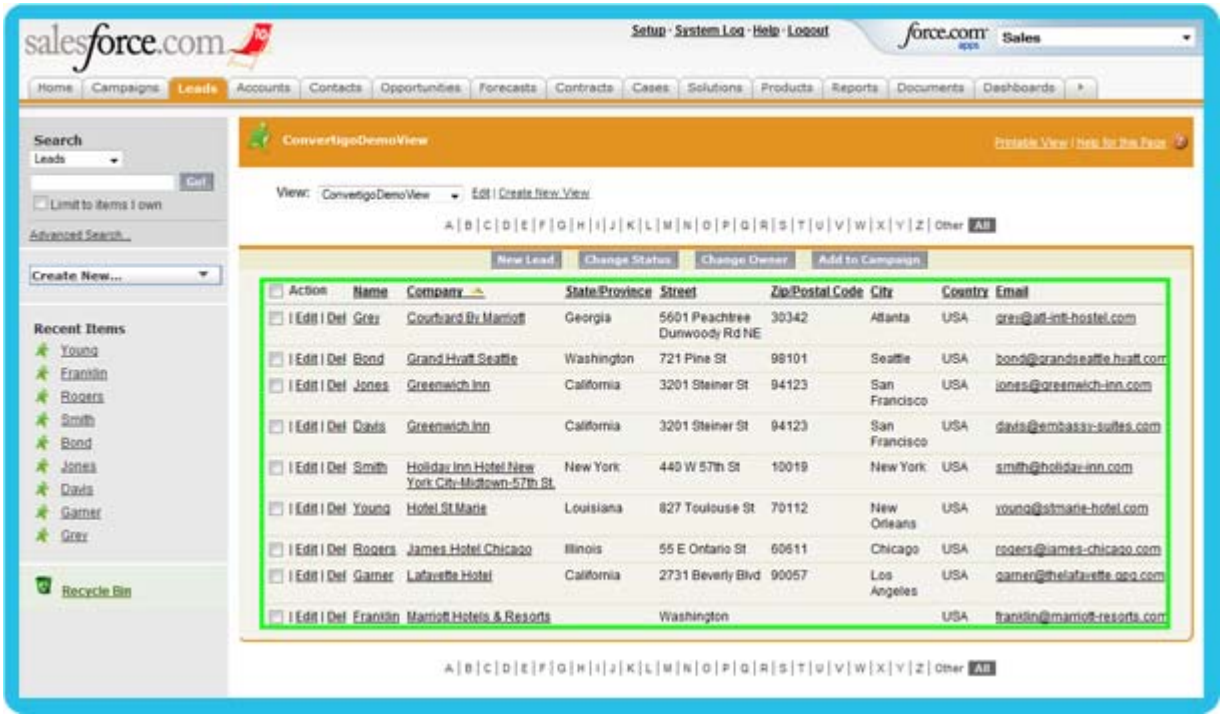


Figure 2 - 269: Table extraction rule - SalesForce Leads web page

We can notice that this Web page contains data in a table, surrounded in green in previous figure. These data are organized into a structure based on an HTML `table` element in the page DOM. Using a *Table* extraction rule, we want to extract these data.

This extraction rule will extract sets of data from this web page by extracting the real HTML `table` element :

- listing `TR` elements as rows,
- extracting data from `TD` and/or `TH` elements as columns,
- naming data columns after the header line columns.

The *Table* extraction rule creation wizard automatically helps configurating the rule's rows and columns for standard HTML `table` cases, such as this one. The columns and rows are then automatically displayed and are adjustable in the **New Table** wizard page:

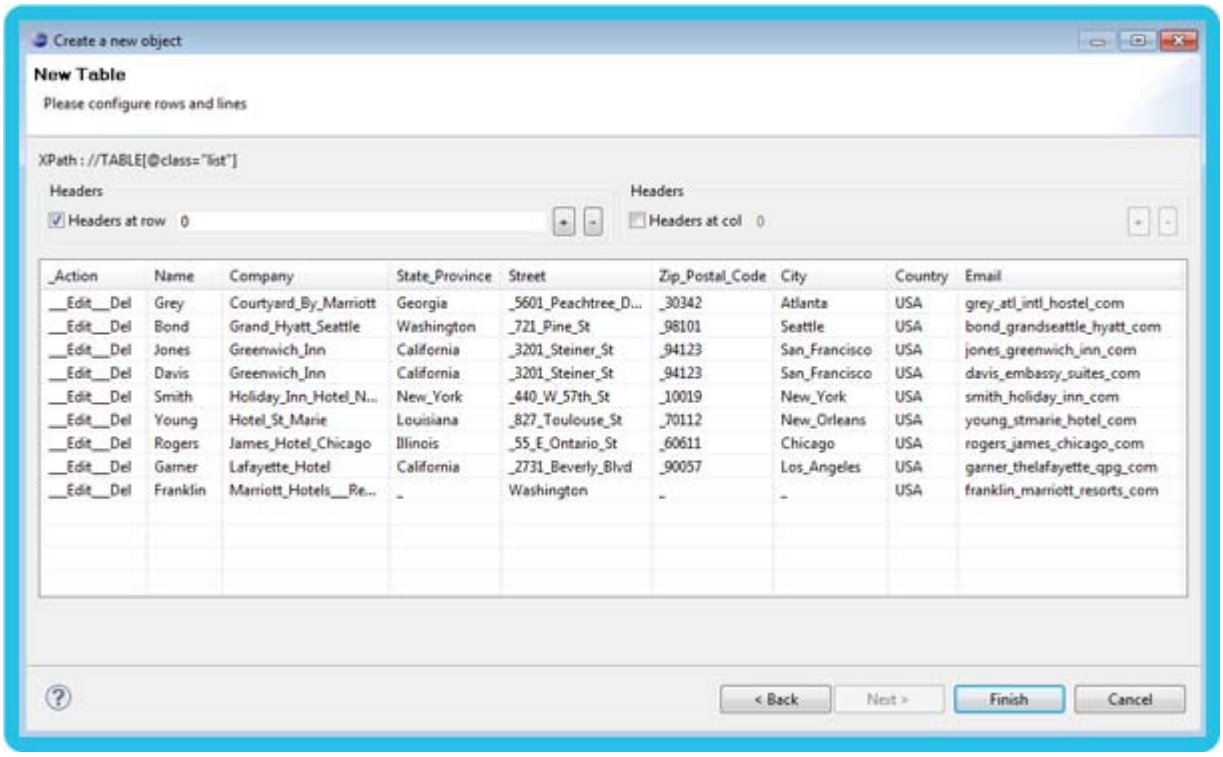


Figure 2 - 270: Table extraction rule - New Table wizard page

The *Table* extraction rule is then automatically created by the wizard with the following parameters:

```
Table [
  xpath="//TABLE[@class="list"]"
  tagname="XMLTable"
  accumulate data in same table=false
  display referer=false
  flip table orientation=false
  description={
    row [row tag name="row" row xpath="(./TBODY/TR)[position()>1]"
      column [name="_Action" extract children=true
      xpath="(./TD|./TH)[1]"],
      column [name="Name" extract children=true
      xpath="(./TD|./TH)[2]"],
      column [name="Company" extract children=true
      xpath="(./TD|./TH)[3]"],
      column [name="State_Province" extract children=true
      xpath="(./TD|./TH)[4]"],
      column [name="Street" extract children=true
      xpath="(./TD|./TH)[5]"],
      column [name="Zip_Postal_Code" extract children=true
      xpath="(./TD|./TH)[6]"],
      column [name="City" extract children=true
      xpath="(./TD|./TH)[7]"],
      column [name="Country" extract children=true
      xpath="(./TD|./TH)[8]"],
```

```
        column [name="Email" extract children=true
        xpath="(./TD|./TH)[9]"],
      ],
    }
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio, to change the tagname to `Leads`:

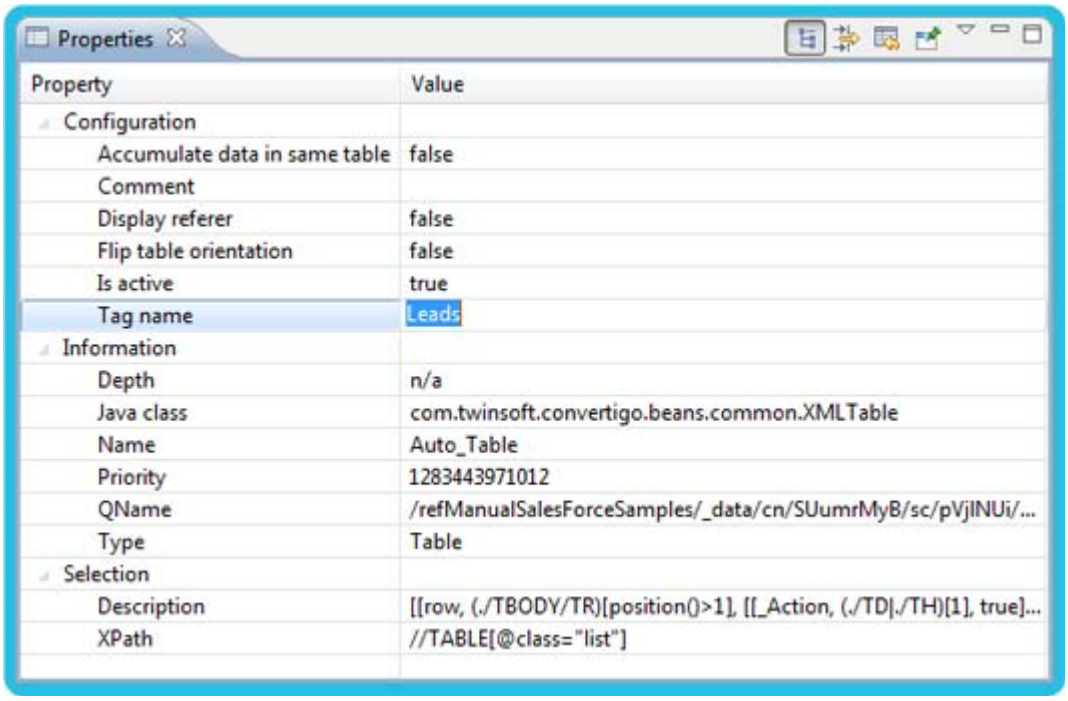


Figure 2 - 271: Table extraction rule - Configuration example

The **Description** property elements (rows and columns) are defined in the **Projects** view of the Convertigo Studio:

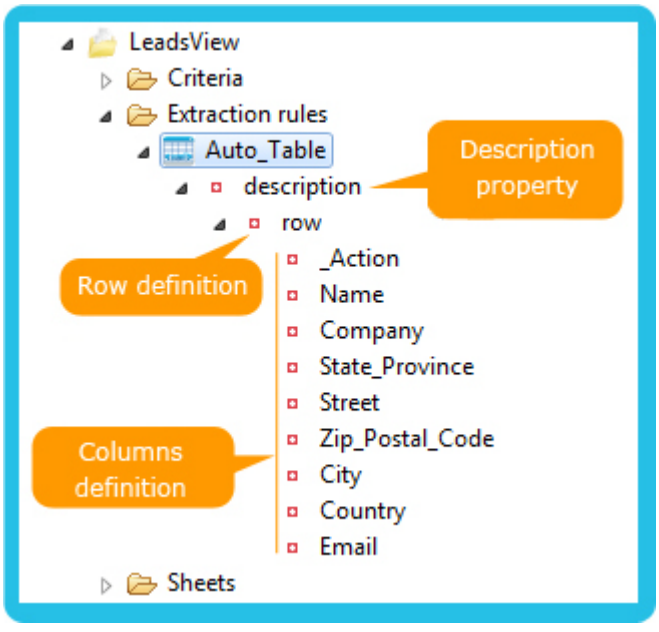


Figure 2 - 272: Table extraction rule - Description property: rows and columns definition

Row properties and each column properties are edited in the **Properties** view of the Convertigo Studio:

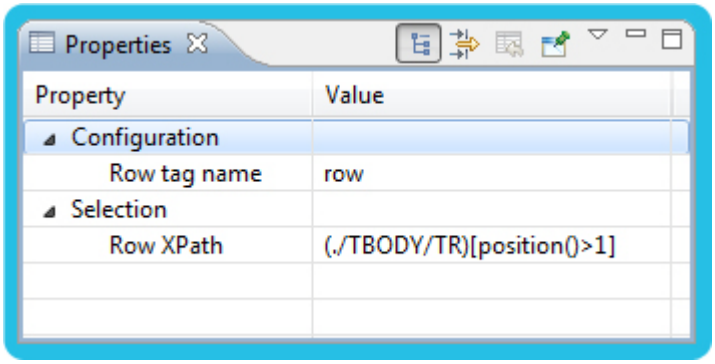


Figure 2 - 273: Table extraction rule - Row properties configuration example

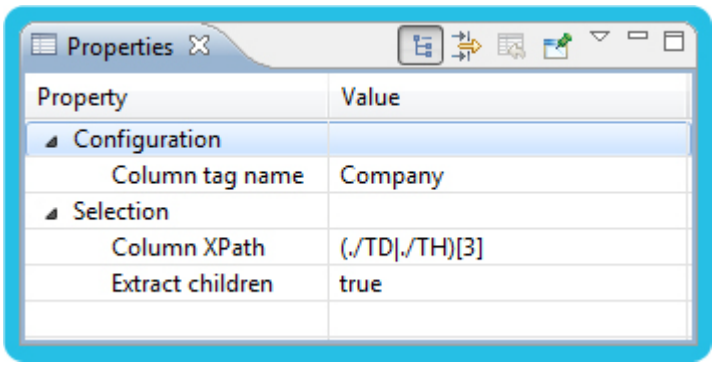


Figure 2 - 274: Table extraction rule - Column properties configuration example

When the rule is executed, the resulting XML includes the table element, its rows, and their columns data:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<document connector="SalesForceConnector" context="studio_refManualSalesForce
  <Leads>
    <row>
      <_Action> | Edit | Del</_Action>
      <Name>Grey</Name>
      <Company>Courtyard By Marriott</Company>
      <State_Province>Georgia</State_Province>
      <Street>5601 Peachtree Dunwoody Rd NE</Street>
      <Zip_Postal_Code>30342</Zip_Postal_Code>
      <City>Atlanta</City>
      <Country>USA</Country>
      <Email>grey@atl-intl-hostel.com</Email>
    </row>
    <row>
      <_Action> | Edit | Del</_Action>
      <Name>Bond</Name>
      <Company>Grand Hyatt Seattle</Company>
      <State_Province>Washington</State_Province>
      <Street>721 Pine St</Street>
      <Zip_Postal_Code>98101</Zip_Postal_Code>
      <City>Seattle</City>
      <Country>USA</Country>
      <Email>bond@grandseattle.hyatt.com</Email>
    </row>
    <row>
      <_Action> | Edit | Del</_Action>
      <Name>Jones</Name>
      <Company>Greenwich Inn</Company>
      <State_Province>California</State_Province>
      <Street>3201 Steiner St</Street>
      <Zip_Postal_Code>94123</Zip_Postal_Code>
      <City>San Francisco</City>
      <Country>USA</Country>
      <Email>jones@greenwich-inn.com</Email>
    </row>
    <row>
      <_Action> | Edit | Del</_Action>
      <Name>Davis</Name>
      <Company>Greenwich Inn</Company>
      <State_Province>California</State_Province>
      <Street>3201 Steiner St</Street>
      <Zip_Postal_Code>94123</Zip_Postal_Code>
      <City>San Francisco</City>
      <Country>USA</Country>
      <Email>davis@embassy-suites.com</Email>
    </row>
    <row>
      <_Action> | Edit | Del</_Action>
      <Name>Smith</Name>
      <Company>Holiday Inn Hotel New York City-Midtown-57th St.</Company>
      <State_Province>New York</State_Province>
      <Street>440 W 57th St</Street>
      <Zip_Postal_Code>10019</Zip_Postal_Code>
```


Figure 2 - 275: Table extraction rule - Resulting XML with automatically parametered rule

Specific Description Example

If we consider the following web page from Google search engine:

Figure 2 - 276: Table extraction rule - Google results web page

We can notice that Google results are recurrent elements on this Web page and they are all contained in a root DIV element. Using a *Table* extraction rule, we want to extract two pieces of data for each Google result: the title and the URL pointed by the link.

This extraction rule will extract sets of data from this web page by:

- listing all Google results and defining them as row element,
- extracting each Google result title by concatenating the texts contained in the title HTML element with the texts contained in its child HTML elements,
- and inserting this text into a `title` element,
- extracting the text contained in the hypertext link on each Google title result,
- and inserting it into an `url` element.

The *Table* extraction rule is created with the following parameters:

```
Table [
  xpath="//DIV[@id="res"]"
  tagname="results"
  accumulate data in same table=false
  display referer=false
  flip table orientation=false
  description={
    row [row tag name="result" row xpath=".//OL/LI[@class="g"]"
      column [name="title" extract children=true xpath="./H3"],
      column [name="url" extract children=false
    xpath="/H3/A[@class="l"]/@href"]
    ],
  }
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

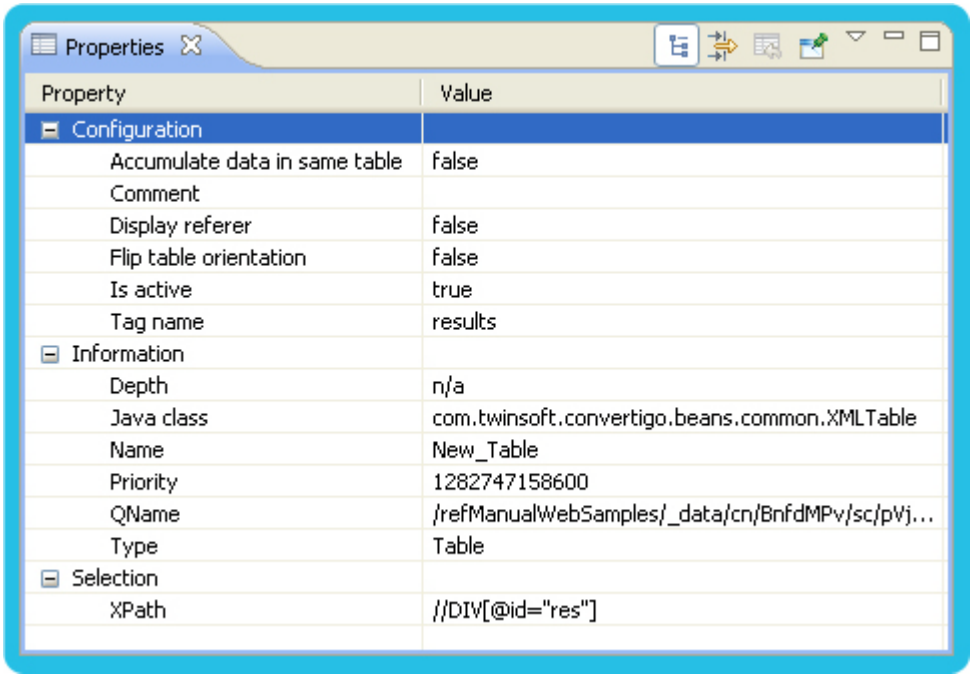


Figure 2 - 277: Table extraction rule - Configuration example

The **Description** property elements (rows and columns) are defined in the **Projects** view of the Convertigo Studio:

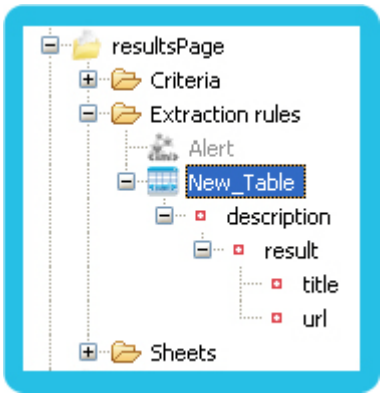


Figure 2 - 278: Table extraction rule - Description property: rows and columns definition

Row properties and each column properties are edited in the **Properties** view of the Convertigo Studio:

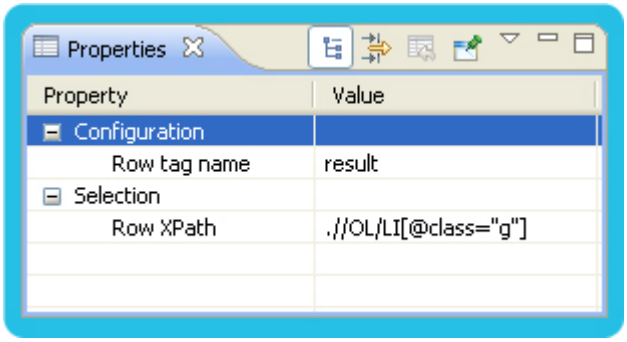


Figure 2 - 279: Table extraction rule - Row properties configuration example

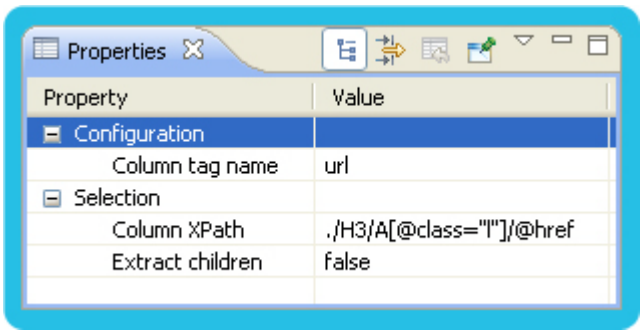


Figure 2 - 280: Table extraction rule - Column properties configuration example

When the rule is executed, the resulting XML includes the table element, its rows, and their columns data:

```
<document connector="refManualWebClippingSamplesConnector" context="studio_refManualWebSamples:refManualWebClippi
  <results>
    <result>
      <title>Convertigo Web Clipper (CWC)</title>
      <url>https://greenhouse.lotus.com/plugins/plugincatalog.nsf/assetDetails.xsp?action=openDocument&amp;documentId
    </result>
    <result>
      <title>C-EMS Web Clipper</title>
      <url>http://www.convertigo.com/en/products/modules/182-c-ems-web-clipper.html</url>
    </result>
    <result>
      <title>Convertigo Documentation</title>
      <url>http://www.convertigo.com/en/docs.html</url>
    </result>
    <result>
      <title>Starting with Convertigo Web Integrator</title>
      <url>http://download.convertigo.com/webrepository/doc/CWI/PDF/startingWithConvertigoWebIntegrator.pdf</url>
    </result>
    <result>
      <title>Starting with Convertigo Web Clipper</title>
      <url>http://download.convertigo.com/webrepository/doc/CWC/PDF/startingWithConvertigoWebClipper.pdf</url>
    </result>
    <result>
      <title>Heloworld.com - Helo World</title>
      <url>http://whois.domaintools.com/heloworld.com</url>
    </result>
    <result>
      <title>Proxyph.com - Proxy Ph</title>
      <url>http://whois.domaintools.com/proxyph.com</url>
    </result>
    <result>
      <title>C-EMS Web Clipper : C-EMS Web Clipper « découpe » des parties de ...</title>
      <url>http://www.quelsoft.com/fiche/c-ems-web-clipper-c20-618-1160.html</url>
    </result>
    <result>
      <title>Le mashup frappe à la porte de l'entreprise</title>
      <url>http://www.01net.com/editorial/361012/le-mashup-frappe-a-la-porte-de-lentreprise/</url>
    </result>
    <result>
      <title>Swiv - Flux RSS Placement produit - Veille d'information</title>
      <url>http://swiv.eu/fr.htm?s=placement+produit</url>
    </result>
  </results>
</document>
```

Figure 2 - 281: Table extraction rule - Resulting XML with rule

Specific description with flip table option example

The following rule has a specific description, row and column XPaths are inverted to fit with the flip table option selected.

The following figure describes thedifferences between standard table and table with flipping process, on which configuration this property applies and the different XML that are generated without and with this property:

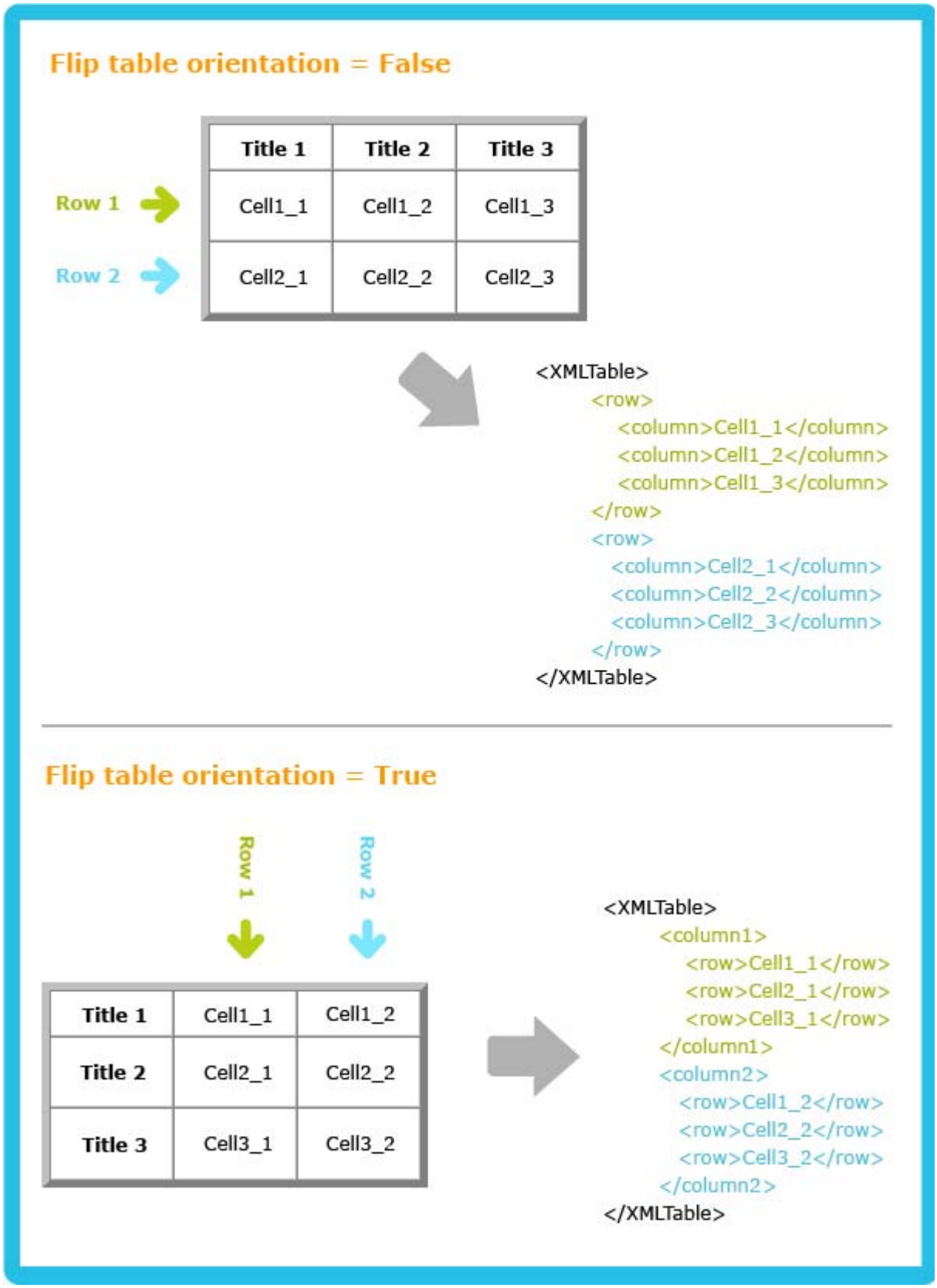


Figure 2 - 282: Table extraction rule - Table flipping example

The two corresponding *Table* extraction rules are parametered as follows:

```
Table [
  xpath="//TABLE[@id='standard']"
```

```
   tagname="table_standard"
   displayreferer="true"
   description={
      Row [
         xpath=".//TR[position()>1]"
         tagname="row"
         columns={
            Column [xpath="./TD" tagname="column" childs="true"],
         }
      ]
   }
]
```

for the first table, without flipping process, and as follows:

```
Table [
   xpath="//TABLE[@id='reverse']"
   tagname="table_reverse"
   displayreferer="true"
   description={
      Row [
         xpath=".//TR[1]"
         tagname="column1"
         columns={
            Column [xpath="./TD[position()>1]" tagname="row"
            childs="true"],
         }
      ],
      Row [
         xpath=".//TR[2]"
         tagname="column2"
         columns={
            Column [xpath="./TD[position()>1]" tagname="row"
            childs="true"],
         }
      ]
   }
]
```

for the second table, with flipping process.

TEXT 

OBJECT DESCRIPTION

Extracts data from a web page in an XML text node.

The *Text* extraction rule helps you extract a text from an HTML page. It is applied if the result of the Xpath expression evaluation exists in the HTML page DOM.

It creates a simple XML element containing text extracted from the first matching node. This text element is appended to the resulting HTML transaction DOM as follows:

```
<text_tagname referer="referer_url">extracted text from xpath</text_tagname>
```

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Display referer	boolean	configuration	Defines whether the referer URL is displayed in the output XML element. If this property is set to `true`, the referer URL is added as an attribute, named `referer`, to the XML element added by the extraction rule.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Recurse	boolean	configuration	Defines whether text extraction should recurse on child elements of the matching node.
Tag name	String	configuration	Defines the tag name in the resulting XML (default tag name is `XMLText`).
XPath	String	selection	Defines the Xpath expression of nodes on which the extraction rule applies. Depending on the extraction rule, the execution of this **Xpath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

If we consider the following web page from Google search engine:

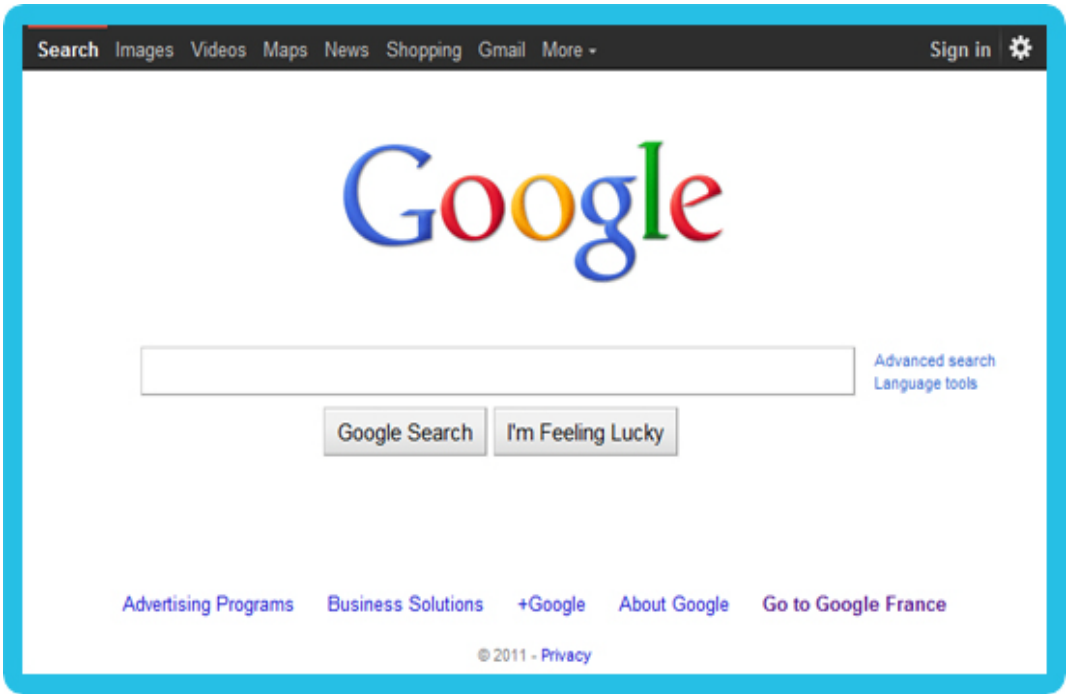


Figure 2 - 283: Text extraction rule - HTML web page

We can see four links under the search form. We want to extract the text of the first link thanks to a *Text* extraction rule.

The Xpath matching on all links is generated using the **Xpath Evaluator** of the Convertigo Studio:

Figure 2 - 284: Text extraction rule - Generating Xpath in the Xpath Evaluator

Then the rule is created with the following parameters:

```
Text [
  xpath=//DIV[@id="fll"]//A
  recurse=true
  tag name="link"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

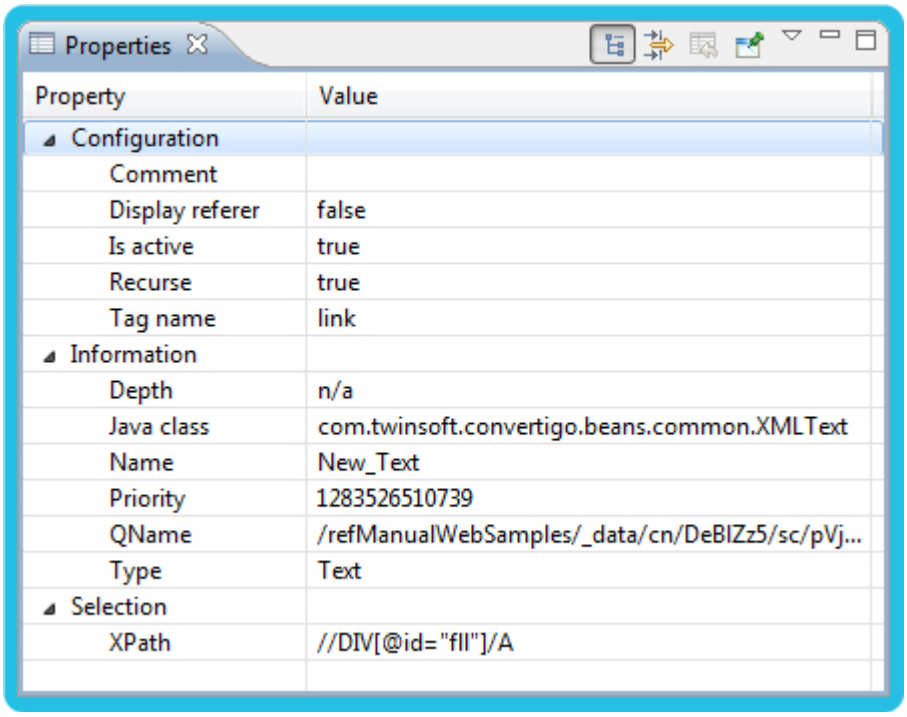


Figure 2 - 285: Text extraction rule - Configuration example

After extraction, the link text is extracted in the output XML as follows:

```
<document connector="refManualWebSamplesConnector" context="studio
  <link>Advertising Programs</link>
</document>
```

Figure 2 - 286: Text extraction rule - Resulting XML with rule

HTTP HEADERS

OBJECT DESCRIPTION

Extracts the HTTP headers of the response to the request Convertigo did to get the current web page.

The *HTTP headers* extraction rule always extracts HTTP headers of responses. They are appended to the HTML transaction output DOM as follows:

```
<HttpHeaders>
  <header headerName="header1" headerValue="value1"/>
  <header headerName="header2" headerValue="value2"/>
</HttpHeaders>
```

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let's consider the following page from Google search engine:

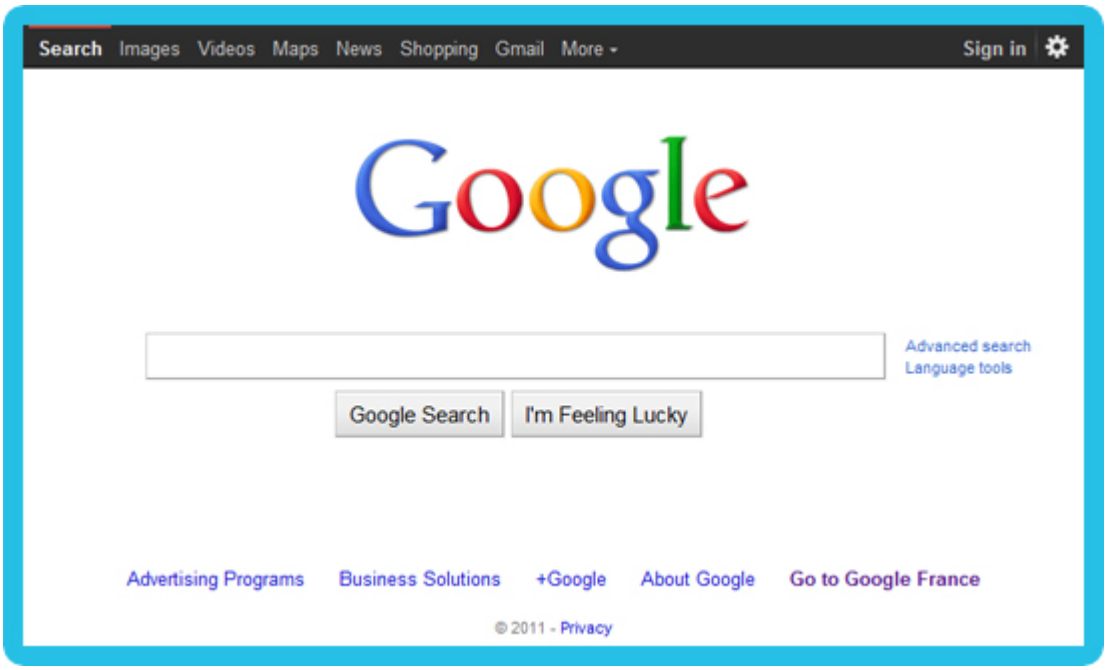


Figure 2 - 287: HTTP headers extraction rule - HTML web page

In this example, we want to extract the HTTP headers of the page thanks to an *HTTP headers* extraction rule, in a screen class defined in `simpleGoogleConnector` connector of `sample_refManual_statements` project.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

The rule has no property to set, it is just created on corresponding screen class.

Once it is generated, the *Add link* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

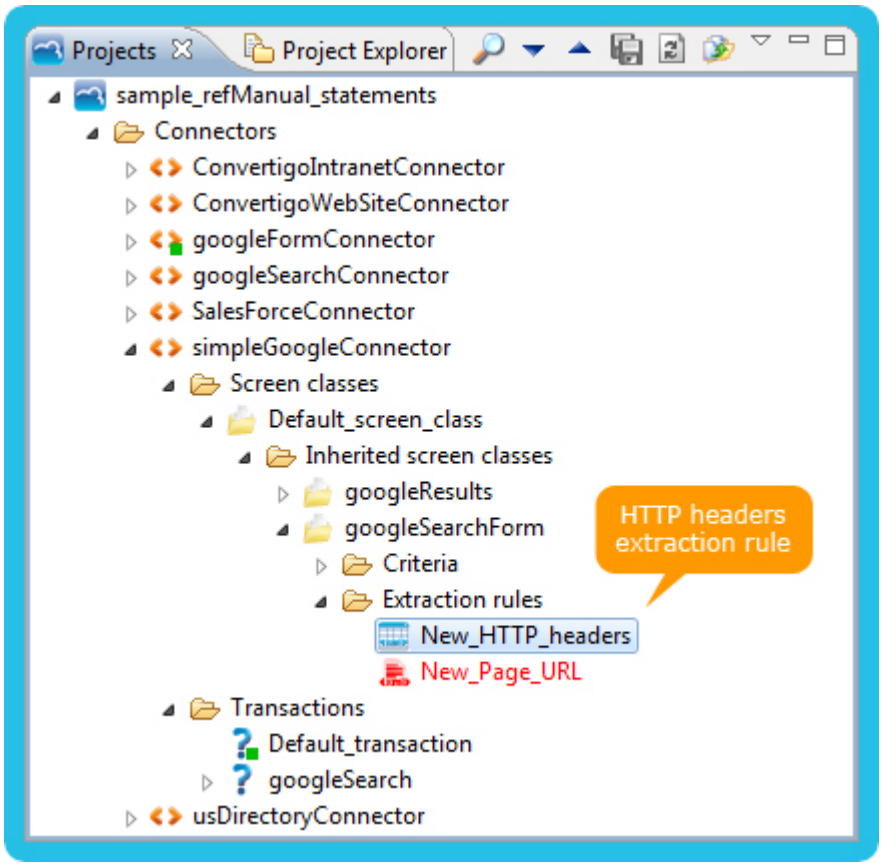


Figure 2 - 288: HTTP headers extraction rule - Object in Projects view

After execution of the extraction rule, the HTTP headers are extracted in the output XML as follows:

```
<document connector="simpleGoogleConnector" context="studio_sample_refManual_s
  <HttpHeaders>
    <header headerName="Content-Length" headerValue="0"/>
    <header headerName="Content-Type" headerValue="text/html"/>
    <header headerName="Date" headerValue="Thu, 12 Jan 2012 17:28:29 GMT"/>
    <header headerName="Server" headerValue="GFE/2.0"/>
  </HttpHeaders>
</document>
```

Figure 2 - 289: HTTP headers extraction rule - Resulting XML with rule

PAGE URL

OBJECT DESCRIPTION

Retrieves the current page URL.

The *Page URL* extraction rule always extracts the URL of current web page. It is appended to the HTML transaction output DOM as follows:

```
<HttpUrl>urlContent</HttpUrl>
```

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Tag name	String	configuration	Defines the tag name. By default, the URL is extracted in a `HttpUrl` tag. This property allows to change this XML tag.

EXAMPLES

Let's consider the following page from Google search engine:

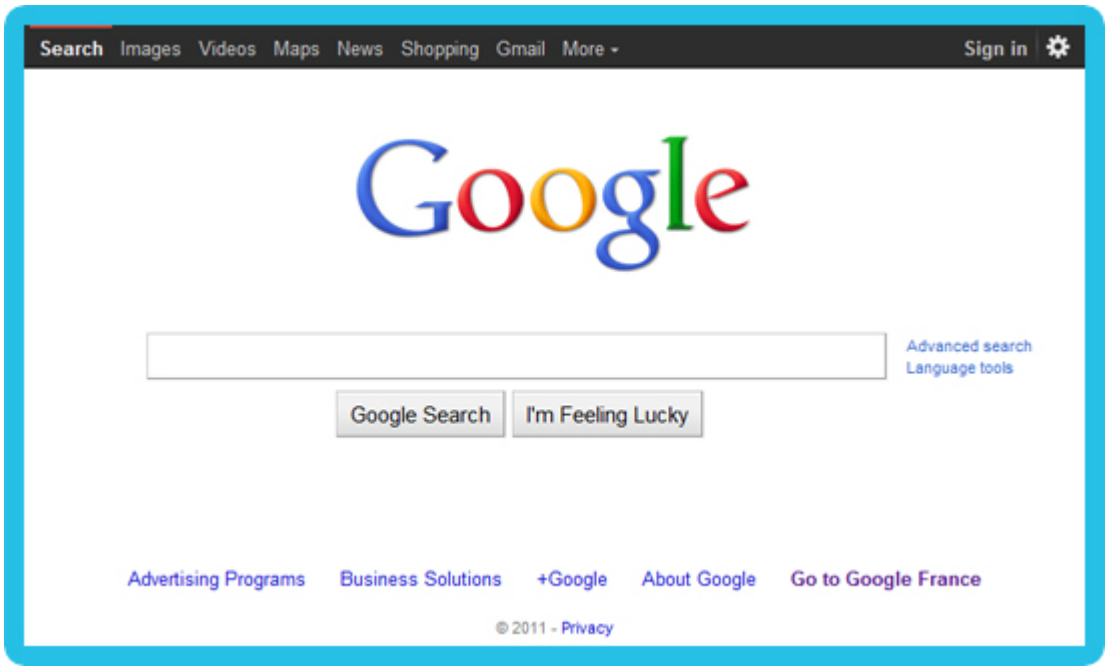


Figure 2 - 290: Page URL extraction rule - HTML web page

In this example, we want to extract the url of the page thanks to a *Page URL* extraction rule, in a screen class defined in `simpleGoogleConnector` connector of

`sample_refManual_statements` project.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Convertigo Samples and Demos** > **Reference Manual examples** > **HTML connector statements examples** in the **New Project** wizard.*

The rule is created with the following parameters:

```
Page URL [
   tag name="myPageUrl"
]
```

This parameter is edited in the **Properties** view of the Convertigo Studio:

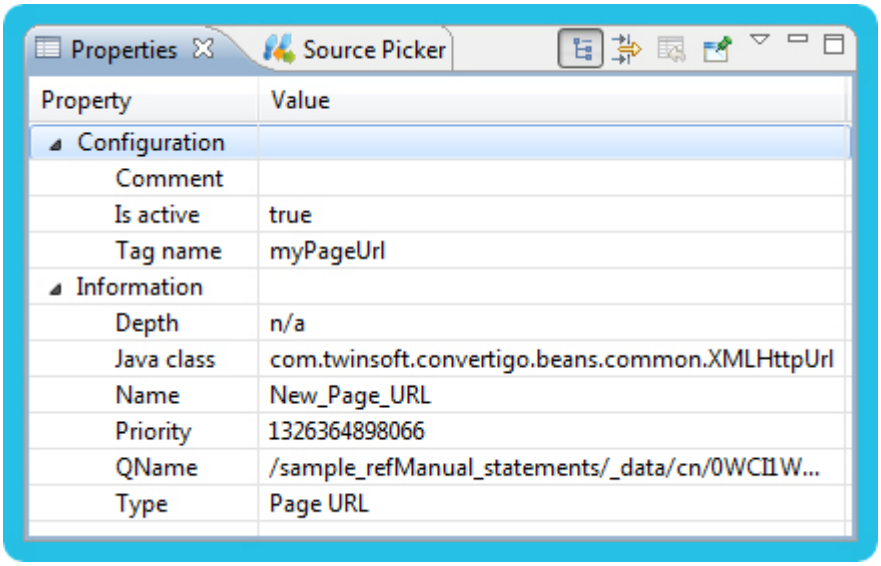


Figure 2 - 291: Page URL extraction rule - Configuration example

Once it is generated, the *Add link* extraction rule appears as follows in the screen class **Extraction rules** folder in the **Projects** view:

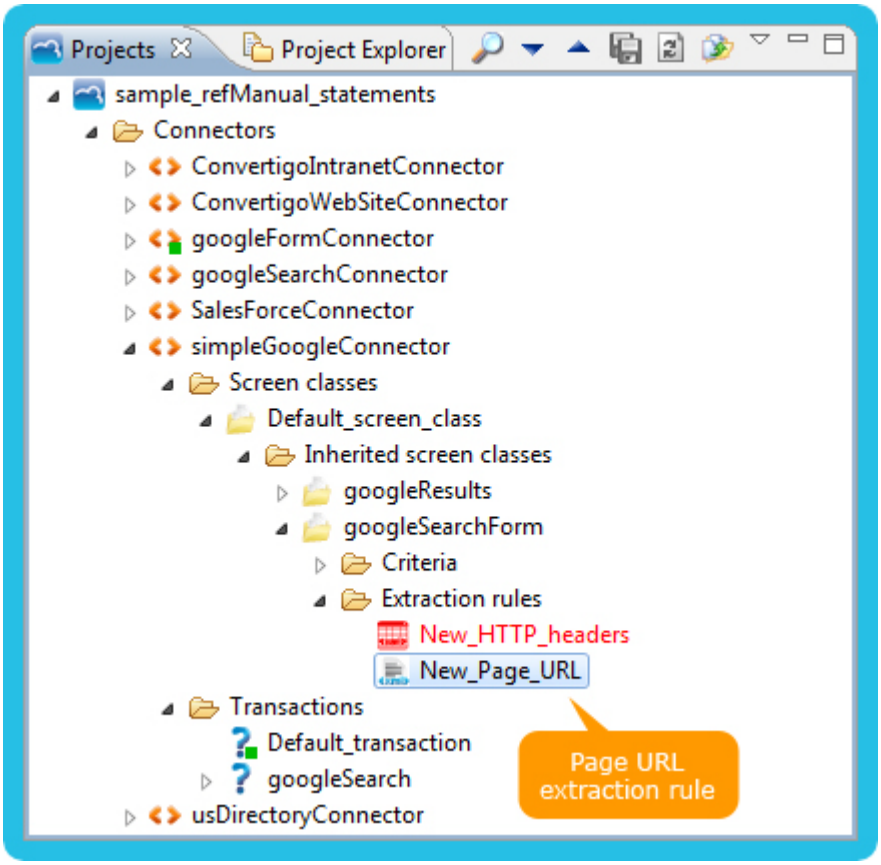


Figure 2 - 292: Page URL extraction rule - Object in Projects view

After execution of the extraction rule, the page URL is extracted in the output XML as follows:

```
<document connector="refManualWebClippingSamplesCor
  <myPageUrl>http://www.google.com/</myPageUrl>
</document>
```

Figure 2 - 293: Page URL extraction rule - Resulting XML with rule

PRINT SCREEN 

OBJECT DESCRIPTION

Takes a screenshot of the current web page.

The *Print screen* extraction rule creates a screenshot of the currently displayed web page and generates a `Base64` representation that is inserted in an element of the output XML.

The *Print screen* extraction rule can take a screenshot of the whole web page or of a part of the page. To select only a part of the page, you can use the **Capture corner** and **Capture size** properties.

The image compression method of the generated binary data can be chosen using the **Image format** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Capture corner left	int	configuration	Defines the horizontal space in pixel between the left of the browser window and the left of the captured frame. The *Print screen* extraction rule can capture only a part of the web page. Use this property to choose the position of the left of the screenshot area. Leave this property to its `0` default value if you want to capture a screenshot from the left of the page.
Capture corner top	int	configuration	Defines the vertical space in pixel between the top of the browser window and the top of the captured frame. The *Print screen* extraction rule can capture only a part of the web page. Use this property to choose the position of the top of the screenshot area. Leave this property to its `0` default value if you want to capture a screenshot from the top of the page.
Capture size height	int	configuration	Defines the height of the captured frame (in pixels). The *Print screen* extraction rule can capture only a part of the web page. Use this property to choose the height of the screenshot area. Leave this property to its `-1` default value if you want to capture the whole height of the web page.
Capture size width	int	configuration	Defines the width of the captured frame (in pixels). The *Print screen* extraction rule can capture only a part of the web page. Use this property to choose the width of the screenshot area. Leave this property to its `-1` default value if you want to capture the whole width of the web page.

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Data url prefix	boolean	configuration	Includes a prefix to the Base64 binary data to allow a direct use of the image from the XML. Setting this property to `true` adds a `` prefix to the image binary data in the output XML, with `<xxx>` value is `jpg` or `png` depending on the **Image format** property value. This allows the developer to directly use the image data without writing it in a file. For example, it can be used in an `src` attribute of an `IMG` HTML tag.
Image format	ImageFormat	configuration	Defines the image compression method used to generate the image binary data. This property can take several values: • `png`: using this value generates a fine but heavy image, • `jpeg`: using this value generates a blurred but lightweight image.
Image scale	float	configuration	Defines the ratio to reduce (<1) or increase (>1) the size of the final captured image. The *Print screen* extraction rule can automatically perform a transformation on the captured image: • increasing its original size if you use a value superior to `1`, • reducing its original size if you use a value inferior to `1`.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Tag name	String	configuration	Defines the tag name in the resulting XML (default tag name is `PrintScreen`).
Waiting delay	long	selection	Defines the minimum delay (in ms) to wait after the "completed" event to realize the screenshot. This property allows to define a time to wait before the screenshot is performed and after the last `document:completed` event, in order to be sure that the page is fully rendered before the image is generated. The default value is set to `100` ms.

2.8.4 Statements

HANDLER STATEMENTS

HANDLER 

OBJECT DESCRIPTION

Defines an event handler.

A *Handler* is similar to a `Function` except that it is automatically executed when the associated event occurs. Events that can be associated with it are:

- Transaction starting event, which is fired when Convertigo starts executing the client request to a transaction,
- XML generated event, which is fired when Convertigo has generated the XML response, just before the transaction ends.

Notes:

- A *Handler* contains other statements that define the actions to be performed. It can return a result value (empty string by default). If `cancel` is returned by the *Transaction starting Handler*, the rest of the transaction execution is canceled.
- *Handlers* can only be added to a transaction, one *Handler* of each event type per transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Event type	String	standard	Defines the handler associated event type. *Handlers* can be associated to the following event types: • `TransactionStarted` event type: executed when the transaction start event occurs, • `XmlGenerated` event type: executed when the XML generated event occurs.
Infinite loop protection	boolean	standard	Defines whether the handler should be protected against infinite loops in transaction. If set to `true` (default value), the handler prevents infinite loops by throwing a Convertigo Engine exception when an infinite loop is detected. Default value should not be changed unless you specifically want the handler to authorize loops in transaction.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Result	String	standard	Defines the handler's default resulting value. Depending on the **Event type**, this property can be chosen among several available values. For a *Transaction starting Handler*, this property can take the following values: • `<empty>` or `""`: continues the transaction execution, • `cancel`: stops the transaction and cancels the rest of the transaction execution. Note: The *Handler*'s default return value defined thanks to this property can be overridden by a child *Return* statement.

EXAMPLES

Example 1: transaction start Handler

Let's consider the examples defined for several statements, like *Browser property change* statement, *Credentials statement*, *Context Get* statement or *Exception* statement. These examples all explain the need to add statements in *transaction start Handlers*:

- *Browser property change* statement or *Credentials* statement are to be executed before the transaction connects to the target website and executes *Screen classes handlers*,
- *Context Get* statement, *If* statement or *Exception* statement are set in this *Handler* because of the transaction behavior that is implemented (tests on variables values before connecting to websites).

In all cases, the *Handlers* are created with the following parameters:

```
Handler [
   event type=TransactionStarted
   result=<empty>
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

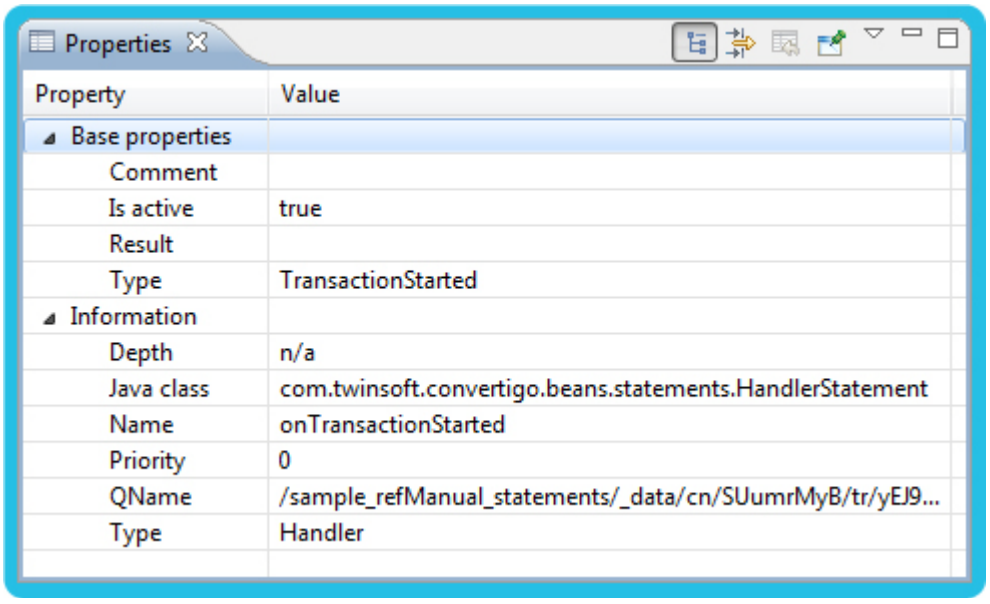


Figure 2 - 294: Transaction start Handler - Configuration example

The *transaction start Handlers* are created in the **Functions** folder of each transaction and appear as folows in the **Projects** view.

- The *transaction start Handlers* including statements having to be executed before the transaction connects to the target website appear as follows in the **Projects** view:

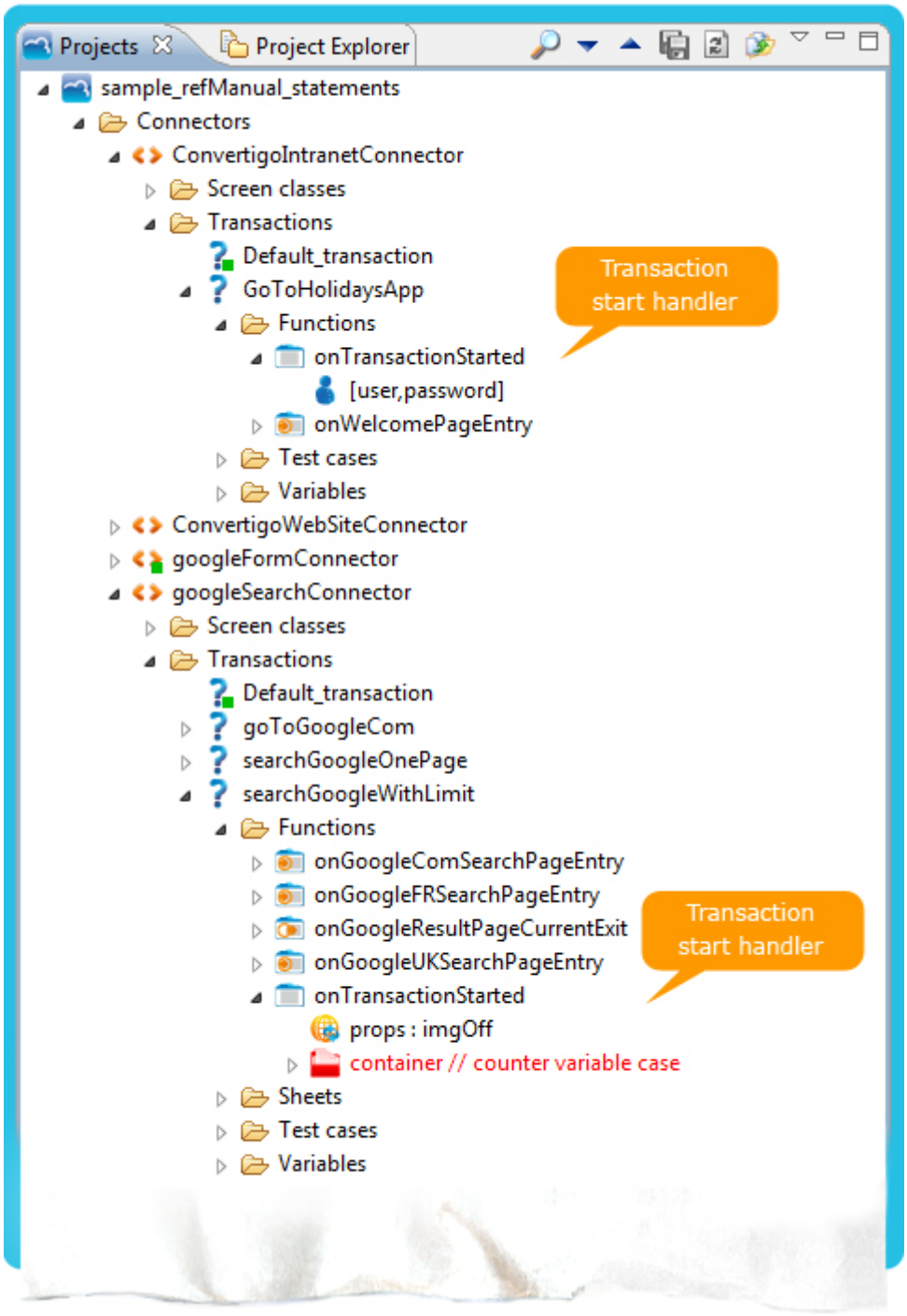


Figure 2 - 295: Transaction start Handler - Objects in Projects view

- The *transaction start Handlers* that are set because of the transaction behavior that is implemented appear as follows in the **Projects** view:

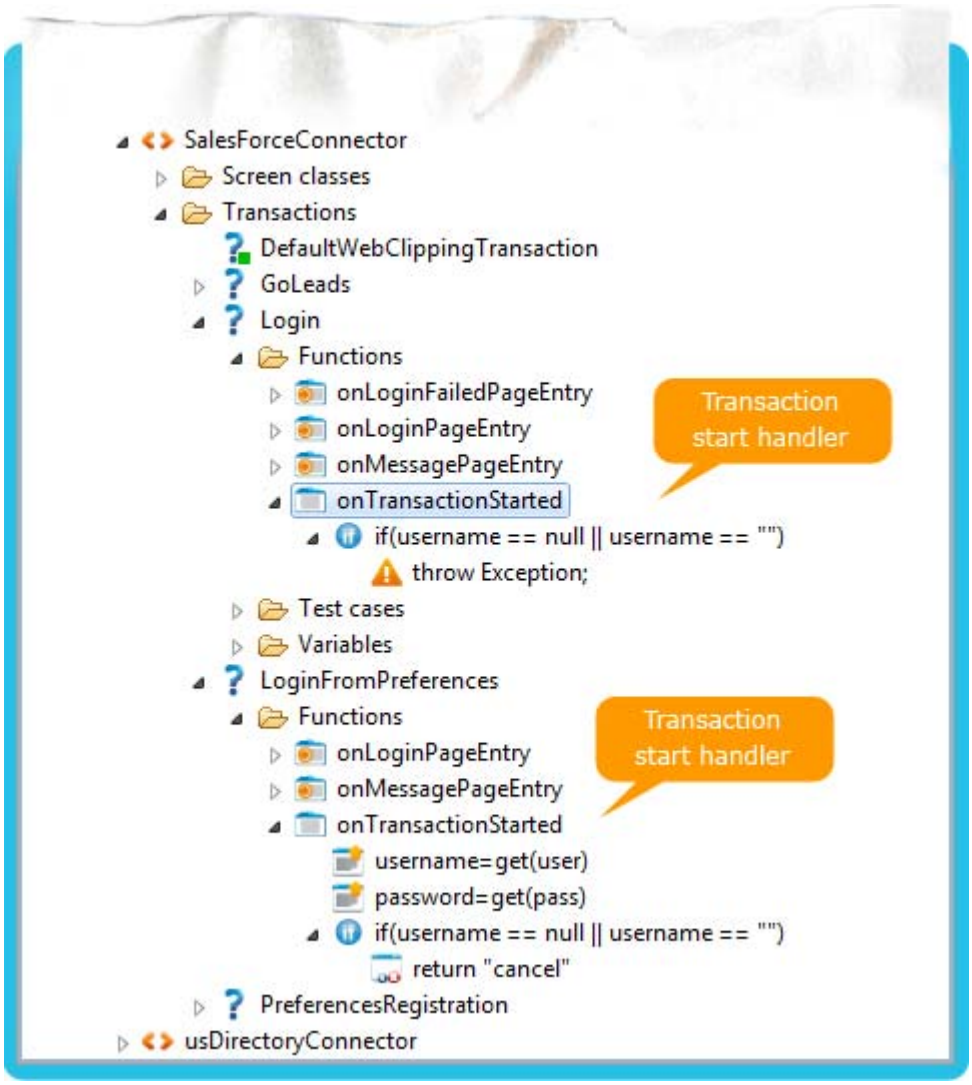


Figure 2 - 296: Transaction start Handler - Objects in Projects view

We can observe that on one of these *transaction start Handlers*, the default result value is overridden by a *Return* statement returning `"cancel"` value, whereas the *Handler* result value is empty:

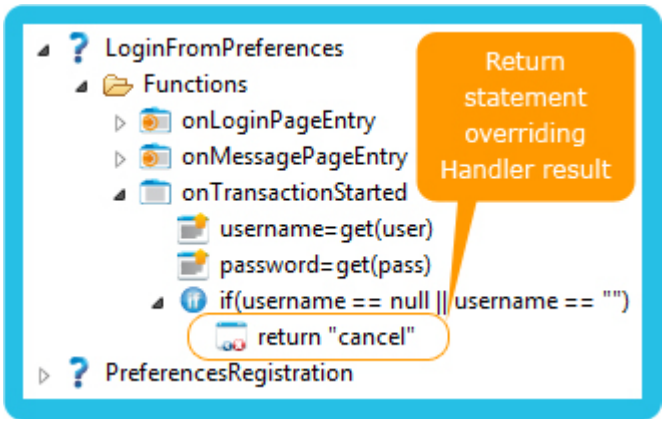


Figure 2 - 297: Transaction start Handler - Return statement overridding the default empty result value

SCREEN CLASS ENTRY HANDLER 

OBJECT DESCRIPTION

Defines a screen class entry handler.

A *Handler* is similar to a `Function` except that it is automatically executed when the associated event occurs. The *Screen class entry handler* is a handler associated with the entry on a screen class event. It is executed when Convertigo detects the screen class corresponding to this *Screen class entry handler*, before executing the extraction rules associated with this screen class.

In other words, a *Screen class entry handler* is executed when arriving on the screen class associated with this handler.

Notes:

- A *Screen class entry handler* contains other statements that define the actions to be performed on this screen class. It can return a result value (`redetect` by default as it is an entry handler).
- *Screen class entry handlers* can only be added to a transaction, one *Screen class entry handler* for each screen class per transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Infinite loop protection	boolean	standard	Defines whether the handler should be protected against infinite loops in transaction. If set to `true` (default value), the handler prevents infinite loops by throwing a Convertigo Engine exception when an infinite loop is detected. Default value should not be changed unless you specifically want the handler to authorize loops in transaction.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Result	String	standard	Defines the handler's default resulting value. Depending on the handler type, this property can be chosen among several available values. For a *Screen class entry handler*, this property can take the following values: • `<empty>` or `""`: goes on and extracts data using extraction rules, • `continue`: similar to `<empty>` value, • `redetect`: does not extract data and redetects a new screen class, • `skip`: stops the transaction without extracting data. Note: The *Handler*'s default return value defined thanks to this property can be overridden by a child *Return* statement.
Screen class	String	standard	Defines the screen class to be monitored. This property allows to associate the *Screen class handler* with the screen class on which it is executed. The possible values for this property are generated from the screen classes defined in the connector. If the screen class is renamed in the connector, the **Screen class** property of associated *Screen class handlers* (entry or exit) are automatically updated.

EXAMPLES

Example 1

Let’s consider the US directory website. Every page of this site contains a search form on its top:

Figure 2 - 298: Screen class entry handler - US directory website pages with search form

A `searchBusiness` transaction, which defines three variables named `business`, `cityZIPcode`, and `state`, sets those three values into search form input fields and launch a search by clicking on the `Search` button.

The transaction is able to fill the search form on the three pages of the US directory website that are identified by screen classes in the project. The actions to perform on every screen are defined thanks to a *Function* statement named `setSearchInputs` as follows:

- set Business input,
- set City / ZIP code input,
- select State in combobox,
- click on the `Search` button.

The transaction may start on a page that is not recognized as belonging to a screen class defined in the project, it will be recognized as belonging to the `Default_screen_class` screen class. In this case, the transaction displays a custom error message depending on the web page URL.

Based on previous specifications, the transaction has actions to perform on four different

screen classes. For this purpose, four *Screen class entry handlers* are created in the transaction, with the following parameters (for example for the `SearchBusinessPage` screen class):

```
Screen class entry handler [
   screen class=SearchBusinessPage
   result=continue
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio (for example for `SearchBusinessPage` *Screen class entry handler*):

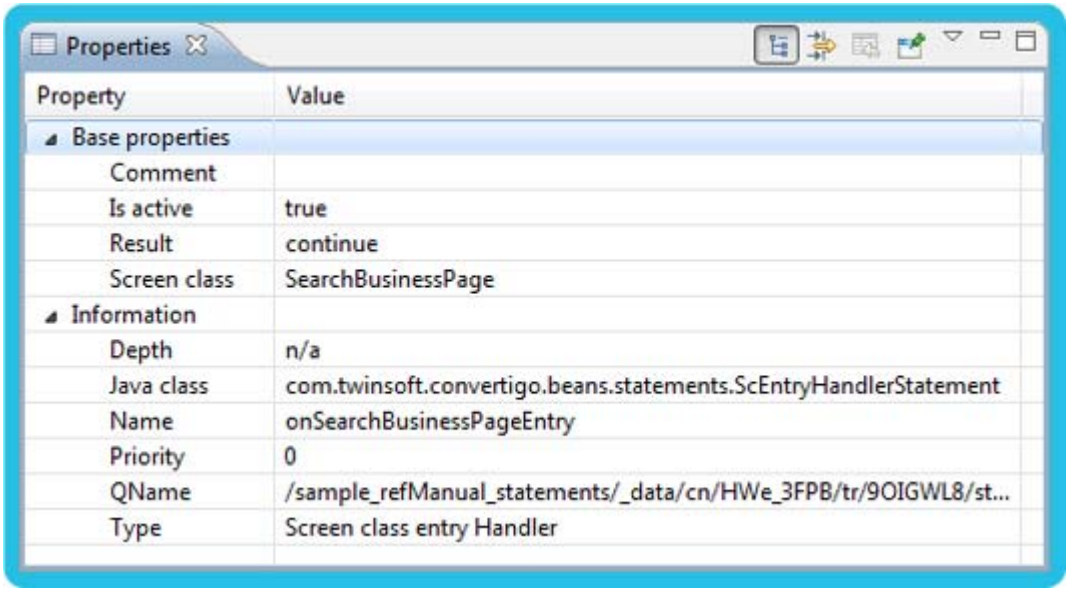


Figure 2 - 299: Screen class entry handler - Configuration example

These *Screen class entry handlers* are created in the **Functions** folder of the transaction, containing statements used to implement the transaction behavior described above, and appear as follows in the **Projects** view:

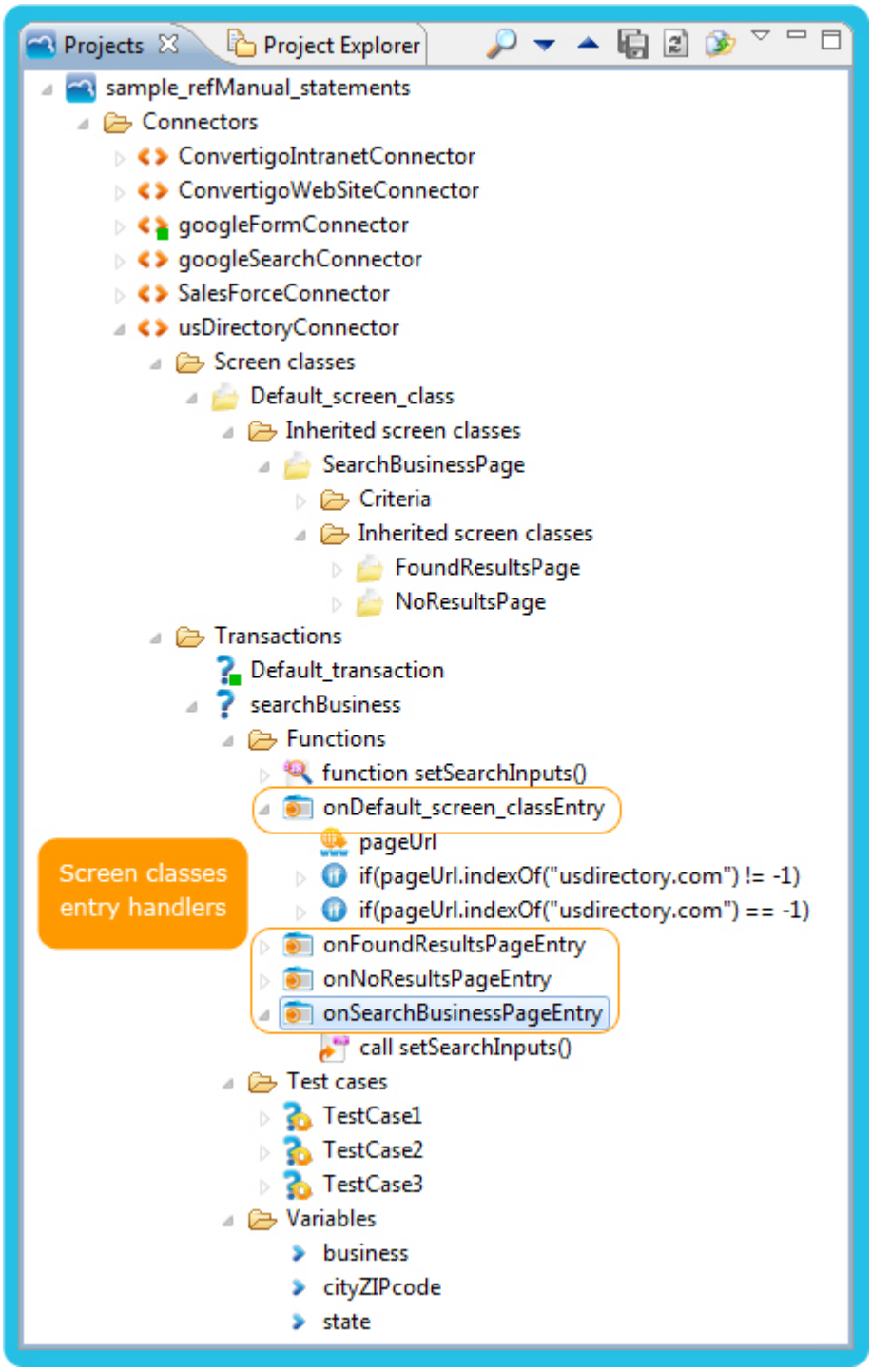


Figure 2 - 300: Screen class entry handler - Objects in Projects view

When executing one of the test cases defined for the transaction, the statements contained in *Screen class entry handlers* are executed depending on the detected screen class.

Example 2

Let's consider the `searchGoogle` transaction set in the context of the "*Starting With Convertigo Web Integrator*" Quick Guide. It contains three *Screen class entry handlers*, defined to handle the actions to perform on the different screen classes met while navigating on Google website. One of these *Screen class entry handlers* is called `onGoogleComSearchPageEntry`, in order to handle the arrival on Google.com search page.

The purpose of the `onGoogleComSearchPageEntry` *Screen class entry handler* is to execute statements implementing the following behavior when accessing the Google.com search homepage:

- input the searched keyword in the field,
- click on the **Google Search** button.

The `onGoogleComSearchPageEntry` *Screen class entry handler* appears as follows:

- in the **Projects** view of the Convertigo Studio:

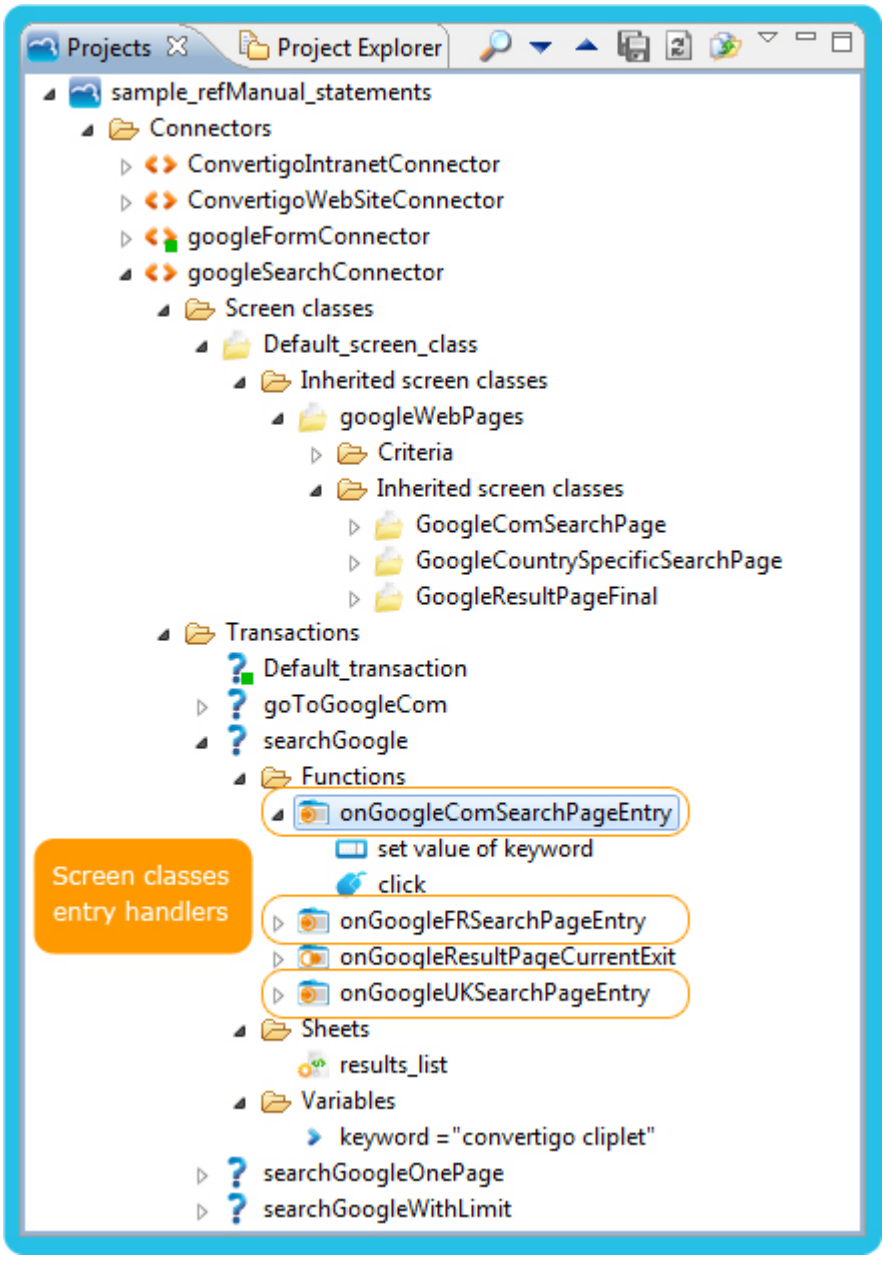


Figure 2 - 301: Screen class entry handler - Objects in Projects view

- in the **Properties** view of the Convertigo Studio:

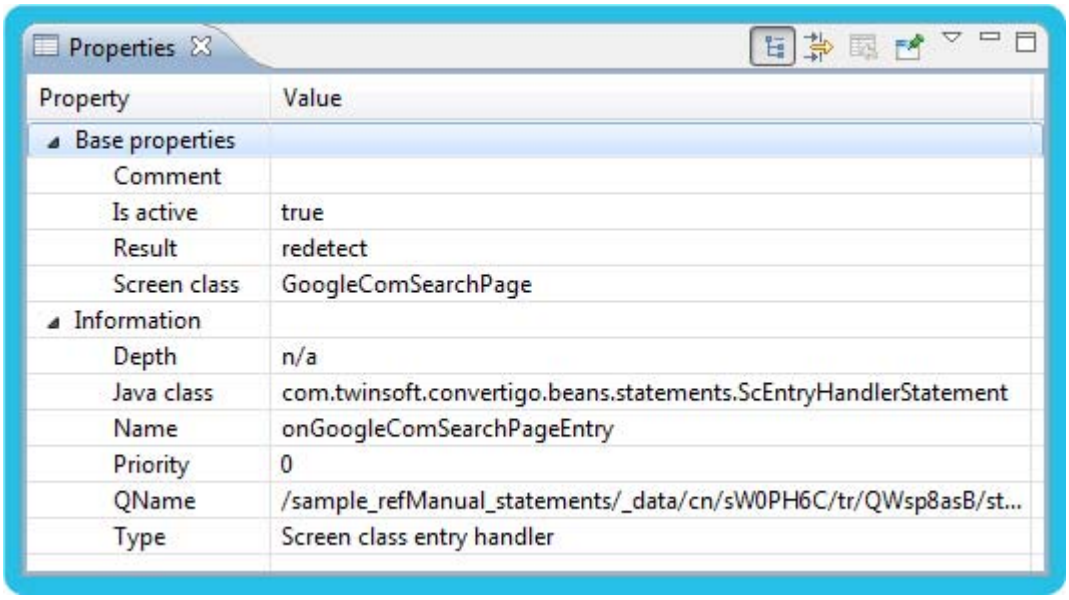


Figure 2 - 302: Screen class entry handler - Object properties

The **Screen Class** property is set to `googleComSearchPage` screen class so as the `onGoogleComSearchPageEntry` *Screen class entry handler* is triggered on detection of the `googleComSearchPage` screen class.

The **Result** property is set to `redetect`, causing the new accessed screen class to be detected after actions have been performed on the current page.

SCREEN CLASS EXIT HANDLER 

OBJECT DESCRIPTION

Defines a screen class exit handler.

A *Handler* is similar to a `Function` except that it is automatically executed when the associated event occurs. The *Screen class exit handler* is a handler associated with the exit from a screen class event. It is executed when Convertigo detects the screen class corresponding to this *Screen class exit handler*, after having executed the extraction rules associated with this screen class.

In other words, a *Screen class exit handler* is executed when leaving the screen class associated with this handler.

Notes:

- A *Screen class exit handler* contains other statements that define the actions to be performed on this screen class. It can return a result value (`accumulate` by default as it is an exit handler).
- *Screen class exit handlers* can only be added to a transaction, one *Screen class exit handler* for each screen class per transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Infinite loop protection	boolean	standard	Defines whether the handler should be protected against infinite loops in transaction. If set to `true` (default value), the handler prevents infinite loops by throwing a Convertigo Engine exception when an infinite loop is detected. Default value should not be changed unless you specifically want the handler to authorize loops in transaction.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Result	String	standard	Defines the handler's default resulting value. Depending on the handler type, this property can be chosen among several available values. For a *Screen class exit handler*, this property can take the following values: • `<empty>` or `""`: stops the process and ends the transaction, • `continue`: similar to `<empty>` value, • `accumulate`: accumulates extracted data (data is extracted from last detected screen class then added to any other extracted data) and redetects a new screen class. Note: The *Handler*'s default return value defined thanks to this property can be overridden by a child *Return* statement.
Screen class	String	standard	Defines the screen class to be monitored. This property allows to associate the *Screen class handler* with the screen class on which it is executed. The possible values for this property are generated from the screen classes defined in the connector. If the screen class is renamed in the connector, the **Screen class** property of associated *Screen class handlers* (entry or exit) are automatically updated.

EXAMPLES

Let's consider the `searchGoogle` transaction set in the context of the *"Starting With Convertigo Web Integrator*" Quick Guide. It contains one *Screen class exit handler* defined to handle the actions to perform on the screen class met while navigating on Google website and from which we want to extract data. This *Screen class exit handler* is called `onGoogleResultsPageCurrentExit`, in order to handle the leaving from Google results page.

The purpose of the `onGoogleResultsPageCurrentExit` *Screen class exit handler* is to execute a specific statement implementing the following behavior when exiting Google results page: click on the **Next** button.

The `onGoogleResultsPageCurrentExit` *Screen class exit handler* appears as follows:

- in the **Projects** view of the Convertigo Studio:

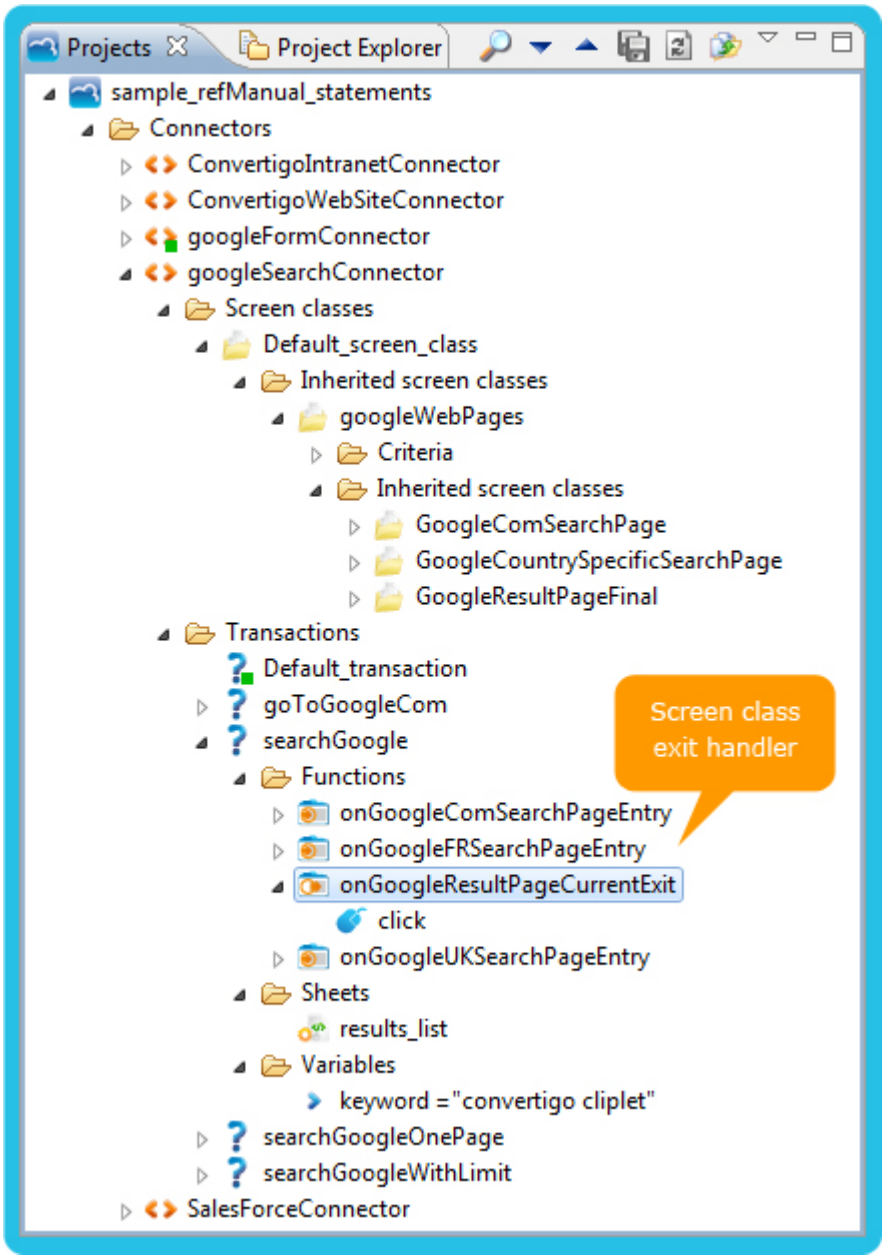


Figure 2 - 303: Screen class exit handler - Object in Projects view

- in the **Properties** view of the Convertigo Studio:

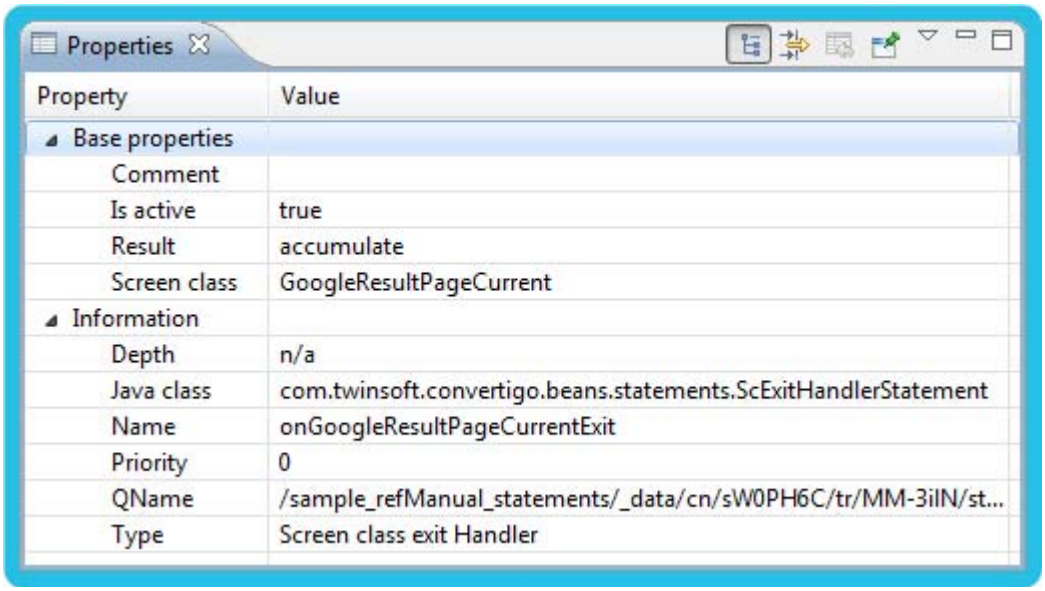


Figure 2 - 304: Screen class exit handler - Object properties

The **Screen Class** property is set to `googleResultsPageCurrent` screen class so as the `onGoogleResultsPageCurrentExit` *Screen class exit handler* is triggered when leaving the `googleResultsPageCurrent` screen class.

The **Result** property is set to `accumulate`, causing the data extracted from pages detected as belonging to the `googleResultsPageCurrent` screen class to be accumulated under the same tag in the XML output, and the new accessed screen class to be detected after actions have been performed on the current page.

DEFAULT ENTRY HANDLER 

OBJECT DESCRIPTION

Defines a transaction default entry handler.

A *Handler* is similar to a `Function` except that it is automatically executed when the associated event occurs. The *Default entry handler* is a screen class entry handler, associated with an entry on a screen class event. If present in a transaction, it is executed when no specific screen class entry handler is defined for the currently detected screen class. In other words, it is a generic screen class entry handler.

Thus, *Default entry handler* can be defined for multiple screen classes on which the programmer knows that the same actions are to be done.

Beware that this handler will be executed for the `Default_screen_class` screen class if no specific handler is defined for this screen class. As the `Default_screen_class` screen class matches every page that is not defined in the connector, this handler can potentially be executed on a lot of unmanaged pages. Such behaviors can lead to infinite loop transactions.

Notes:

- A *Default entry handler* contains other statements that define the actions to be performed. It can return a result value (`redetect` by default as this is an entry handler).
- *Default entry handlers* can only be added to a transaction, one *Default entry handler* per transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Infinite loop protection	boolean	standard	Defines whether the handler should be protected against infinite loops in transaction. If set to `true` (default value), the handler prevents infinite loops by throwing a Convertigo Engine exception when an infinite loop is detected. Default value should not be changed unless you specifically want the handler to authorize loops in transaction.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Result	String	standard	Defines the handler's default resulting value. Depending on the handler type, this property can be chosen among several available values. For a *Default entry handler*, this property can take the following values: • `<empty>` or `""`: goes on and extracts data using extraction rules, • `continue`: similar to `<empty>` value, • `redetect`: does not extract data and redetects a new screen class, • `skip`: stops the transaction without extracting data. Note: The *Handler*'s default return value defined thanks to this property can be overridden by a child *Return* statement.

EXAMPLES

Let's consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

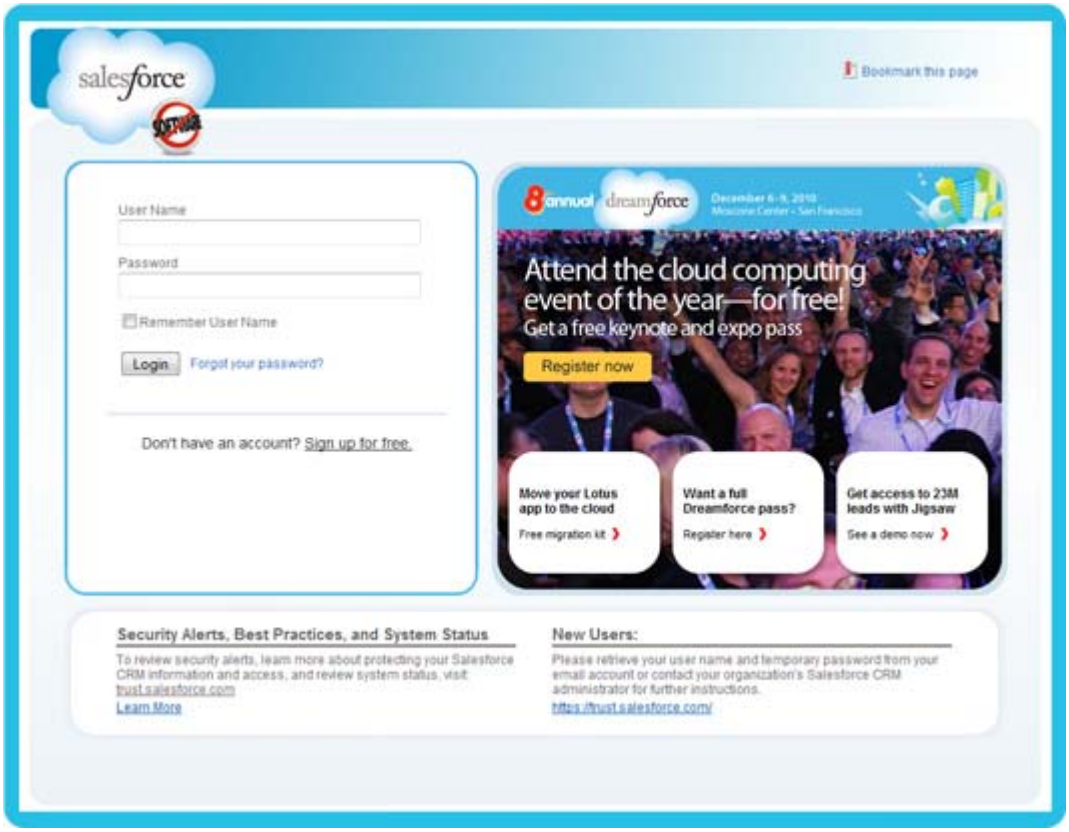


Figure 2 - 305: Default entry handler - SalesForce website authentication page

A `Login` transaction, which defines two variables named `username` and `password`, is written to authenticate the user on SalesForce website. This transaction:

- fills the User Name and Password fields using the `user` / `password` variables,
- submits the authentication form to access to SalesForce welcome page.

When the authentication is properly done, the SalesForce website home page appears as follows:

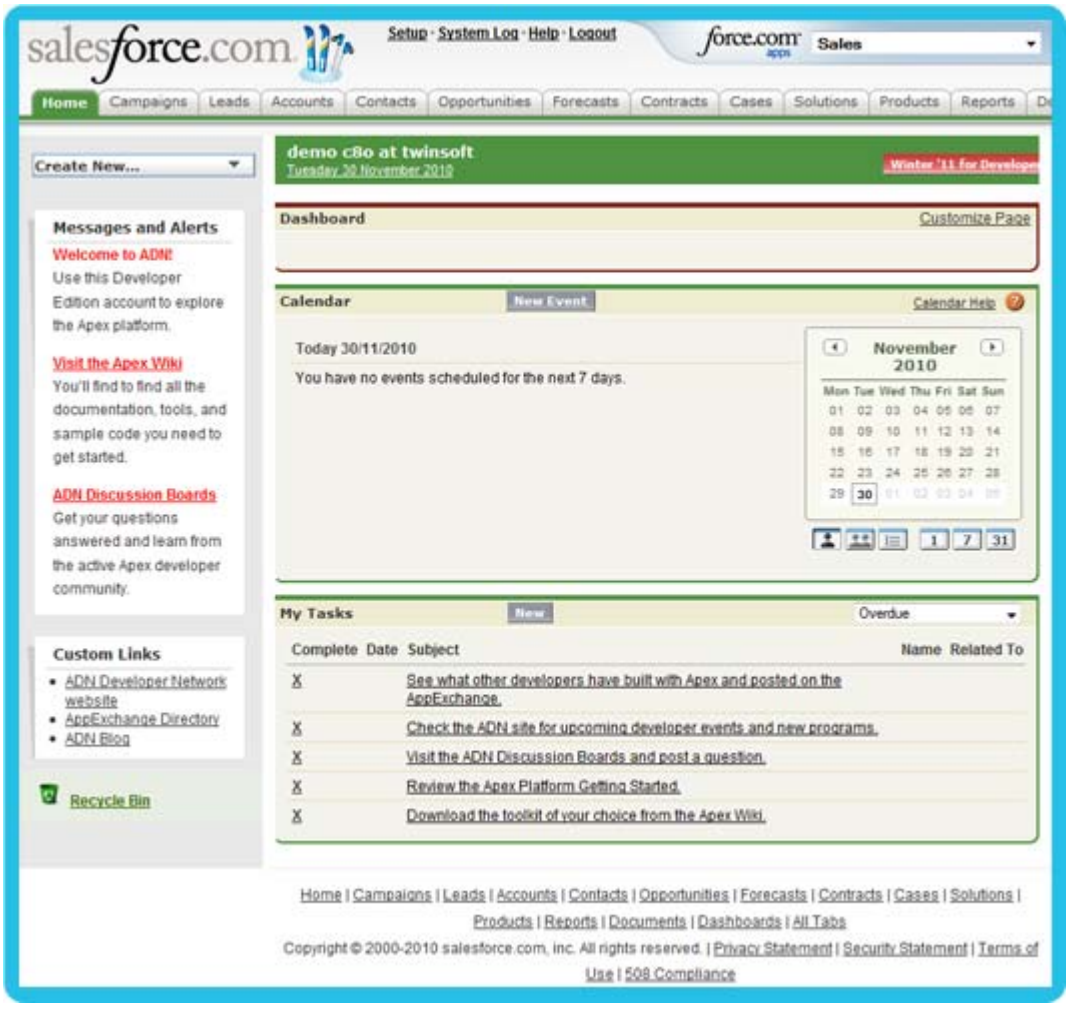


Figure 2 - 306: Default entry handler - SalesForce website home page

A `GoLeads` transaction, which defines a variable named `viewName`, is implemented to access to the list of leads, on the view named after the `viewName` variable value (which default value is `ConvertigoDemoView`)

In order to access to the list of leads in SalesForce, the **Leads** tab has to be selected. No matter on which screen class the transaction starts, the **Leads** tab is available for the user to click on it. A *Default entry handler* is then created in order to add the clicking on **Leads** tab action. It is created in the transaction, with the following parameters:

```
Default entry handler [
   result=redetect
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

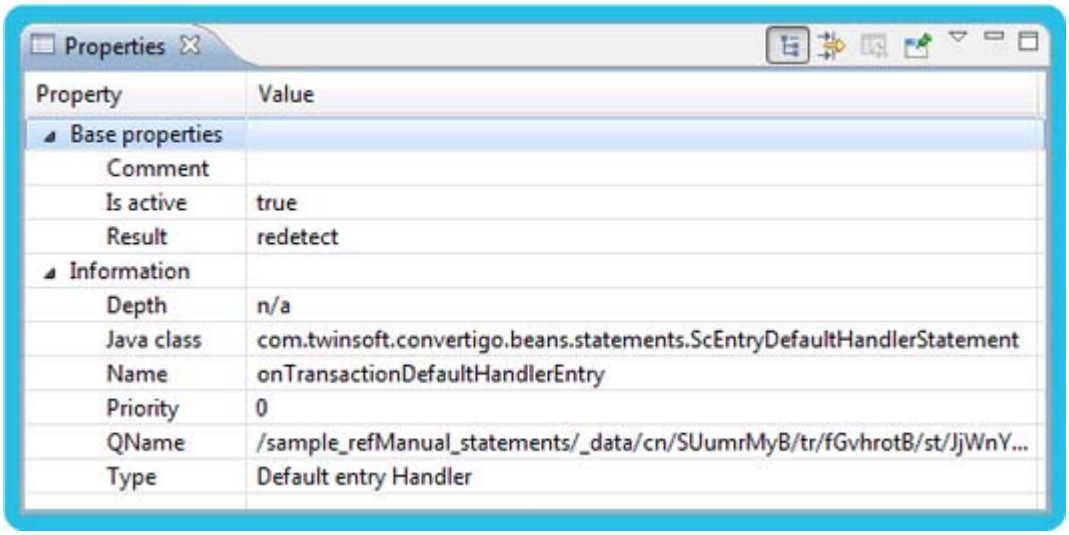


Figure 2 - 307: Default entry handler - Configuration example

The *Default entry handler* is created in the **Functions** folder of the transaction, including statements performing actions on the website, and appears as follows in the **Projects** view:

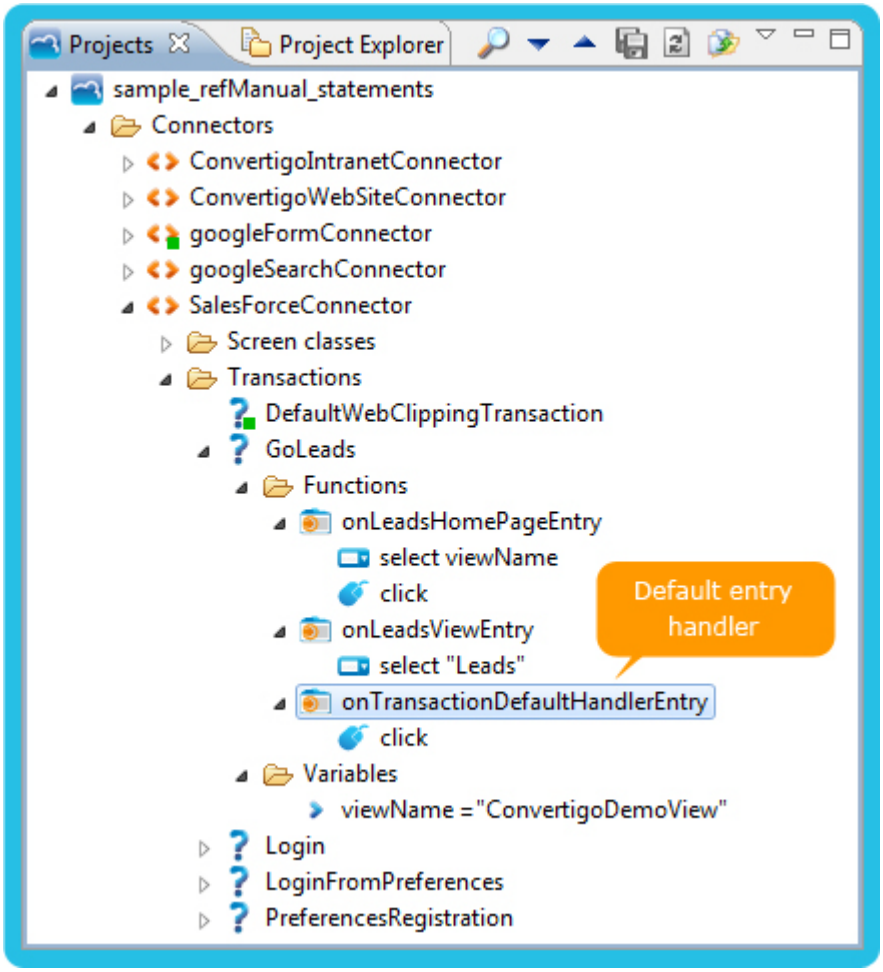


Figure 2 - 308: Default entry handler - Object in Projects view

When executing the `GoLeads` transaction after having executed the `Login` transaction, the transaction starts on the home page of SalesForce website. This page is associated with the `HomePage` screen class. As no specific screen class handler is defined for this screen class,

the *Default entry handler* is executed and the transaction accesses **Leads** tab.

DEFAULT EXIT HANDLER 

OBJECT DESCRIPTION

Defines a transaction default exit handler.

A *Handler* is similar to a `Function` except that it is automatically executed when the associated event occurs. The *Default exit handler* is an exit screen class handler, associated with an exit from a screen class event. If present in a transaction, it is executed when no specific screen class exit handler is defined for the currently detected screen class. In other words, it is a generic screen class exit handler.

Thus, *Default exit handler* can be defined for multiple screen classes on which the programmer knows that the same actions are to be done.

Beware that this handler will be executed for the `Default_screen_class` screen class if no specific handler is defined for this screen class. As the `Default_screen_class` screen class matches every page that is not defined in the connector, this handler can potentially be executed on a lot of unmanaged pages. Such behaviors can lead to infinite loop transactions.

Notes:

- A *Default exit handler* contains other statements that define the actions to be performed. It can return a result value (`accumulate` by default as it is an exit handler).
- *Default exit handlers* can only be added to a transaction, one *Default exit handler* per transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Infinite loop protection	boolean	standard	Defines whether the handler should be protected against infinite loops in transaction. If set to `true` (default value), the handler prevents infinite loops by throwing a Convertigo Engine exception when an infinite loop is detected. Default value should not be changed unless you specifically want the handler to authorize loops in transaction.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Result	String	standard	Defines the handler's default resulting value. Depending on the handler type, this property can be chosen among several available values. For a *Default exit handler*, this property can take the following values: • `<empty>` or `""`: stops the process and ends the transaction, • `continue`: similar to `<empty>` value, • `accumulate`: accumulates extracted data (data is extracted from last detected screen class then added to any other extracted data) and redetects a new screen class. Note: The *Handler*'s default return value defined thanks to this property can be overridden by a child *Return* statement.

FUNCTION 

OBJECT DESCRIPTION

Defines a function that can be invoked.

A *Function* statement declares a new function for a transaction. It contains other statements defining actions to be performed by the function. It can only be added under a transaction.

The *Functions* defined for a transaction can then be called from one or several handlers using *Call Function* statements (to perform the same actions on several screen classes for example).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let's consider the US directory website. Every page of this site contains a search form on its top:

Figure 2 - 309: Function statement - US directory website pages with search form

A `searchBusiness` transaction, which defines three variables named `business`, `cityZIPcode`, and `state`, sets those three values into search form input fields and launch a search by clicking on the `Search` button.

As the transaction must be able to fill the search form on every page of the US directory website, it has to define screen class handlers for each existing screen class of the connector. But, the actions to perform on each screen class would be the same:

- set Business input,
- set City / ZIP code input,
- select State in combobox,
- click on the `Search` button.

To avoid defining three times the same statements in different screen class handlers, a *Function* statement, named `setSearchInputs`, is created in the transaction to define once the common actions filling the form. This statement doesn't define specific parameters:

```
Function [
]
```

It is created in the **Functions** folder of the transaction, next to Screen class handlers and contains statements used to implement the transaction behavior described above. It appears as follows in the **Projects** view:

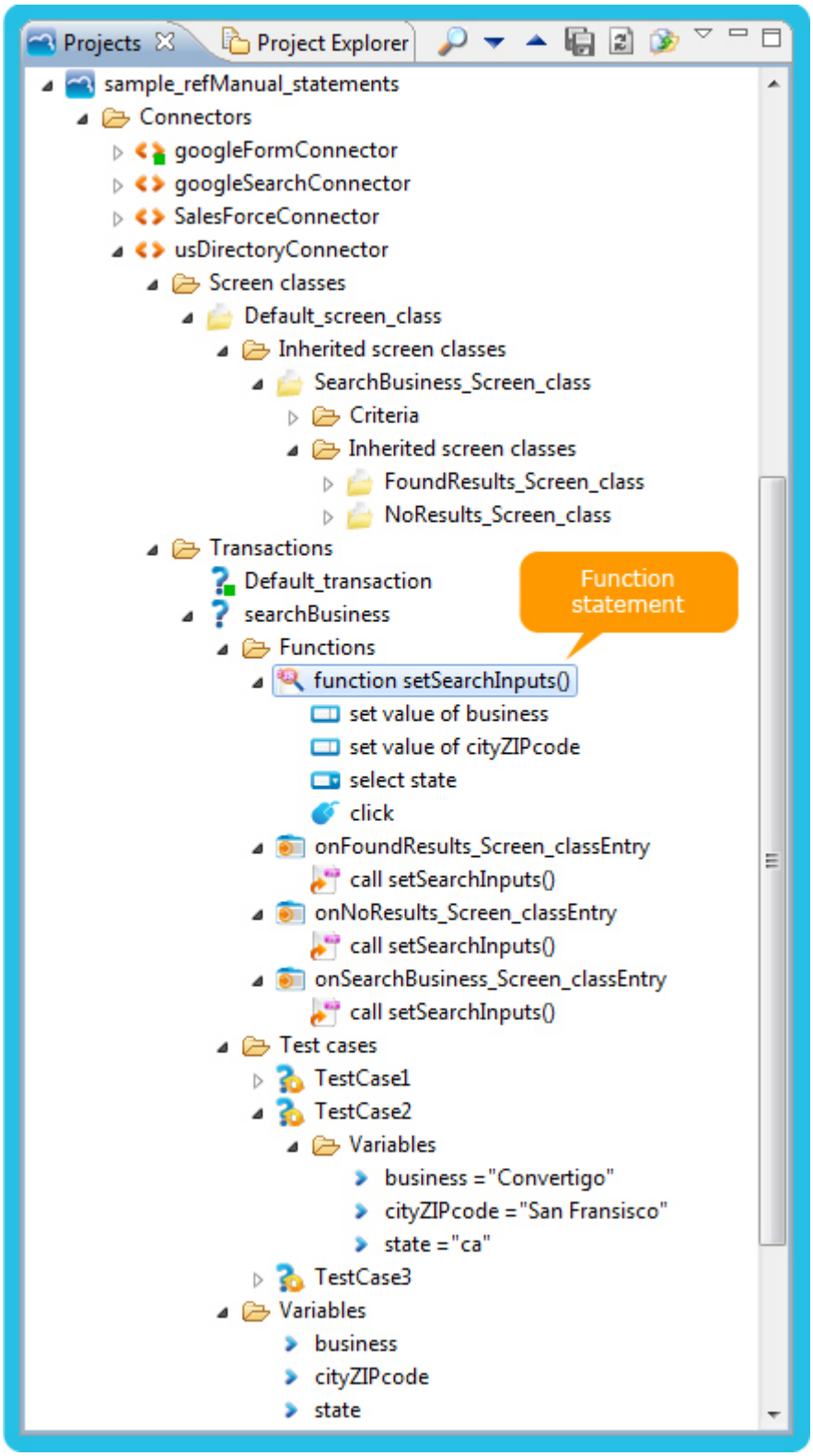


Figure 2 - 310: Function statement - Object in Projects view

This common *Function* is then called in every screen class handler thanks to *Call Function* statements. For more information about those statements, see the documentation and

example of *Call Function* statement object.

When executing test cases defined for the transaction, starting from every page from US directory website, the transaction does the same actions (described above) and searches the business.

FLOW CONTROL STATEMENTS

CONTAINER (WEB) 

OBJECT DESCRIPTION

Defines a statement able to contain other statements.

The *Container* statement can contain a list of statements that have to be grouped. It has no effects on the statements execution order.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let's consider a Convertigo user developping a complex transaction, for example the `searchGoogleWithLimit` transaction set in the context of the "*Starting With Convertigo Web Integrator*" *Quick Guide*. This transaction, which defines two variables named `keyword` and `maxPages`, searches for the keyword in Google search engine and accumulates the results of `maxPages` pages into an XML structure thanks to a *Table* extraction rule.

Several implementations are possible to manage the limited number of result pages to browse. Two of them are:

- decrementing the maxPages variable and compare it with zero,
- incrementing a new counter variable and compare it to maxPages variable.

In order to test different implementations of the same actions, it can be useful to create *Container* statements grouping statements of the same implementation together. These containers can then be disabled to easily test one solution or the other.

Container statements don't define specific parameters:

```
Container [
]
```

They are created in the **Functions** folder of the transaction, under the corresponding Screen class handlers and contain various other statements used to implement the transaction behavior. They appear as follows in the **Projects** view, enabled or disabled:

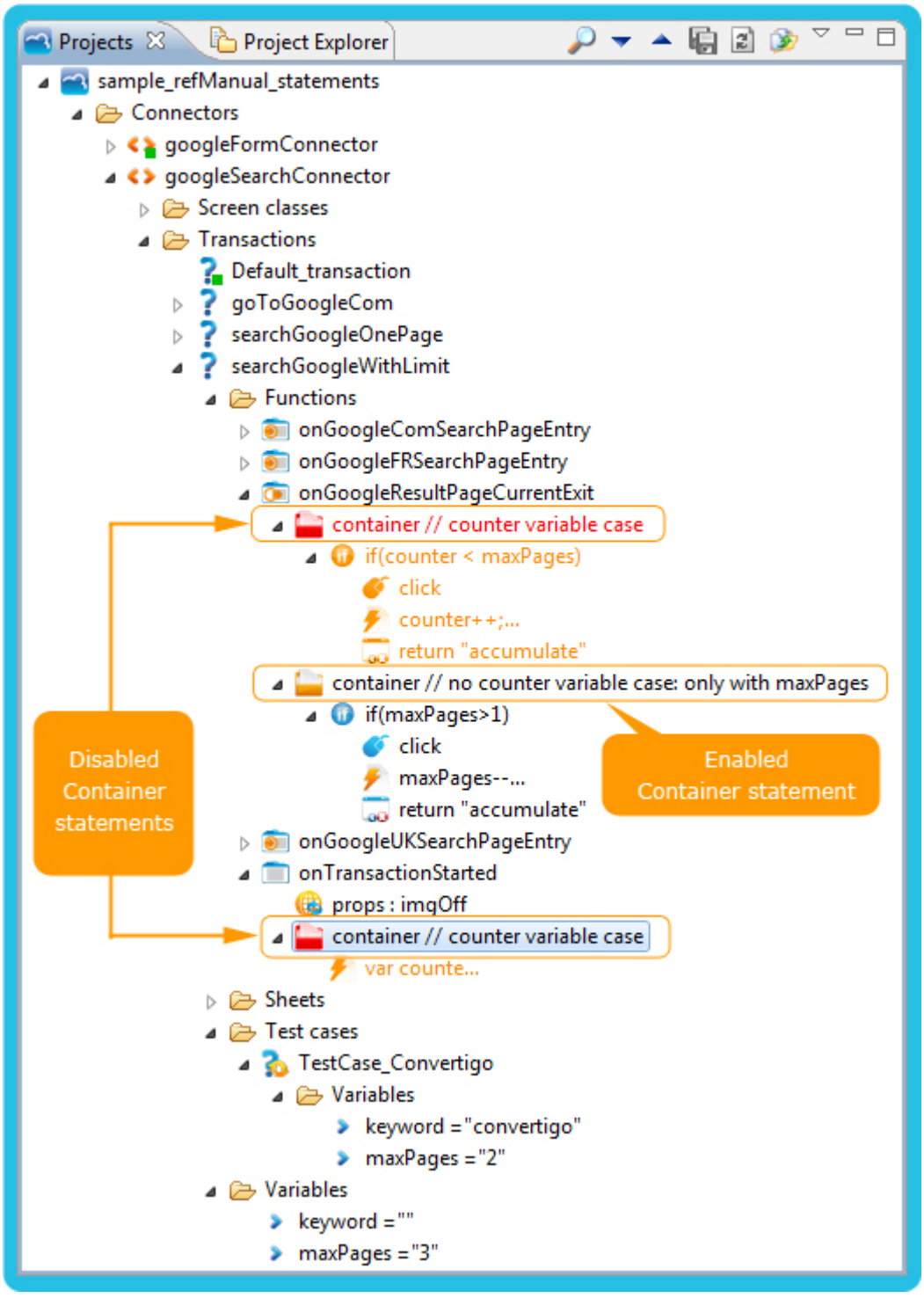


Figure 2 - 311: Container statement - Enabled and disabled objects in Projects view

When disabling one or the two other *Container* statements, all child statements of disabled *Container* are not reachable, meaning that they won't be executed when the transaction runs. Thanks to diabling one container or the two others, executing the test case defined for the transaction will test one or the other implementation.

IF 

Object Description

Defines an `IF` conditional statement based on a JavaScript condition.

The *If* statement is one of the *HTML transaction* conditional statements. It conditionally executes a block of statements, depending on the fulfillment of a condition expression. In other words, if the condition is fulfilled, child statements are executed.

The condition, set in the **Condition** property, is a JavaScript expression that is evaluated during the transaction execution as `true` or `false`.

Note: In Convertigo Studio, when an *If* statement is created in a handler, it can be easily replaced by an *IfThenElse*, using the right-click menu on the statement and choosing the option **Change to** > **IfThenElse**. The **Condition** property remains the same and the statements present in the *If* are moved to the *Then* sub-statement.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child statements.
Is active	boolean	standard	Defines whether the statement is active.

Examples

Let's consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

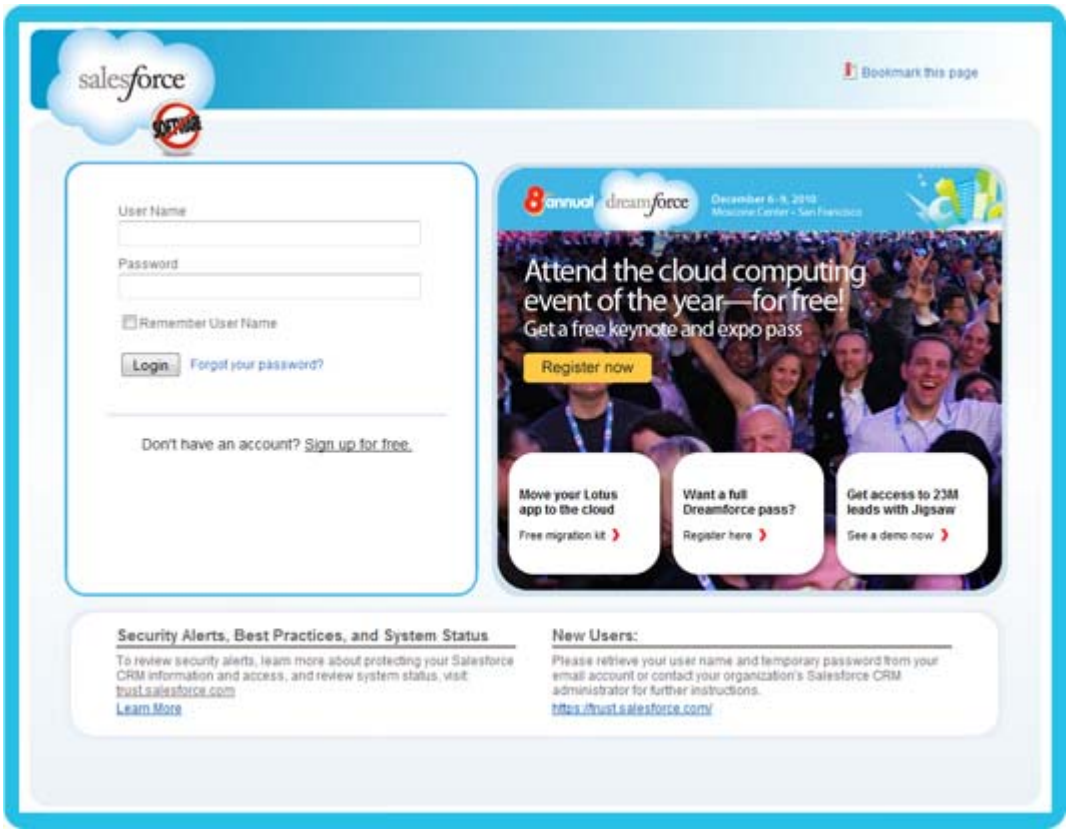


Figure 2 - 312: If statement - SalesForce website authentication page

A `Login` transaction, which defines two variables named `username` and `password`, is written to authenticate the user on SalesForce website. This transaction:

- fills the User Name and Password fields using the `user` / `password` variables,
- submits the authentication form to access to SalesForce welcome page.

When the authentication is not correct, SalesForce website displays an error message, recognized by Convertigo as the `LoginFailedPage` screen class. In this case, and when the username / password is not provided, the transaction raises a Convertigo Engine Exception.

To detect when username / password is not provided, an *If* statement is created in the transaction, with the following parameters:

```
If [
  condition: username == null || username == ""
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

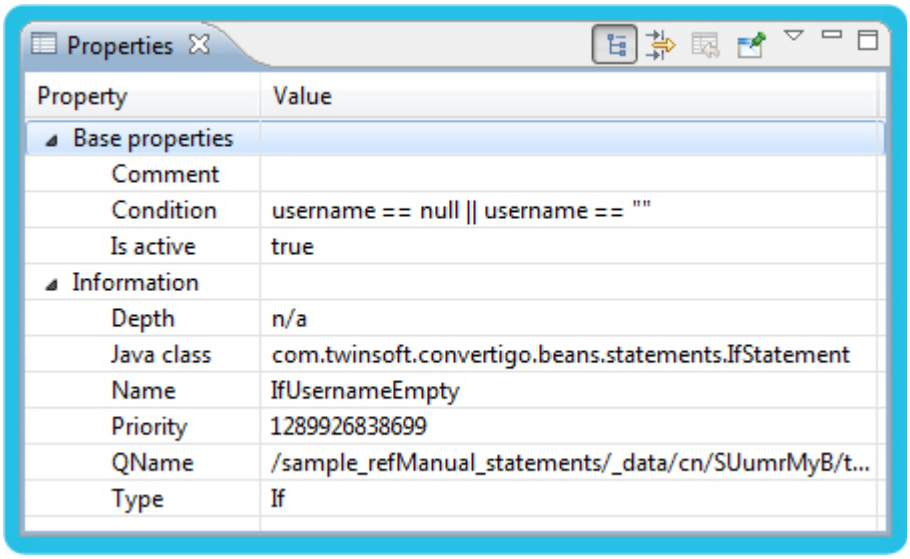


Figure 2 - 313: If statement - Configuration example

The **Condition** property is set to a complex JavaScript expression that tests at once if the `username` variable value is not provided (`==null`) or if it equals an empty string (`=""`).

The statement is created in the **Functions** folder of the transaction, under the transaction start event handler and contains other statements such as an *Exception* statement, and appears as follows in the **Projects** view:

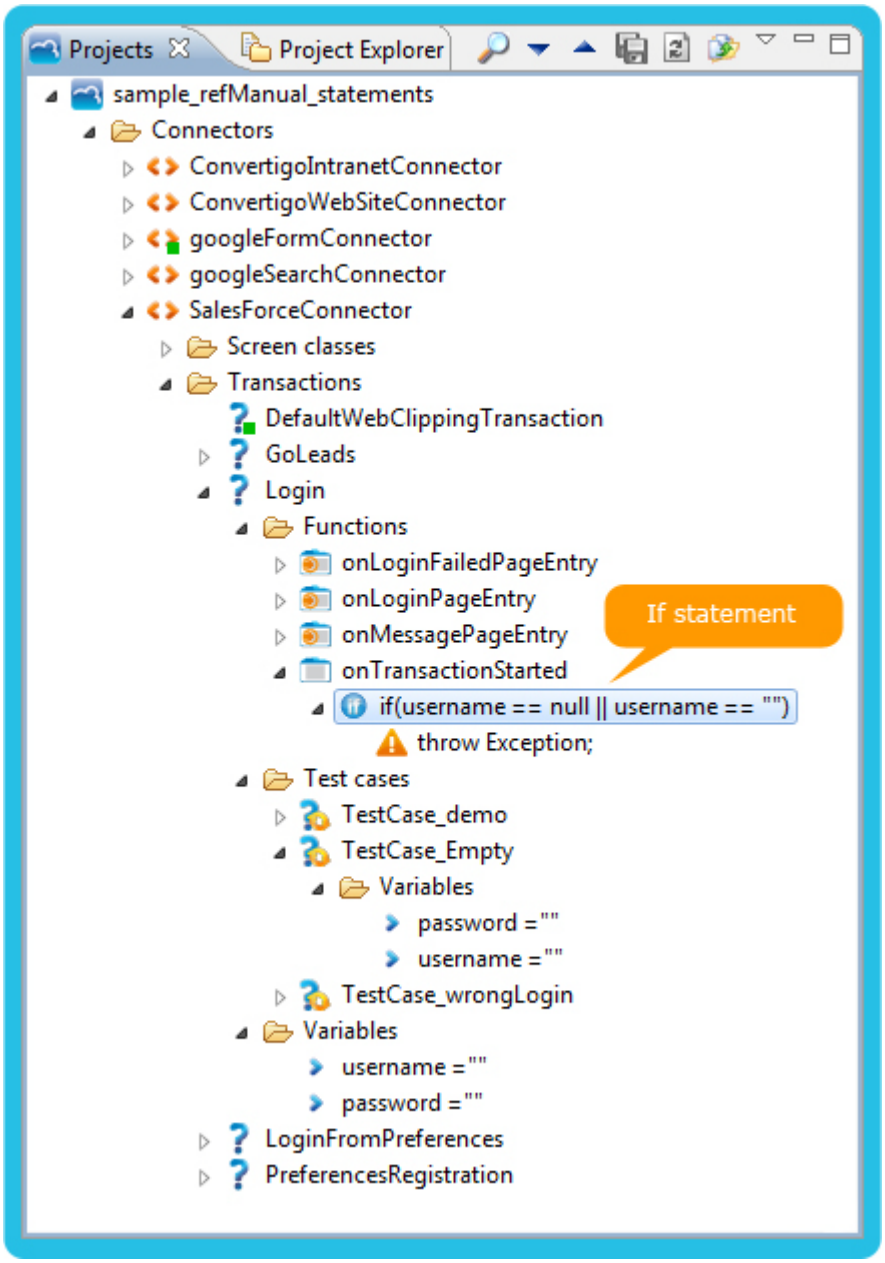


Figure 2 - 314: If statement - Object in Projects view

When executing the test cases named `TestCase_Empty` defined for the `Login` transaction, the transaction raises an Engine Exception by executing the *Exception* statement, as the condition of the *If* statement is evaluated to `true`.

IfThenElse 

Object Description

Defines an `IF...THEN...ELSE...` conditional statement based on a JavaScript condition.

The *IfThenElse* statement is one of the *HTML transaction* conditional statements. It contains two child steps (*Then* and *Else*) which are executed depending on the condition fulfillment:

- `Then` step and child steps are executed when the condition is verified,
- `Else` step and child steps are executed when the condition is not verified.

The condition, defined in the **Condition** property, is a JavaScript expression that is evaluated during the transaction execution as `true` or `false`.

Note: In Convertigo Studio, when an *IfThenElse* statement is created in a handler, it can be easily replaced by an *If*, using the right-click menu on the statement and choosing the option **Change to** > **If**. The **Condition** property remains the same and the statements present in the sub-statements are:

- statements present in the *Then* statement are moved to the *If*,
- statements present in the *Else* statement are deleted.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child statements.
Is active	boolean	standard	Defines whether the statement is active.

While

Object Description

Defines a `WHILE` loop statement based on a JavaScript condition.

This statement executes a group of child statements until the condition expression set in the **Condition** property is found to be false.

Note: You can add other statements to this statement: these are the statements executed in the loop.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child statements.
Is active	boolean	standard	Defines whether the statement is active.

Examples

Let's consider a `fillForm` transaction, which defines one multivaluated variable named `inputs`. This transaction sets input field values into a web page containing a `FORM` HTML element.

As the transaction doesn't know in advance the number of inputs to fill in the `FORM`, it is implemented to loop on each element of the `inputs` variable, dynamically received from the caller, and to set each value in the nth HTML `INPUT` element of the web page. When there is no more `INPUT` element to fill, the transaction exits the loop and ends.

In order to loop on each element of the `inputs` variable, a *While* statement is created in the `fillForm` transaction with the following parameters:

```
While [
   condition: iterator < inputs.length
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

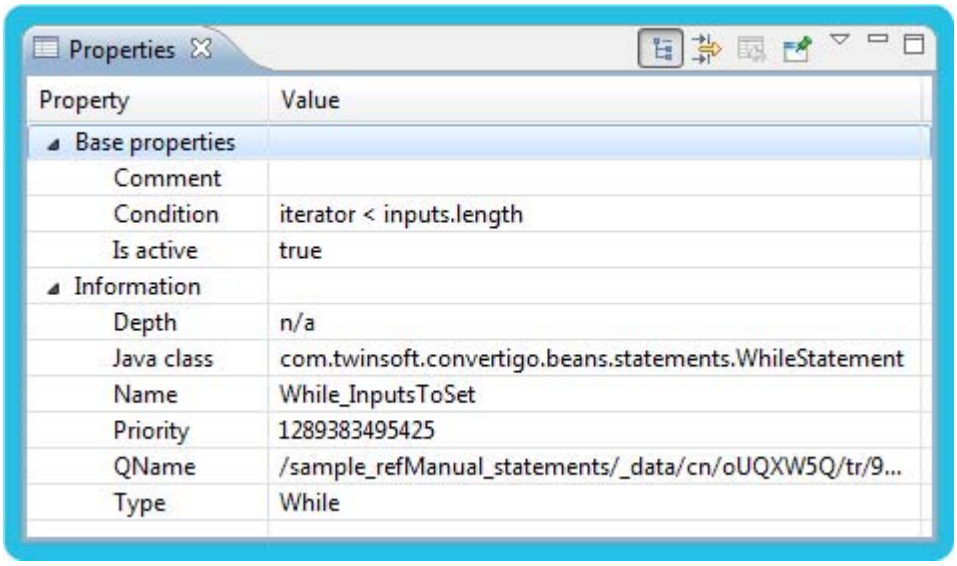


Figure 2 - 315: While statement - Configuration example

The **Condition** property is set to a JavaScript expression testing the value of the counter variable, named `iterator`, against the length of the `inputs` variable.

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler cand contains other statements used to implement the transaction behavior described above. It appears as follows in the **Projects** view:

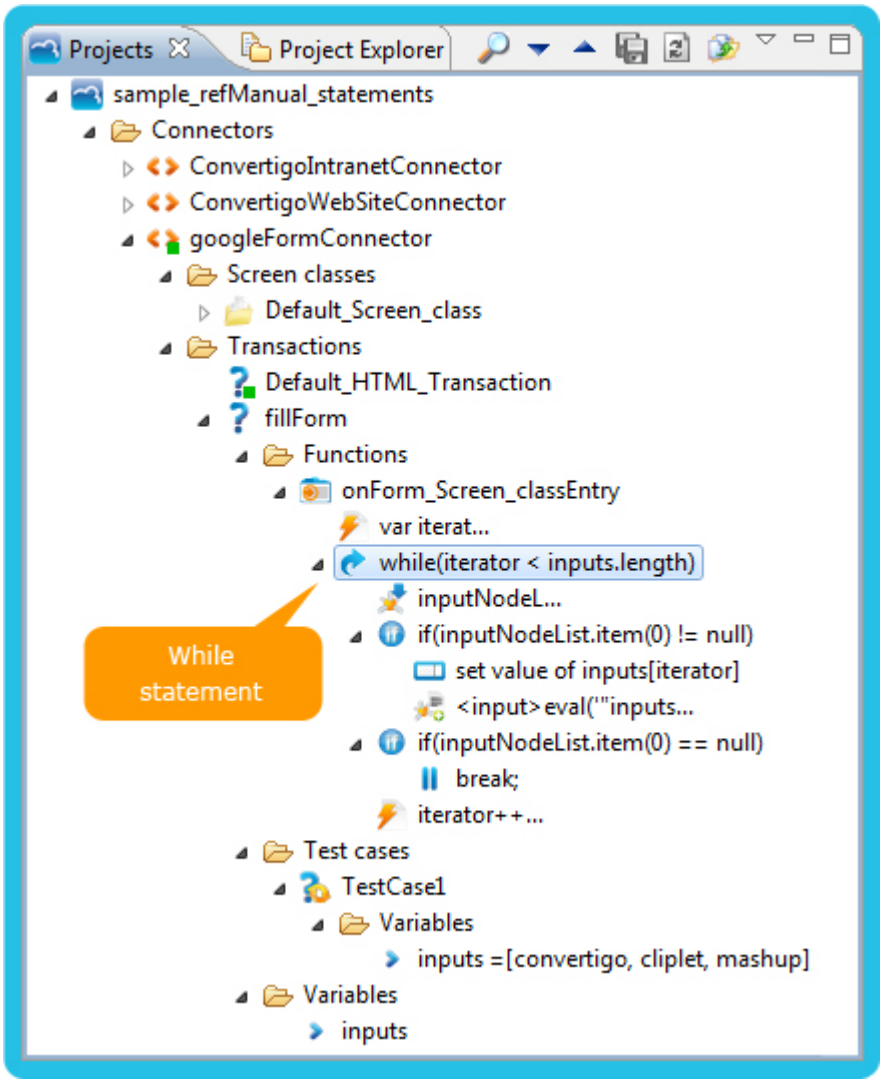


Figure 2 - 316: While statement - Object in Projects view

The test case defined for the transaction contains three values in `inputs` variable. When executing it on Google search page, containing a `FORM` element with only one `INPUT` to fill, the transaction sets the first value of `inputs` variable into the `INPUT` element and exists the loop as no `INPUT` element is found for the next value.

Executing it on a web page that contains more than three `INPUT` elements to fill, the transaction will stop after three loops because the `iterator` variable value will be bigger than the `inputs` variable length.

DO WHILE 

OBJECT DESCRIPTION

Defines a `DO...WHILE` loop statement based on a JavaScript condition.

This statement executes a group of child statements once, then repeats execution of the loop until the condition expression set in the **Condition** property is found to be false.

Note: You can add other statements to this statement: these are the statements executed in the loop.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Condition	JS expression	standard	Defines the block condition expression. This property is a JavaScript expression that will be evaluated as condition (`true` or `false`) in order to decide whether to execute or not the child statements.
Is active	boolean	standard	Defines whether the statement is active.

RETURN (WEB) 

OBJECT DESCRIPTION

Defines a RETURN statement.

A *Return* statement exits from the current function or handler and returns a value from it. The returned value is specified in the **Expression** property as a JavaScript expression evaluated during the transaction execution.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the statement value. This property is a JavaScript expression that is evaluated during the transaction execution and gives the statement's result.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let’s consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

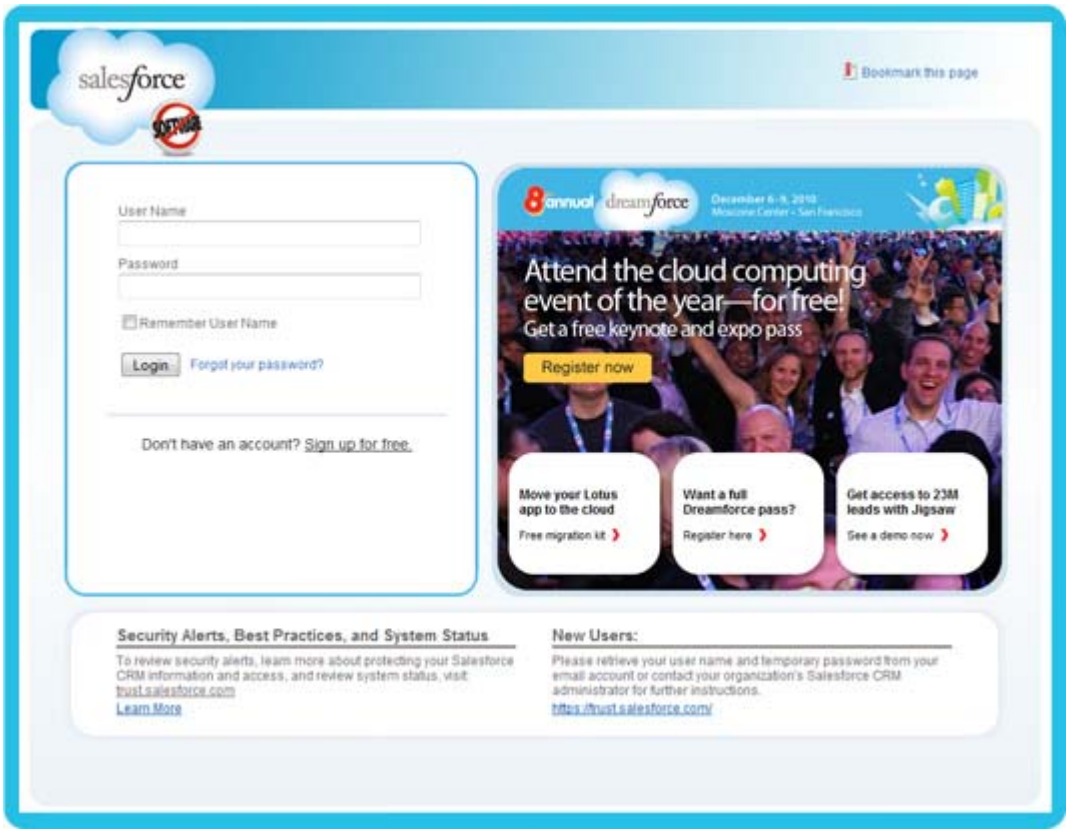


Figure 2 - 317: Return statement - SalesForce website authentication page

In the context of a mashup application including a SalesForce widget, we want the user to be able to customize its user / password. A `PreferencesRegistration` transaction, which defines two variables named `username` and `password`, allows to register in Convertigo context the user authentication values.

Then, a `LoginFromPreferences` transaction, which does not define any variable, authenticates the user on SalesForce website using the previously saved username and password.

When the user authentication values are not previously registered in the Convertigo context, case detected thanks to an *If* statement testing the retrieved values, the transaction is aborted before connecting to SalesForce website.

To do so, a *Return* statement is created with the following parameters:

```
Return [
   expression: "cancel"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

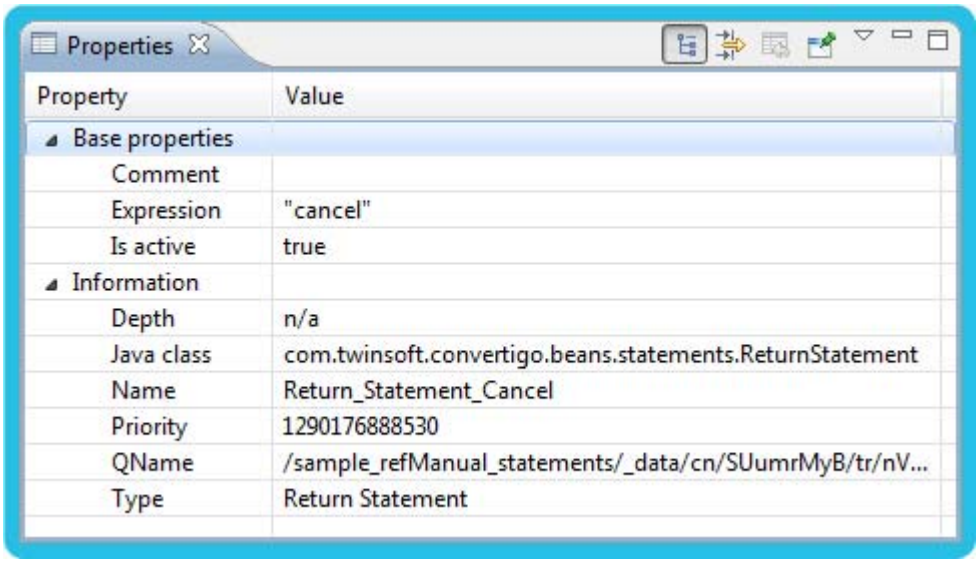


Figure 2 - 318: Return statement - Configuration example

The *Return* statement is created in the **Functions** folder of the transaction, in the transaction start handler and appears as follows in the **Projects** view:

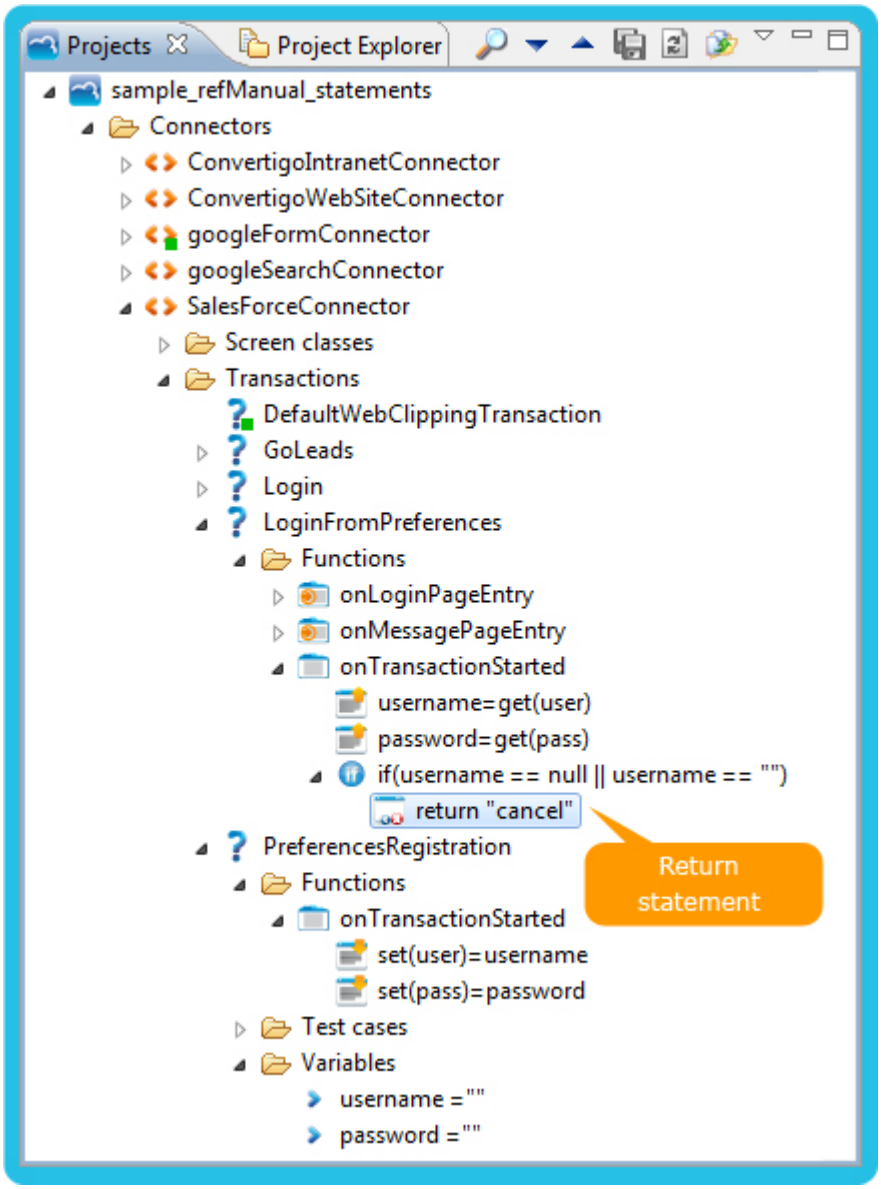


Figure 2 - 319: Return statement - Object in Projects view

We can observe that the default result value of the start transaction handler is empty, it is visible in the handler properties:

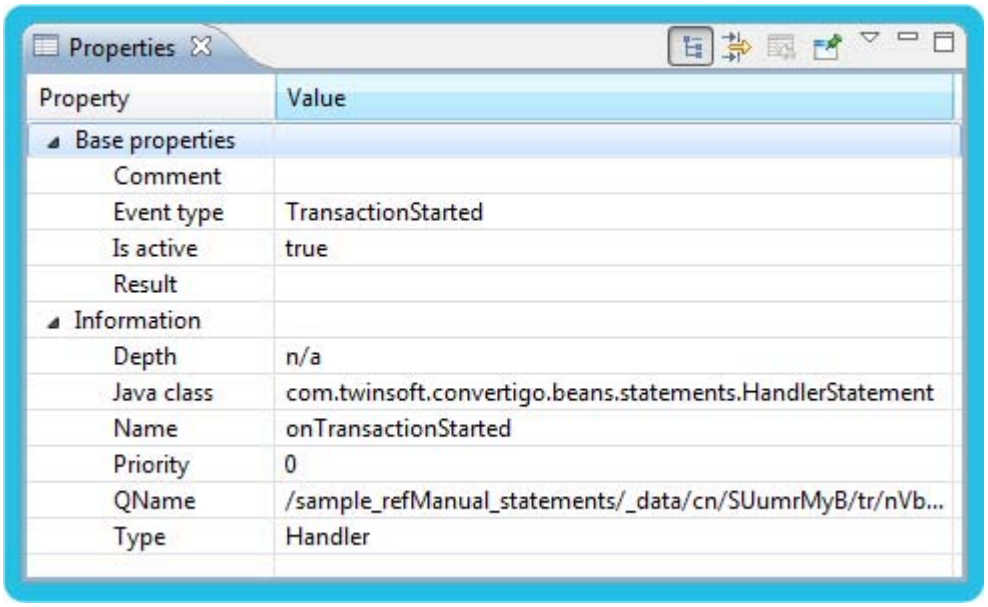


Figure 2 - 320: Return statement - Start transaction handler properties

This result value is overridden by the *Return* statement when the username is tested `null` or empty.

BREAK 

OBJECT DESCRIPTION

Defines a BREAK statement.

A *Break* statement terminates the current loop statement.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the statement value. This property is a JavaScript expression that is evaluated during the transaction execution and gives the statement's result.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let's consider a `fillForm` transaction, which defines one multivaluated variable called `inputs`. This transaction sets input field values into a web page containing a `FORM` HTML element.

As the transaction doesn't know in advance the number of inputs to fill in the `FORM`, it is implemented to loop on each element of the `inputs` variable, dynamically received from the caller, and to set each value in the nth HTML `INPUT` element of the web page. But when there is no more `INPUT` element to fill, the transaction must exit the loop and end.

To do so, a *Break* statement is created in case no nth `INPUT` element is found. This statement doesn't define specific parameters:

```
Break [
]
```

It is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and various other statements used to implement the transaction behavior described above. It appears as follows in the **Projects** view:

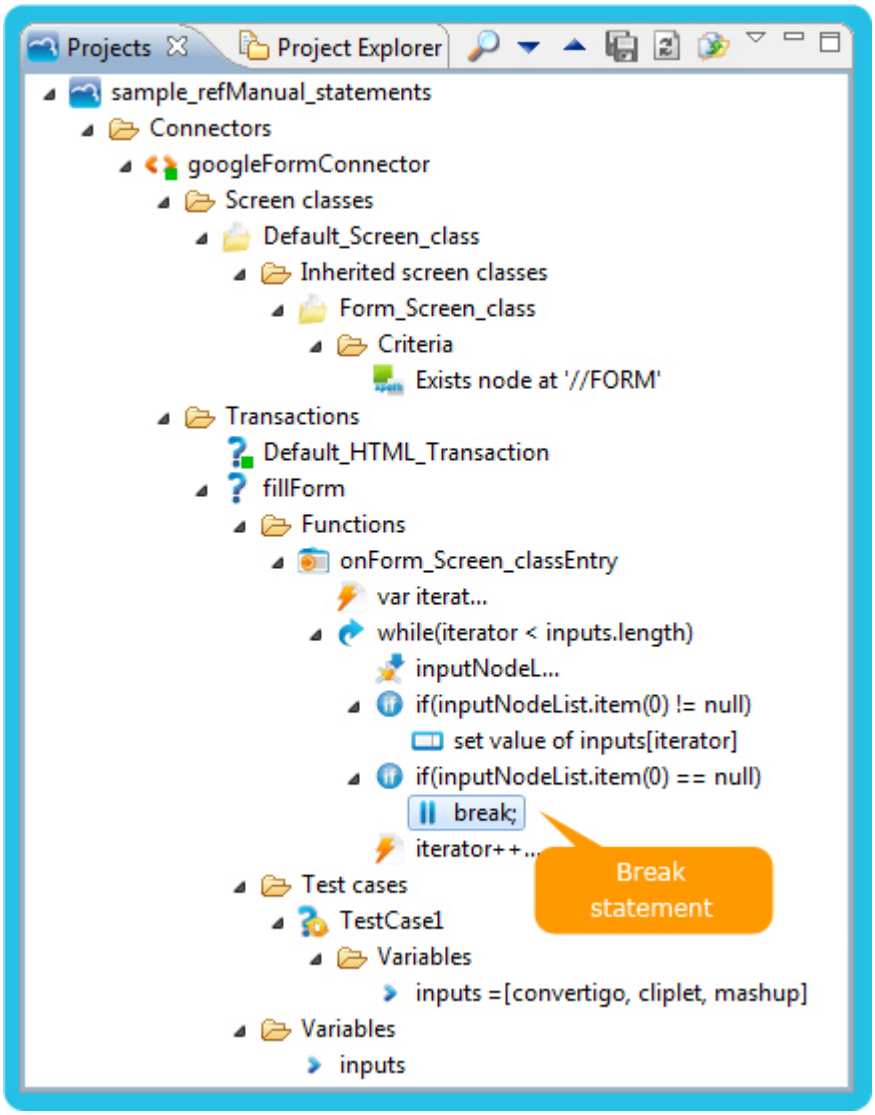


Figure 2 - 321: Break statement - Object in Projects view

The test case defined for the transaction contains three values in `inputs` variable. When executing it on Google search page, containing a `FORM` element with only one `INPUT` to fill, the transaction:

- sets the first value of `inputs` variable into the `INPUT` element of the web page,
- exists the loop thanks to the *Break* statement configured when no `INPUT` element is found for the next value.

Figure 2 - 322: Break statement - Executing fillForm transaction on Google search page

CALL FUNCTION 

OBJECT DESCRIPTION

Call any *Function* statement defined in the same transaction.

If *Function* statements have been defined for a given transaction, you can call them by setting a *Call function* statement in any handler of the given transaction.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Function name	String	standard	Defines the name of the function to call. This property allows to choose the *Function* statement to call by selecting it in a list of the available ones for the current transaction.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let's consider the US directory website. Every page of this site contains a search form on its top:

Figure 2 - 323: Call function statement - US directory website pages with search form

A `searchBusiness` transaction, which defines three variables named `business`, `cityZIPcode`, and `state`, sets those three values into search form input fields and launch a search by clicking on the `Search` button.

As the transaction must be able to fill the search form on every page of the US directory website, it has to define screen class handlers for each existing screen class of the connector. But, the actions to perform on each screen class would be the same:

- set Business input,
- set City / ZIP code input,
- select State in combobox,
- click on the `Search` button.

To avoid defining three times the same statements in different screen class handlers, a *Function* statement, named `setSearchInputs`, is created in the transaction to define once the common actions filling the form. For more information about *Function* statement, see the documentation and example of *Function* statement object.

To call this common *Function* in every screen class handler, three identical *Call function*

statements are created with the following parameter:

```
Call function [
   function name=setSearchInputs
]
```

They are created in the **Functions** folder of the transaction, under corresponding Screen class handlers. They appear as follows in the **Projects** view:

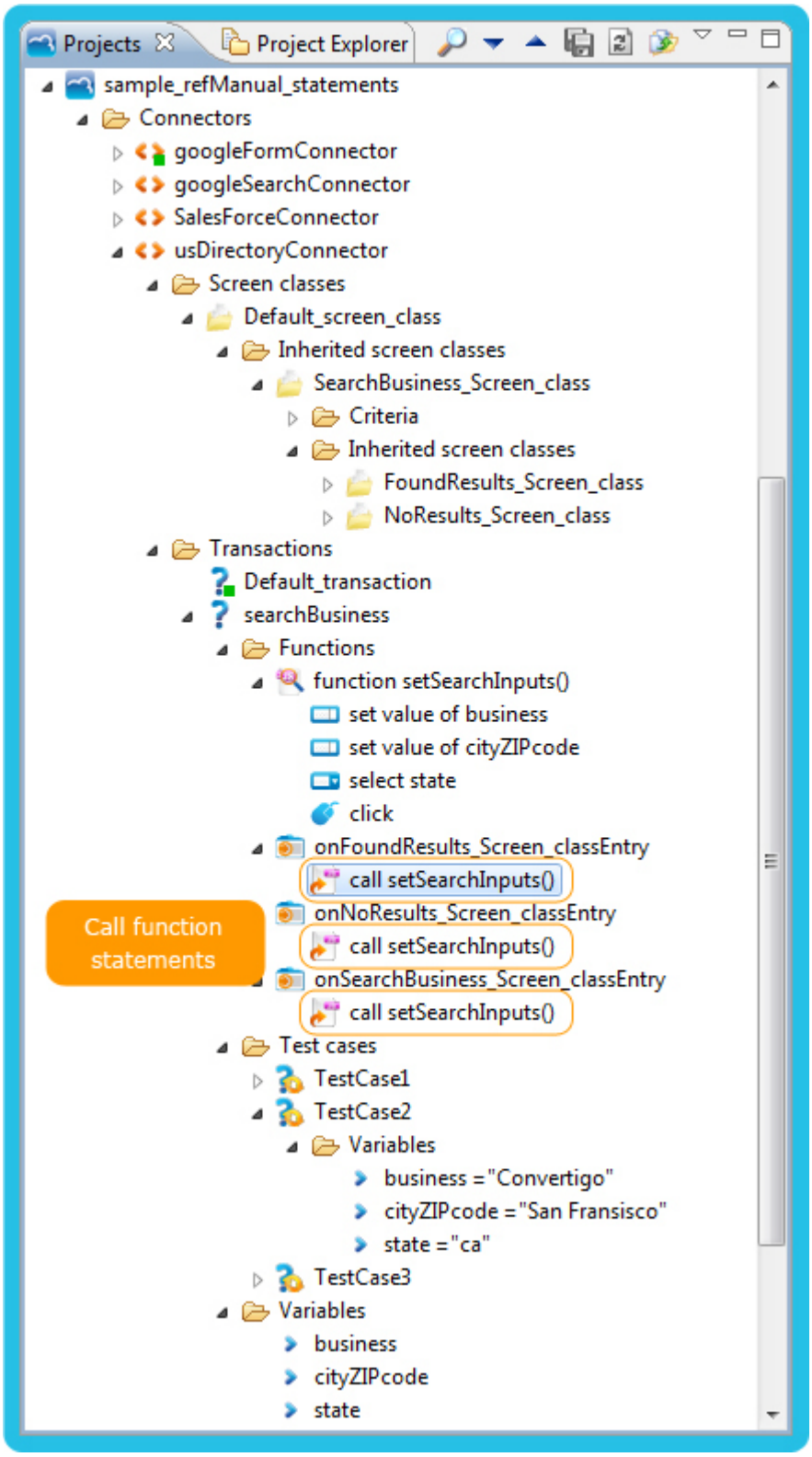


Figure 2 - 324: Call function statement - Objects in Projects view

The one parameter of each *Call function* statement is edited in the **Properties** view of the Convertigo Studio:

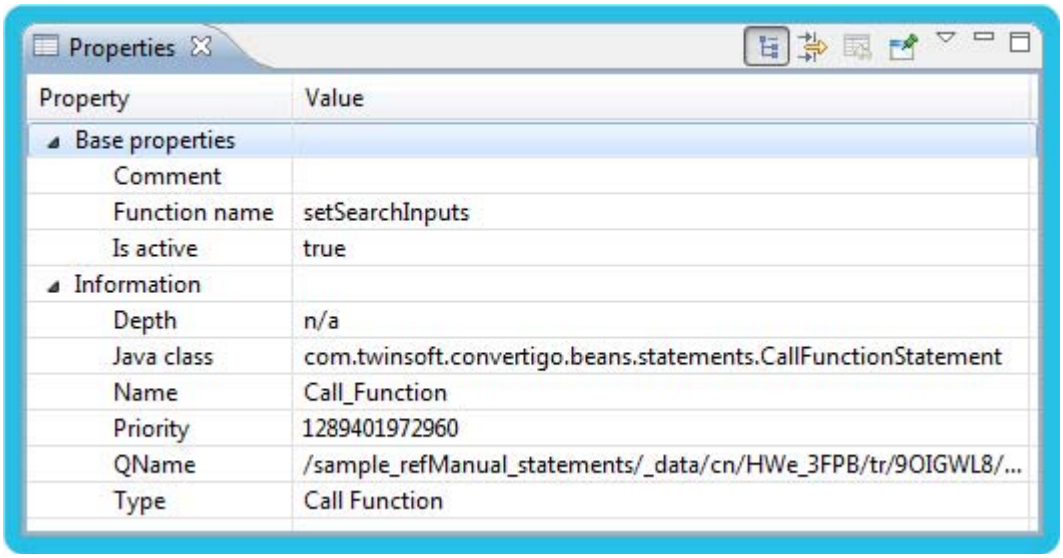


Figure 2 - 325: Call function statement - Configuration example

When executing test cases defined for the transaction, starting from any page of US directory website, the transaction does the same actions described above.

IfXpathExists 

Object Description

Defines an `IF` conditional statement looking for node(s) on a web page.

The *IfXpathExists* statement is one of the *HTML transaction* conditional statements. It conditionally executes a block of statements, depending on the fulfillment of a condition expression. In other words, if the condition is fulfilled, child statements are executed.

The condition is the existence in the current web page of nodes matching the XPath defined through the **XPath** property.

Note: In Convertigo Studio, when an *IfXpathExists* statement is created in a handler, it can be easily replaced by an *IfXpathExistsThenElse*, using the right-click menu on the statement and choosing the option **Change to** > **IfXpathExistsThenElse**. The **XPath** property remains the same and the statements present in the *IfXpathExists* are moved to the *Then* sub-statement.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
XPath	JS expression	standard	Defines the XPath expression to test. This property is a JavaScript expression that is evaluated during the transaction execution as an XPath. The execution of this **XPath** on the web page DOM is used as a condition in order to decide whether to execute or not the child statements: • `true` if one or several nodes are matching the XPath, • `false` if no node matches.

IfXpathExistsThenElse 

Object Description

Defines an `IF...THEN...ELSE...` conditional statement looking for node(s) on a web page.

The *IfXpathExistsThenElse* statement is one of the *HTML transaction* conditional statements. It contains two child steps (*Then* and *Else*) which are executed depending on the condition fulfillment:

- `Then` step and child steps are executed when the condition is verified,
- `Else` step and child steps are executed when the condition is not verified.

The condition is the existence in the current web page of nodes matching the XPath defined through the **XPath** property.

Note: In Convertigo Studio, when an *IfXpathExistsThenElse* statement is created in a handler, it can be easily replaced by an *IfXpathExists*, using the right-click menu on the statement and choosing the option **Change to** > **IfXpathExists**. The **XPath** property remains the same and the statements present in the sub-statements are:

- statements present in the *Then* statement are moved to the *IfXpathExists*,
- statements present in the *Else* statement are deleted.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
XPath	JS expression	standard	Defines the XPath expression to test. This property is a JavaScript expression that is evaluated during the transaction execution as an XPath. The execution of this **XPath** on the web page DOM is used as a condition in order to decide whether to execute or not the child statements: • `true` if one or several nodes are matching the XPath, • `false` if no node matches.

JAVASCRIPT STATEMENTS

TRANSACTION JS 

OBJECT DESCRIPTION

Defines a scripting statement.

This helpful statement allows to handle JavaScript code that will be executed in the transaction scope. This JavaScript code is able to:

- initialize variables,
- perform complex calculations,
- access the context object to get useful properties such as `contextID`, `httpSession`, `isCacheEnabled`, `lockPooledContext`, etc.,
- use some context methods to manipulate the result XML DOM (only in the transaction `XML Generated handler`), encode and decode data, abort transaction, etc.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the statement value. This property is a JavaScript expression that is evaluated during the transaction execution and gives the statement's result.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let's consider a `fillForm` transaction, which defines one multivaluated variable named `inputs`. This transaction sets input field values into a web page containing a `FORM` HTML element.

As the transaction doesn't know in advance the number of inputs to fill in the `FORM`, it is implemented to loop on each element of the `inputs` variable, dynamically received from the caller, and to set each value in the nth HTML `INPUT` element of the web page. When there is no more `INPUT` element to fill, the transaction exits the loop and ends.

In order to loop on each element of the `inputs` variable and find the corresponding nth `INPUT` element in the web page, a counter variable has to be declared before the loop and incremented in each loop. To do so, two *Transaction JS* statements are created in the `fillForm` transaction, one before the loop, one in the loop. These statements are created with the following parameters:

- for the declaration of the variable:

```
Transaction JS [
  expression: var iterator = 0;
]
```

- for the incrementation of the variable:

```
Transaction JS [
  expression: iterator++;
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

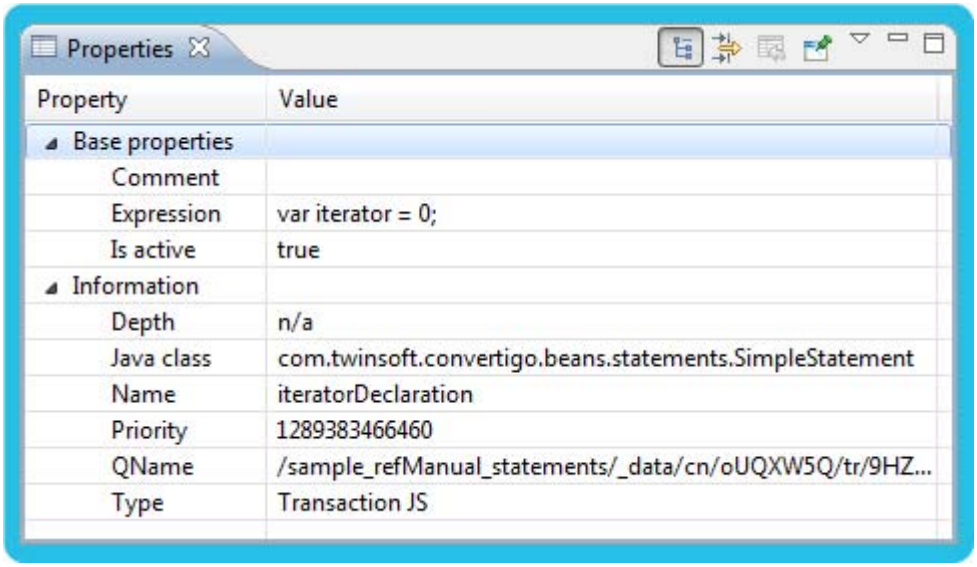


Figure 2 - 326: Transaction JS statement - Configuration example

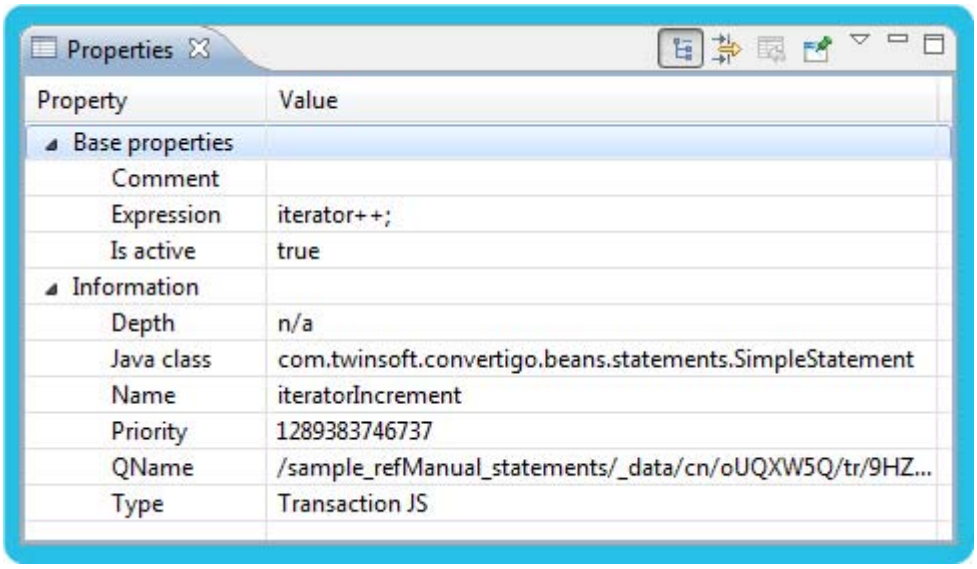


Figure 2 - 327: Transaction JS statement - Configuration example

The **Expression** property is set for both statements to a JavaScript expression representing the script to be executed.

The statements are created in the **Functions** folder of the transaction, under the corresponding Screen class handler and various other statements used to implement the transaction behavior described above. They appear as follows in the **Projects** view:

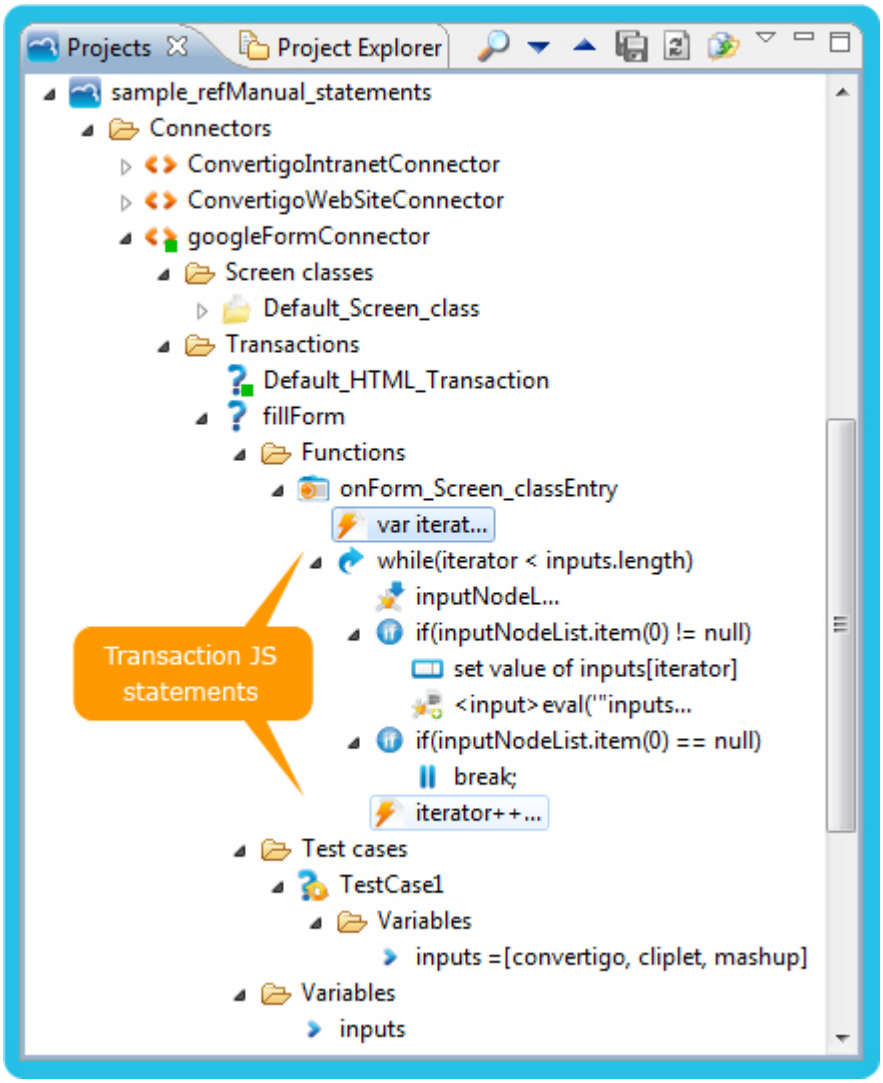


Figure 2 - 328: Transaction JS statement - Objects in Projects view

The test case defined for the transaction contains three values in `inputs` variable. When executing it on Google search page, containing a `FORM` element with only one `INPUT` to fill, the transaction sets the first value of `inputs` variable into the `INPUT` element and exists the loop as no `INPUT` element is found for the next value.

Executing it on a web page that contains more than three `INPUT` elements to fill, the transaction will stop after three loops because the `iterator` variable value will be bigger than the `inputs` variable length.

USER INPUT CONTROL STATEMENT

KEY ACTION 

OBJECT DESCRIPTION

Simulates a key action.

This statement enables Convertigo to send key events to any HTML element from the target web page.

The event can be one of the following types:

- `keypress`: simulates a `keypress` event,
- `keydown`: simulates a `keydown` event,
- `keyup`: simulates a `keyup` event.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	standard	Defines the JavaScript action corresponding to the event to perform. Depending of the statement, this property can take several values. These values are indicated in the object's description.
Character code	int	standard	Defines the ASCII character code for any alphanumeric key. This property allows setting the Unicode character associated with the depressed key. Otherwise, the value is zero and the key code value can be set in **Key code** property.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Key code	int	standard	Defines the key code. This property allows setting the virtual key code value to be sent, if the key has a key code value. Otherwise, the value is zero and the character code can be set in **Character code** property. For more information on key codes, see Appendix "*Keycodes table*".
Press alt key	boolean	standard	Defines whether the "`alt`" key is to be depressed when firing the event. On some platforms, this key may map to an alternative key name.
Press ctrl key	boolean	standard	Defines whether the "`ctrl`" key is to be depressed when firing the event.
Press meta key	boolean	standard	Defines whether the "`meta`" key is to be depressed when firing the event. On some platforms, this key may map to an alternative key name.
Press shift key	boolean	standard	Defines whether the "`shift`" key is to be depressed when firing the event.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

INPUT HTML SET VALUE 

OBJECT DESCRIPTION

Fills an HTML input field.

This statement enables Convertigo to set a value in a `text` type `input` HTML element.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Expression	JS expression	standard	Defines the expression evaluated to give the text to input. This property is a JavaScript expression that is evaluated during the transaction execution and gives the data to be set as input.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

UI event	boolean	standard	Defines whether UI events must be used to change the item value. Setting this property to `false` is saving CPU time, but sometimes the target website does not have the full behavior it has with a real user. In this case, setting this property to `true` will help having a behavior closer to the original site.
XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

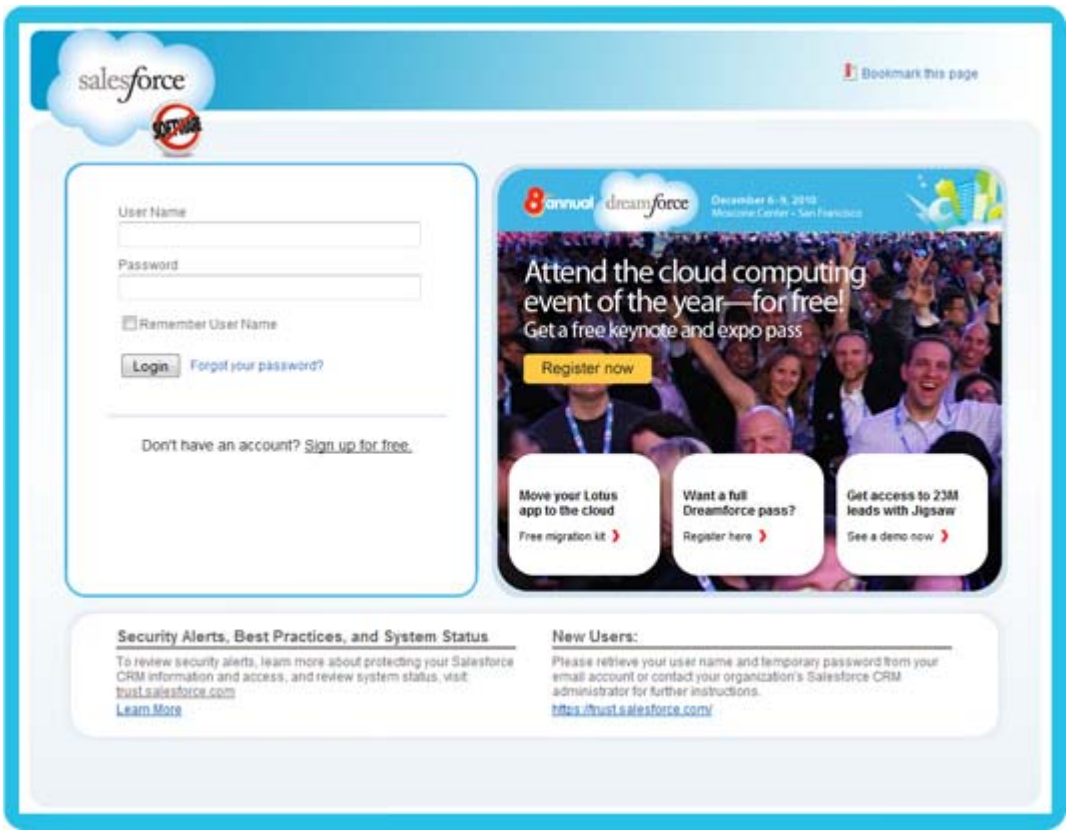


Figure 2 - 329: Input HTML set value statement - SalesForce website authentication page

A `Login` transaction, which defines two variables named `username` and `password`, is written to authenticate the user on SalesForce website.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

The `Login` transaction:

- fills the User Name and Password fields using the `user` / `password` variables,
- submits the authentication form to access to SalesForce welcome page.

When the authentication is not correct or when the username / password is not provided, the transaction raises a Convertigo Engine Exception.

In order to fill the User Name and Password fields, *Input HTML set value* statements are created in the transaction, with the following parameters:

- for the User Name field:

```
Input HTML set value [
  xpath='//INPUT[@id="username"]'
```

```
   expression: username
   uiEvent=false
   synchronization=[No Wait]
]
```

- for the Password field:

```
Input HTML set value [
   xpath='//INPUT[@id="password"]'
   expression=password
   uiEvent=false
   synchronization=[Wait time, timeout= 0 ms]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

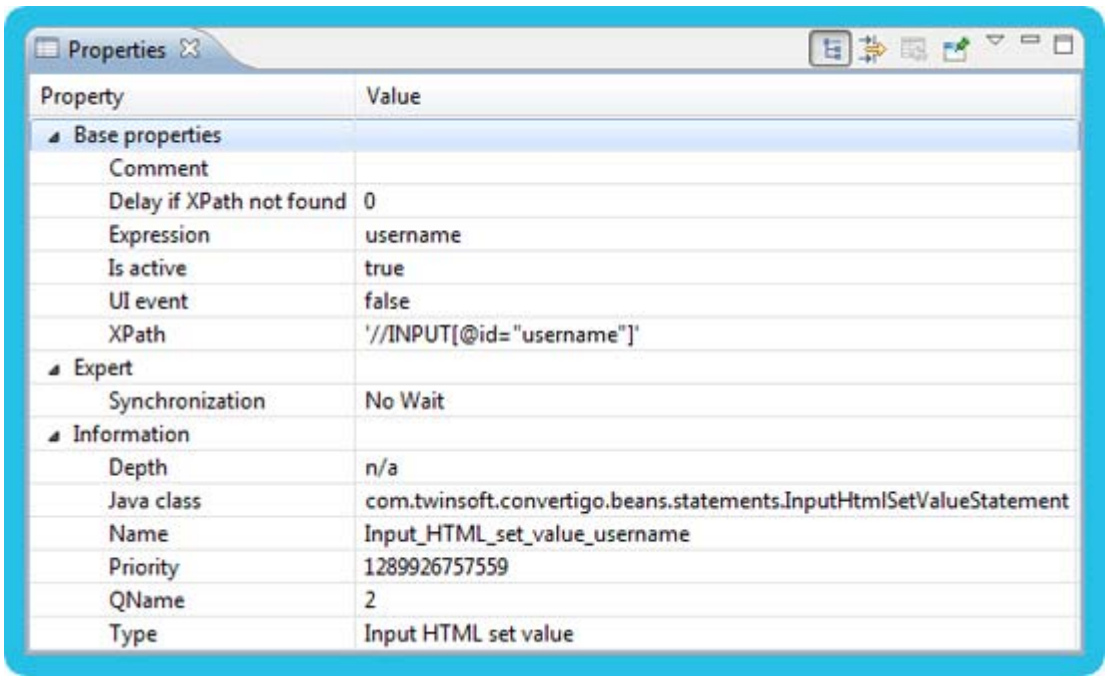


Figure 2 - 330: Input HTML set value statement - Configuration example

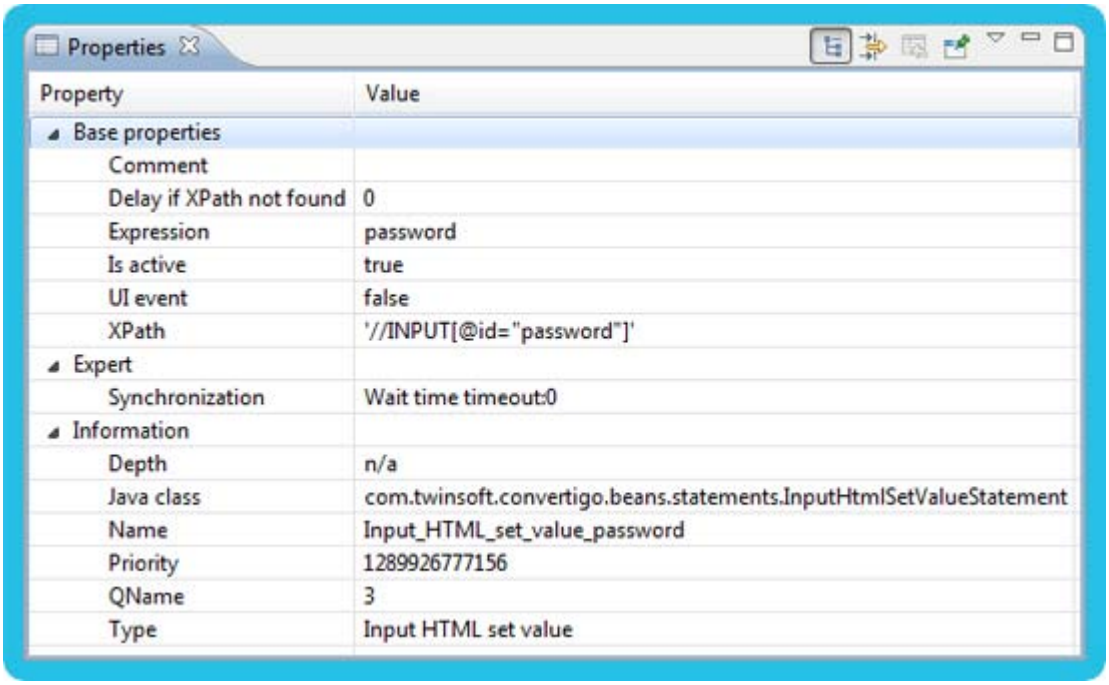


Figure 2 - 331: Input HTML set value statement - Configuration example

Expression property of both statements contains the JavaScript expression resulting in the inputted value when evaluated, it is directly the transaction variable corresponding to each field: `username` or `password`. The **XPath** property of both statements contains an Xpath identifying precisely, thanks to an `id` attribute, each `INPUT` element from the web page DOM on which the action has to be performed.

Synchronization property of both statements is set to `No Wait`, which is the default value of this property, or `Wait time` with a `timeout` to `0ms`. These two synchronization values are equivalent and do not wait for anything to happen before continuing. In our case, these values are set because no change is supposed to happen in the web page when filling these fields. This property is edited for both statements in the **Trigger editor**:

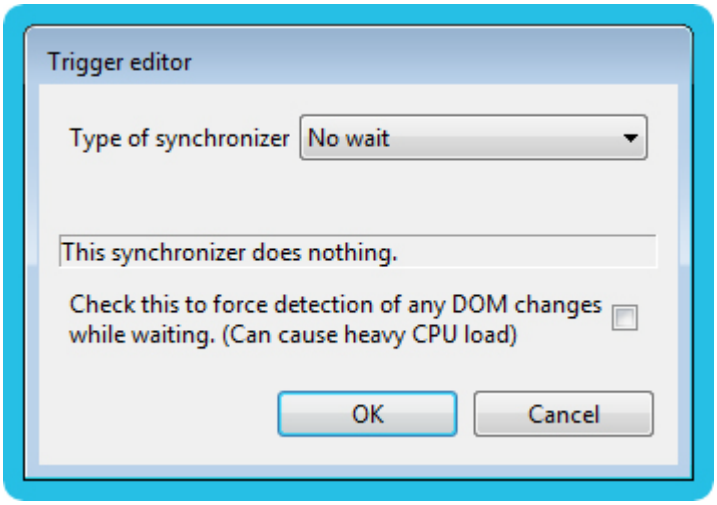


Figure 2 - 332: Input HTML set value statement - Synchronization property edition for No Wait

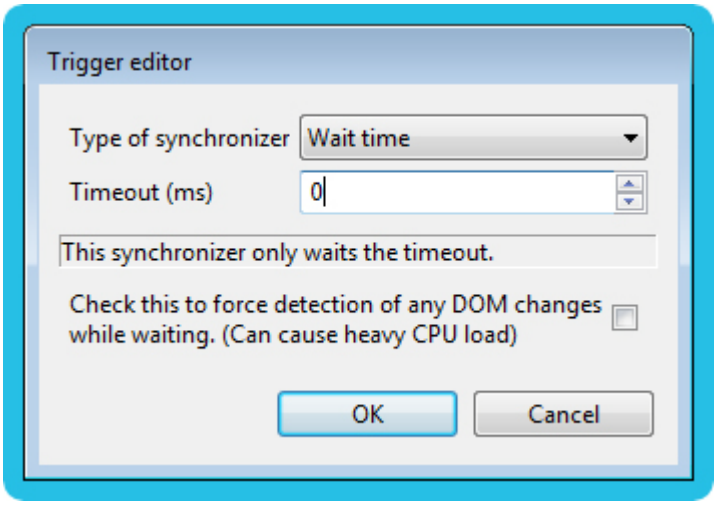


Figure 2 - 333: Input HTML set value statement - Synchronization property edition for Wait time 0ms

The statements are created in the **Functions** folder of the transaction, under the corresponding Screen class handler, next to other statements that implement the transaction behavior described above. It appears as follows in the **Projects** view:

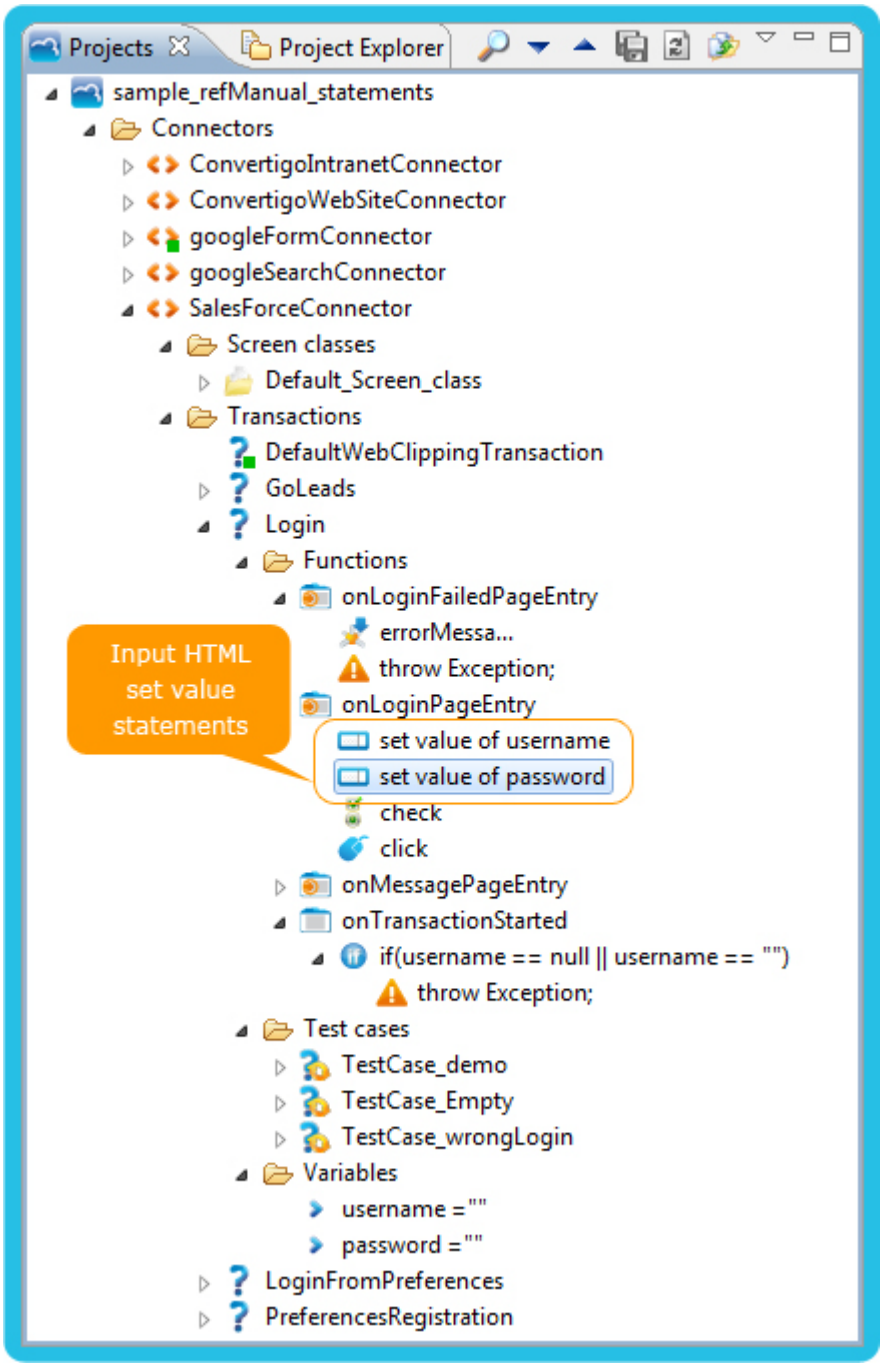


Figure 2 - 334: Input HTML set value statement - Objects in Projects view

When executing the test cases named `TestCase_demo` defined for the `Login` transaction, the transaction authenticates correctly the user in Sales Force website and checks the check box allowing to remember the user name.

INPUT HTML SET SELECTED 

OBJECT DESCRIPTION

Selects an `option` in an HTML combo box.

This statement enables Convertigo to select an `option` in a `select` HTML element.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Expression	JS expression	standard	Defines the expression evaluated to give the text identifying the `option` to select. This property is a JavaScript expression that is evaluated during the transaction execution and gives the data that identifies the `option` to be selected in the combo box. The `option` identification also depends on the **Selection mode** property value. Note: It is possible to select several `option` in a same `select` by setting a JavaScript array instead of a simple string variable in **Expression** property.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Selection mode	String	standard	Defines which mode is used to identify the `option` to select. There are three modes that can be used to identify the `option` to select in the combo box: • `by index`: the `option` is selected by its index order in the options list. The **Expression** property evaluation matches the index of the sought `option`. • `by value`: the `option` is selected by its `value` attribute. The **Expression** property evaluation matches the `value` attribute of of the sought `option`. • `by content`: the `option` is selected by its text content. The **Expression** property evaluation matches the text content of of the sought `option`.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

UI event	boolean	standard	Defines whether UI events must be used to change the item value. Setting this property to `false` is saving CPU time, but sometimes the target website does not have the full behavior it has with a real user. In this case, setting this property to `true` will help having a behavior closer to the original site.
XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the US directory website. Every page of this site contains a search form on its top:

Figure 2 - 335: Input HTML set selected statement - US directory website pages with search form

A `searchBusiness` transaction, which defines three variables named `business`, `cityZIPcode`, and `state`, sets those three values into search form input fields and launch a search by clicking on the `Search` button.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

The `searchBusiness` transaction is able to fill the search form on the three pages of the US directory website that are identified by screen classes in the project. The actions to perform on every screen are defined thanks to a *Function* statement named `setSearchInputs` as follows:

- set Business input,

- set City / ZIP code input,
- select State in combobox,
- click on the `Search` button.

In order to select the state in State combo box, a *Input HTML set selected* statement is created with the following parameters:

```
Input HTML set selected [
  xpath='//SELECT[@class="form_select_state" and @id="qs"]'
  expression: state
  selection mode=by value
  uiEvent=false
  synchronization=[No Wait]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

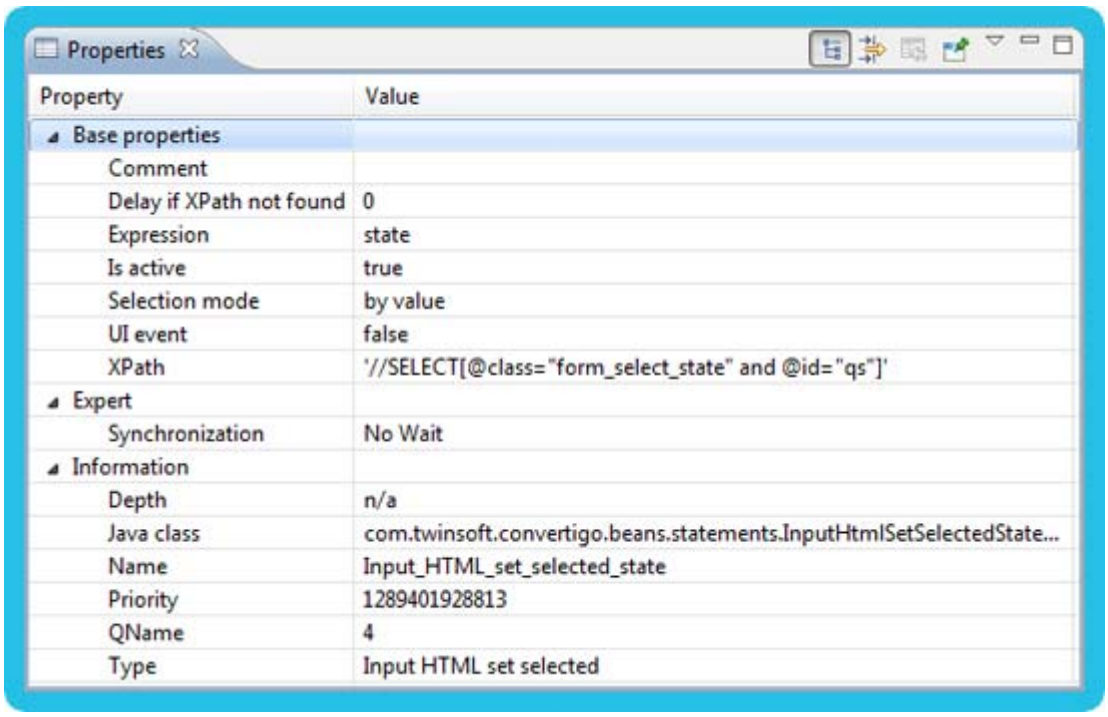


Figure 2 - 336: Input HTML set selected statement - Configuration example

The **XPath** property contains an Xpath identifying precisely, thanks to an `id` attribute, the `SELECT` element from the web page DOM on which the action has to be performed. The **Selection mode** property defines that the entry to select is chosen by `value`, meaning that the `value` attribute of the `OPTION` element to select equals the evaluated value of the **Expression** property, which is simply set to the `state` transaction variable value.

Synchronization property is set to `No Wait` because no change is supposed to happen in the web page when selecting the value in the combobox, so the transaction does not need to wait before continuing. This property is edited in the **Trigger editor**:

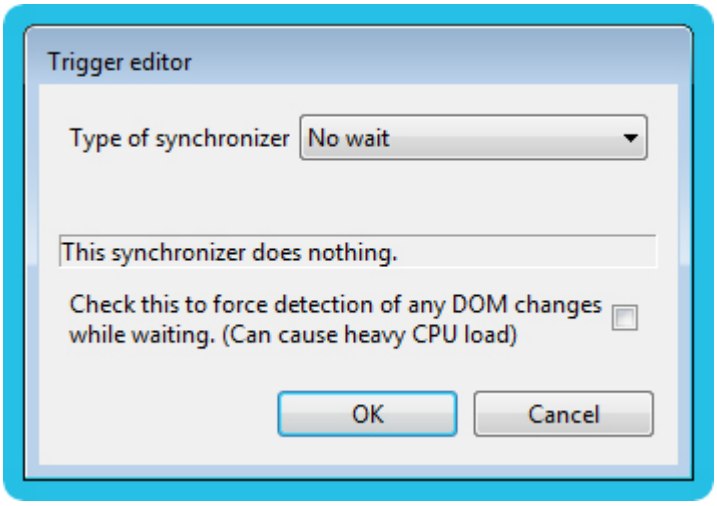


Figure 2 - 337: Input HTML set selected statement - Synchronization property edition

The statement is created in the **Functions** folder of the transaction, under the `setSearchInputs` *Function* statement, next to other statements used to implement the transaction behavior described above, and appears as follows in the **Projects** view:

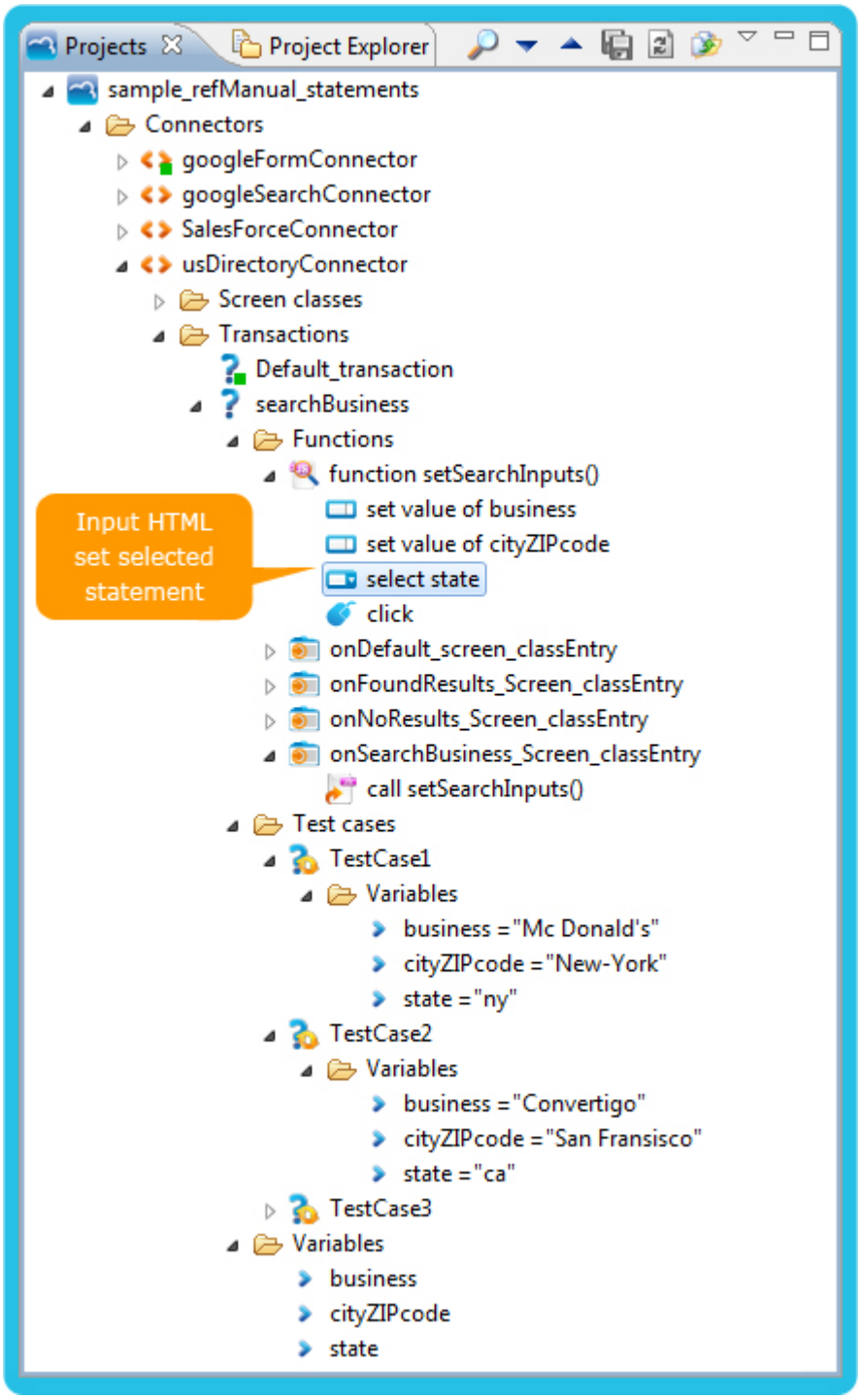


Figure 2 - 338: Input HTML set selected statement - Object in Projects view

When executing test cases defined for the transaction, starting from every page from US directory website, the transaction does the same actions (described above) and searches the business.

INPUT HTML SET CHECKED 

OBJECT DESCRIPTION

Checks / unchecks an HTML check box.

This statement enables Convertigo to check / uncheck a `checkbox` type `input` HTML element.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Checked	boolean	standard	Defines whether to check or uncheck the input. If set to `true`, the item has to be checked. If set to `false`, the item has to be unchecked.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

UI event	boolean	standard	Defines whether UI events must be used to change the item value. Setting this property to `false` is saving CPU time, but sometimes the target website does not have the full behavior it has with a real user. In this case, setting this property to `true` will help having a behavior closer to the original site.
XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the SalesForce website, famous CRM SaaS application. This website needs an authentication thanks to a user / password:

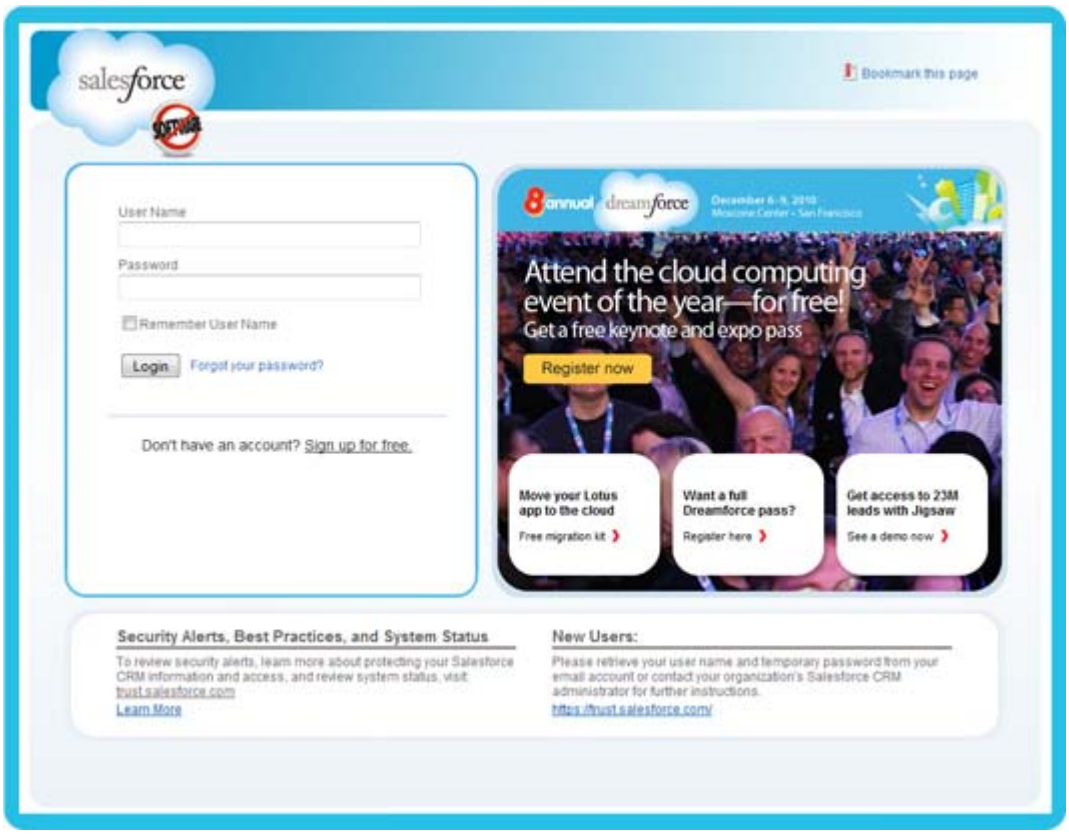


Figure 2 - 339: Input HTML set checked statement - SalesForce website authentication page

A `Login` transaction, which defines two variables named `username` and `password`, is written to authenticate the user on SalesForce website.

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

The `Login` transaction:

- fills the username and password fields using the `user` / `password` variables,
- submits the authentication form to access to SalesForce welcome page.

When the authentication is not correct or when the username / password is not provided, the transaction raises a Convertigo Engine Exception.

A `Remember User Name` check box is present in the login form. we could add additional behavior to the transaction so it checks this check box while authenticating the user.

To do so, an *Input HTML set checked* statement is created in the transaction, with the following parameters:

```
Input HTML set checked [
```

```
  xpath='//INPUT[@id="rememberUn"]'
  checked=true
  uiEvent=false
  synchronization=[Wait time, timeout= 0 ms]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

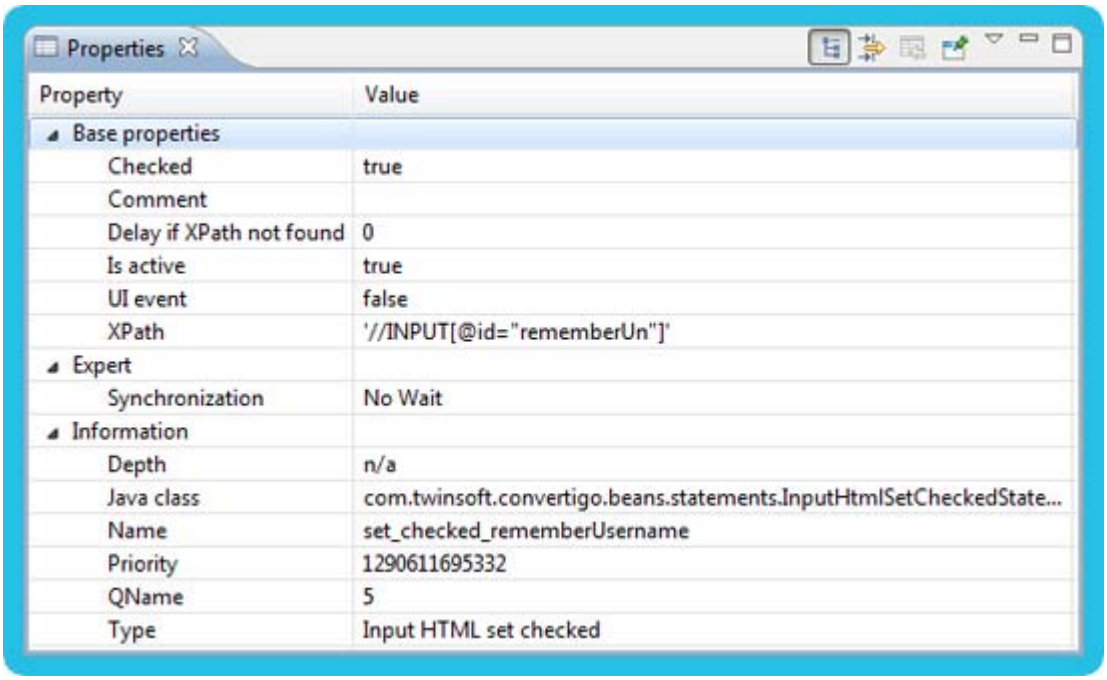


Figure 2 - 340: Input HTML set checked statement - Configuration example

Checked property is set to `true` so as the check box has to be checked by the transaction. The **XPath** property contains an Xpath identifying precisely, thanks to an `id` attribute, the `INPUT` element from the web page DOM on which the action has to be performed.

Synchronization property is set to `No Wait` because no change is supposed to happen in the web page when checking this box, so the transaction does not need to wait before continuing. This property is edited in the **Trigger editor**:

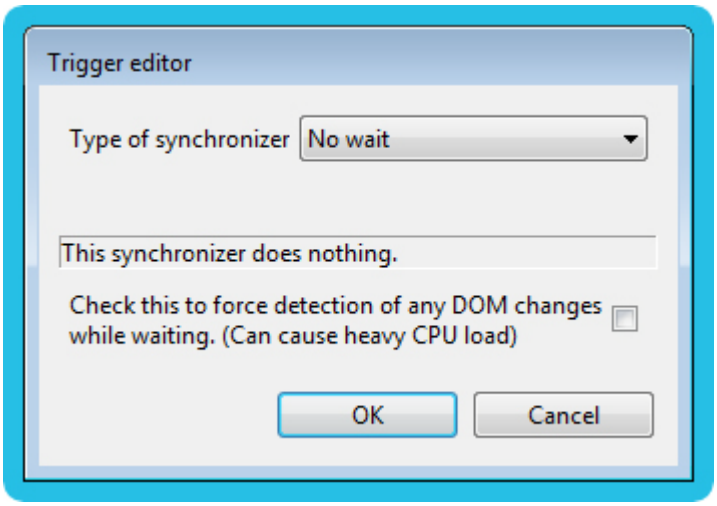


Figure 2 - 341: Input HTML set checked statement - Synchronization property edition

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler, next to other statements that implement the transaction behavior described above. It appears as follows in the **Projects** view:

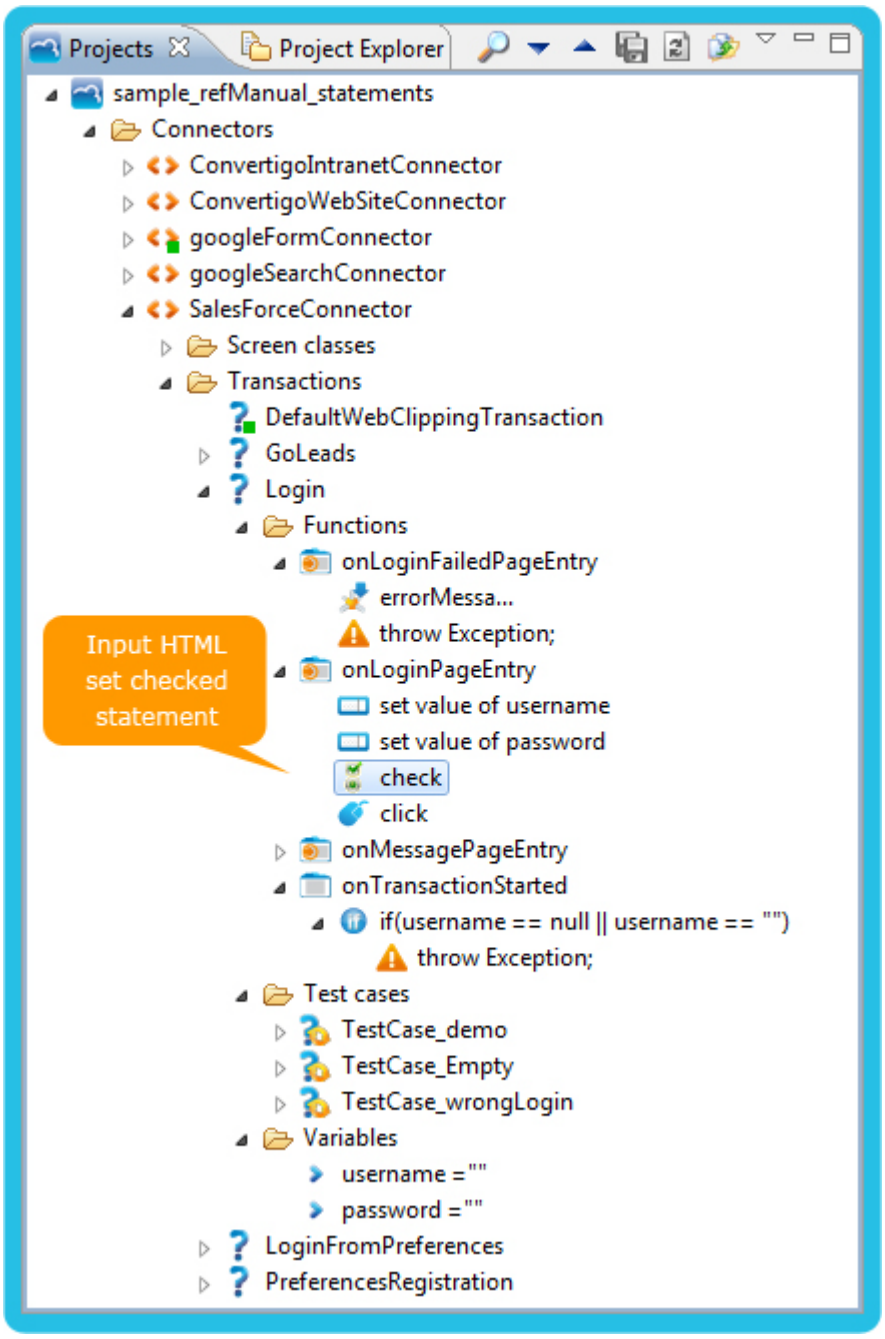


Figure 2 - 342: Input HTML set checked statement - Object in Projects view

When executing the test cases named `TestCase_demo` defined for the `Login` transaction, the transaction authenticates correctly the user in Sales Force website and checks the check box allowing to remember the user name.

MOUSE ACTION 

OBJECT DESCRIPTION

Creates a mouse action on an element of the web page.

This statement allows to trigger mouse events, provided that attributes of the accessed DOM element manage such events.

For example, for a click on a target HTML object, this object can be a link (A HTML element) or a button (`INPUT type="button"` element) or any HTML object the user can click on in the web page.

Supported mouse events, defined in the **Action** property, are the following:

- `click`: simulates a mouse `click` event,
- `mousedown`: simulates a `mousedown` event (first part of a mouse click event, when mouse button is clicked),
- `mouseup`: simulates a `mouseup` event (second part of a mouse click event, when mouse button is released),
- `mouseover`: simulates a `mouseover` event (mouse is moved onto an element),
- `mouseout`: simulates a `mouseout` event (mouse is moved out of an element),
- `mousemove`: simulates a `mousemove` event (mouse is moved anywhere),
- `mousedrag`: simulates a `mousedrag` event (mouse drags an object),
- `dblclick`: simulates a `dblclick` event (mouse double-clicks an object),
- `dragdrop`: simulates a `dragdrop` event (mouse drags and drops an object).

The XPath defined in **XPath** property is applied on the web page to retrieve the element on which the event has to be performed.

Note: For more complex actions, prefer to use the *Mouse action advanced* statement.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	standard	Defines the JavaScript action corresponding to the event to perform. Depending of the statement, this property can take several values. These values are indicated in the object's description.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the `searchGoogleWithLimit` transaction set in the context of the "*Starting With Convertigo Web Integrator*" *tutorial*.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Documentation samples*** *>* ***Web integration*** *in the* ***New Project*** *wizard.*

This transaction, which defines two variables named `keyword` and `maxPages`, searches for this keyword in Google search engine and accumulates the results of `maxPages` pages into an XML structure thanks to a *Table* extraction rule.

When accessing the Google.com search page, the transaction sets the `keyword` variable value into the search field thanks to an *Input HTML set value* statement and clicks on the "Google search" button to run the search. In order to perform this action, the transaction implements a *Mouse action* statement called `clickSearchButton`.

The *Mouse action* statement is created in the `searchGoogleWithLimit` transaction with the following parameters:

```
Mouse action [
  xpath='//INPUT[@name="btnG" and @type="submit"]'
  action=click
  synchronization=[document completed: 1, timeout= 60000 ms]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

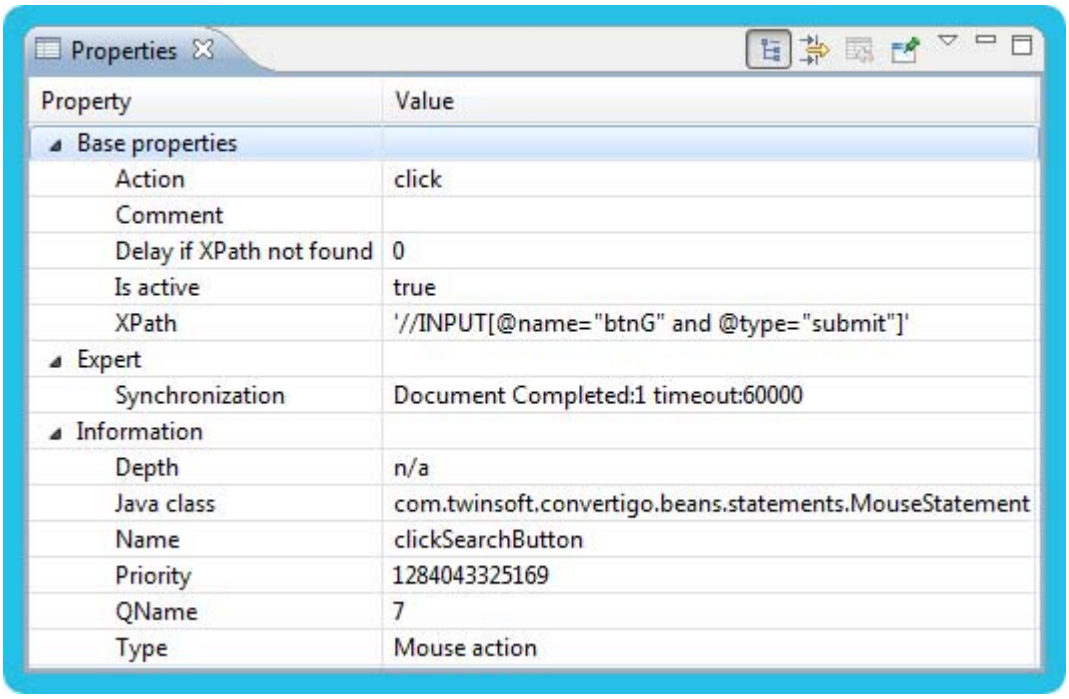


Figure 2 - 343: Mouse action statement - Configuration example

The **Xpath** property is matching the "Google Search" button as the action has to be performed on this element from the page (`INPUT` element button of `submit` type named `btnG`). The **Action** property is set to `click` which simulates a simple click on the element.

The **Synchronization** property indicates that, after triggering the action, the statement waits

for *one* document to be fully loaded before continuing the transaction execution. This property is edited in the **Trigger editor**:

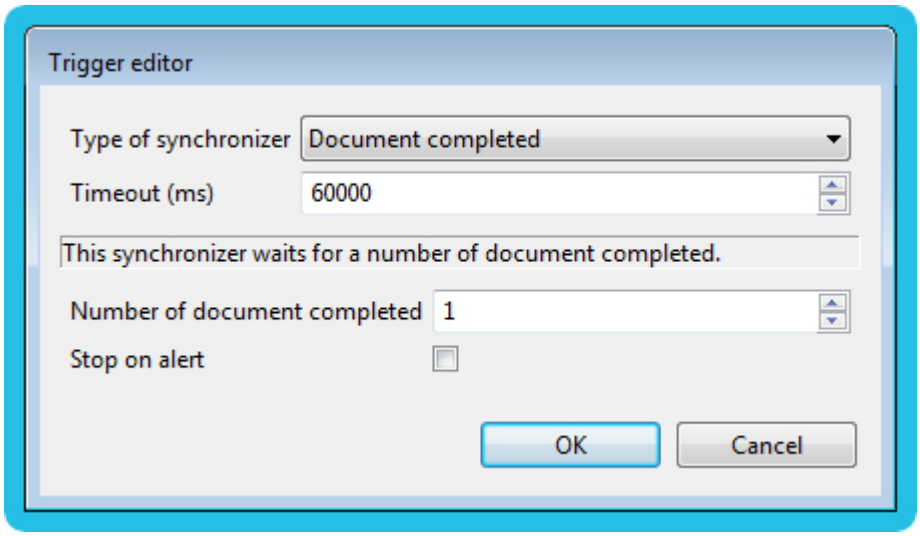


Figure 2 - 344: Mouse action statement - Synchronization property edition

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and appears as follows in the **Projects** view:

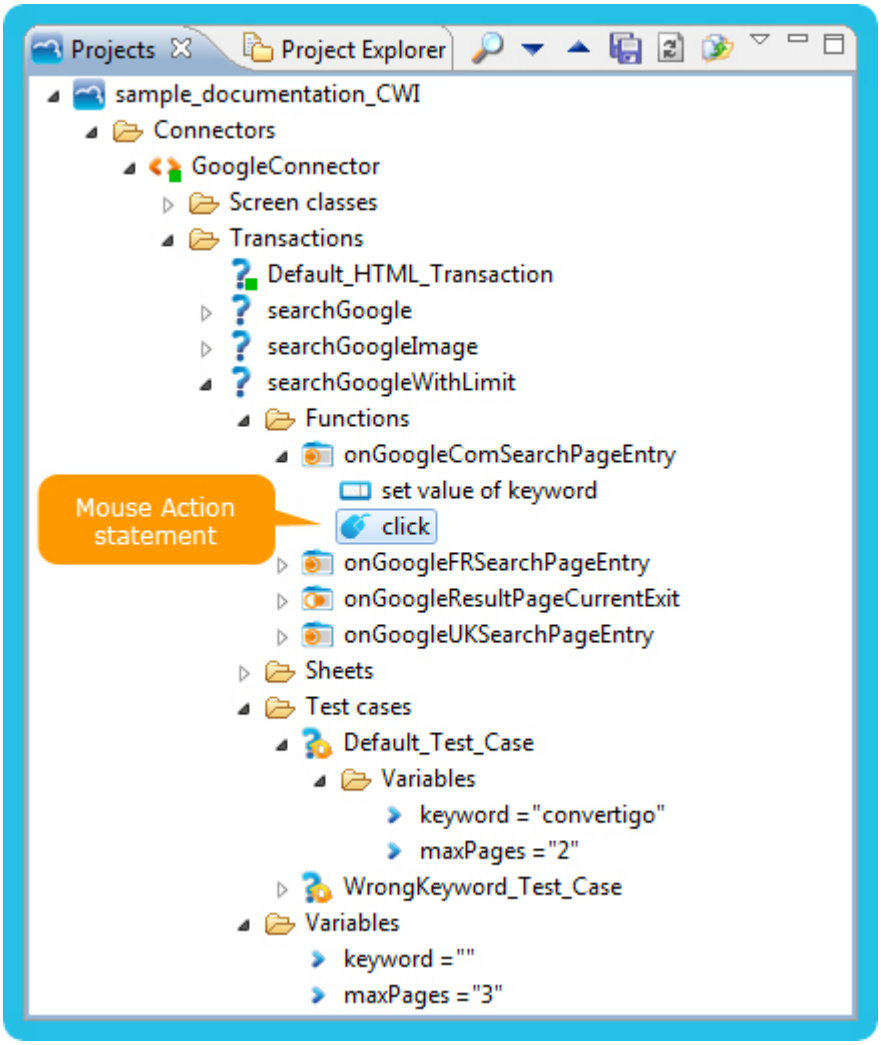


Figure 2 - 345: Mouse action statement - Object in Projects view

When executing the test case defined for the transaction, it fills the keyword, clicks the button and waits for the page to be reloaded (probably with the results page).

MOUSE ACTION ADVANCED 

OBJECT DESCRIPTION

Creates a complex mouse action on an element or a position of the web page.

This statement allows to trigger complex mouse events, provided that attributes of the accessed DOM element manage such events.

Supported mouse events, defined in the **Action** property, are the following:

- `click`: simulates a mouse `click` event,
- `mousedown`: simulates a `mousedown` event (first part of a mouse click event, when mouse button is clicked),
- `mouseup`: simulates a `mouseup` event (second part of a mouse click event, when mouse button is released),
- `mouseover`: simulates a `mouseover` event (mouse is moved onto an element),
- `mouseout`: simulates a `mouseout` event (mouse is moved out of an element),
- `mousemove`: simulates a `mousemove` event (mouse is moved anywhere),
- `mousedrag`: simulates a `mousedrag` event (mouse drags an object),
- `dblclick`: simulates a `dblclick` event (mouse double-clicks an object),
- `dragdrop`: simulates a `dragdrop` event (mouse drags and drops an object).

The XPath defined in **XPath** property is applied on the web page to retrieve the element on which the event has to be performed. If the **XPath** property is empty, the mouse position can be used to find an element on which perform the event.

The mouse position can be defined using the **Client X** and the **Client Y** properties or the **Screen X** and the **Screen Y** properties.

Note: For simple actions, prefer to use the *Mouse action* statement.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	standard	Defines the JavaScript action corresponding to the event to perform. Depending of the statement, this property can take several values. These values are indicated in the object's description.

Property	Type	Category	Description
Button	JS expression	standard	Defines which mouse button is used during the mouse event. Some mouse events are triggered by the depression or release of a mouse button. This property is a JavaScript expression that is evaluated during the transaction execution and indicates which mouse button is used during this state change. This property can take the following values: • `zero`: left button, • `one`: middle button, if applicable, • `two`: right button. For mouse devices configured for left-handed use, in which button actions are reversed, values are reversed too (from right to left).
Client X	JS expression	standard	Defines the client (web browser) relative X coordinate (in pixels). This property is a JavaScript expression that is evaluated during the transaction execution and gives the horizontal coordinate, relatively to the top left corner of the DOM layout, at which the event should occur.
Client Y	JS expression	standard	Defines the client (web browser) relative Y coordinate (in pixels). This property is a JavaScript expression that is evaluated during the transaction execution and gives the vertical coordinate, relatively to the top left corner of the DOM layout, at which the event should occur.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.
Press alt key	JS expression	standard	Defines whether the "`alt`" key is to be depressed when firing the event. This property is a JavaScript expression that is evaluated during the transaction execution as a boolean defining if the `alt` key should be depressed when firing the event. On some platforms, the `alt` key may map to an alternative key name.

Property	Type	Category	Description
Press ctrl key	JS expression	standard	Defines whether the "`ctrl`" key is to be depressed when firing the event. This property is a JavaScript expression that is evaluated during the transaction execution as a boolean defining if the `ctrl` key should be depressed when firing the event. On some platforms, the `ctrl` key may map to an alternative key name.
Press meta key	JS expression	standard	Defines whether the "`meta`" key is to be depressed when firing the event. This property is a JavaScript expression that is evaluated during the transaction execution as a boolean defining if the `meta` key should be depressed when firing the event. On some platforms, the `meta` key may map to an alternative key name.
Press shift key	JS expression	standard	Defines whether the "`shift`" key is to be depressed when firing the event. This property is a JavaScript expression that is evaluated during the transaction execution as a boolean defining if the `shift` key should be depressed when firing the event. On some platforms, the `shift` key may map to an alternative key name.
Screen X	JS expression	standard	Defines the absolute screen `X` coordinate (in pixels). This property is a JavaScript expression that is evaluated during the transaction execution and gives the horizontal coordinate, relatively to the top left corner of the screen, at which the event should occur.
Screen Y	JS expression	standard	Defines the absolute screen `Y` coordinate (in pixels). This property is a JavaScript expression that is evaluated during the transaction execution and gives the vertical coordinate, relatively to the top left corner of the screen, at which the event should occur.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

CREATE EVENT 

OBJECT DESCRIPTION

Creates an event on an element of the web page.

Event, as defined by W3C specifications, is defined in the **Action** property and can be one of the following values:

- `click`,
- `mousedown`,
- `mouseup`,
- `keydown`,
- `keyup`,
- `keypress`,
- `submit`,
- `mouseover`,
- `mouseout`,
- `mousemove`,
- `mousedrag`,
- `dblclick`,
- `dragdrop`,
- `focus`,
- `blur`,
- `select`,
- `change`,
- `reset`,
- `scroll`,
- `load`,
- `unload`,
- `abort`,
- `error`,
- `locate`,
- `move`,
- `resize`,
- `forward`,
- `help`,
- `back`,

- `text`.

The element on which the event has to be performed in the web page is retrieved by the execution of the XPath defined in **XPath** property.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Action	String	standard	Defines the JavaScript action corresponding to the event to perform. Depending of the statement, this property can take several values. These values are indicated in the object's description.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Let's consider the `searchGoogleWithLimit` transaction set in the context of the "*Starting With Convertigo Web Integrator*" *tutorial*. This transaction, which defines two variables named `keyword` and `maxPages`, searches for the keyword in Google search engine and accumulates the results of `maxPages` pages into an XML structure thanks to a *Table* extraction rule.

When accessing the Google.com search page, the transaction sets the `keyword` variable value into the search field and clicks on the "Google search" button to run the search. This click is perfomed by a *Mouse action* statement but can also be performed by a *Create event* statement (for more information about *Mouse action* statement, see *Mouse action* statement documentation and examples).

For this example, we duplicated the `GoogleConnector` *HTML connector* of this project into the `sample_refManual_statements` project, renaming it to `googleSearchConnector`.

You can find the complete example project in the Studio. To open this project, refer to the procedure described in the "Starting with Convertigo Web Integrator" tutorial or the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

In the `searchGoogleWithLimit` transaction, we can disable the previously existing *Mouse action* statement and define a *Create event* statement performing the click on the button. This statement is created with the following parameters:

```
Create event [
   action=click
   xpath='//INPUT[@name="btnG" and @type="submit"]'
   synchronisation=[document completed: 1, timeout= 60000 ms]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

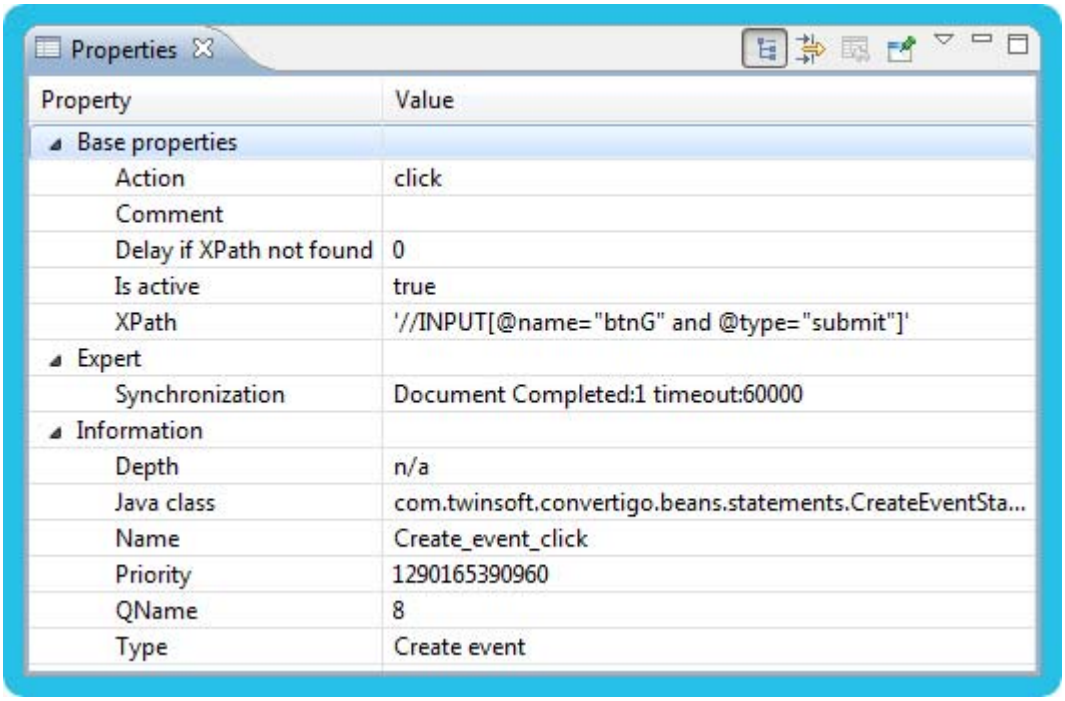


Figure 2 - 346: Create event statement - Configuration example

The **Action** property is set to `click` which simulates the same action as the *Mouse action* statement. The **Xpath** property is the same for both statements as the action has to be performed on the same element from the page.

The **Synchronization** property indicates that, after triggering the event, the statement waits for *one* document to be fully loaded before continuing the transaction execution. This property is edited in the **Trigger editor**:

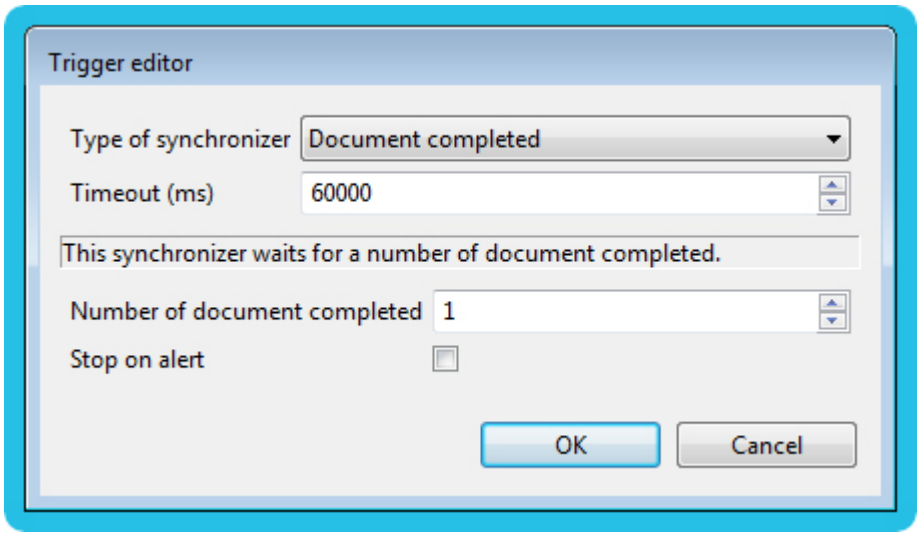


Figure 2 - 347: Create event statement - Synchronization property edition

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and appears as follows in the **Projects** view:

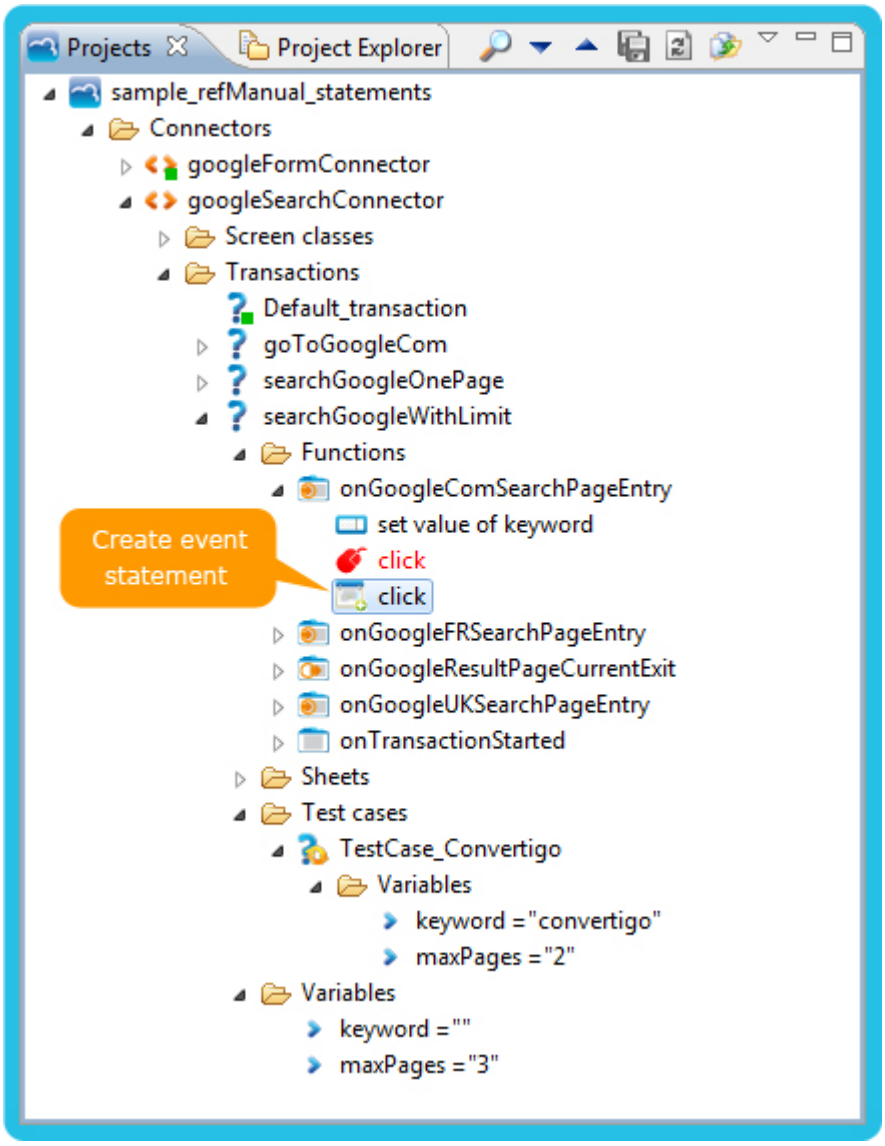


Figure 2 - 348: Create event statement - Object in Projects view

When executing the test case defined for the transaction, it is the executed the same way as it was before the replacement of the *Mouse action* statement.

INPUT HTML SET FILE 

OBJECT DESCRIPTION

Selects a file in an HTML input field.

This statement enables Convertigo to set a value in a `file` type `input` HTML element.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
File path	JS expression	standard	JavaScript expression defining the file path, including the file name, of the file to upload. This file must be a local file. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

BROWSER CONTROL STATEMENTS

CREDENTIALS 

OBJECT DESCRIPTION

Defines authentication credentials.

This statement allows setting credentials to access any application supporting basic authentication (`WWW-Authenticate`). This statement can be used in a `login` transaction where the caller provides credentials to access the target application.

Credentials set for this statement override the connector's credentials property. The statement must be executed before any other statement possibly needing the same credentials. This statement is usually set as part of a *Start transaction handler* (`onTransactionStarted` event handler).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Force basic	boolean	expert	Force sending `Basic` header with those credentials for each *HTML connector* request. If set to `true`, all requests sent by the *HTML connector* for which these credentials were positioned send credentials in the `Basic` header.
Is active	boolean	standard	Defines whether the statement is active.
Password	JS expression	standard	Defines the user password. `WWW-Authenticate` password. May be any JavaScript expression using transaction variables.
User	JS expression	standard	Defines the user name. `WWW-Authenticate` user. May be any JavaScript expression using transaction variables.

EXAMPLES

Let's consider the Convertigo Intranet website, used by Convertigo employees to access internal company information. This website needs a basic authentification:

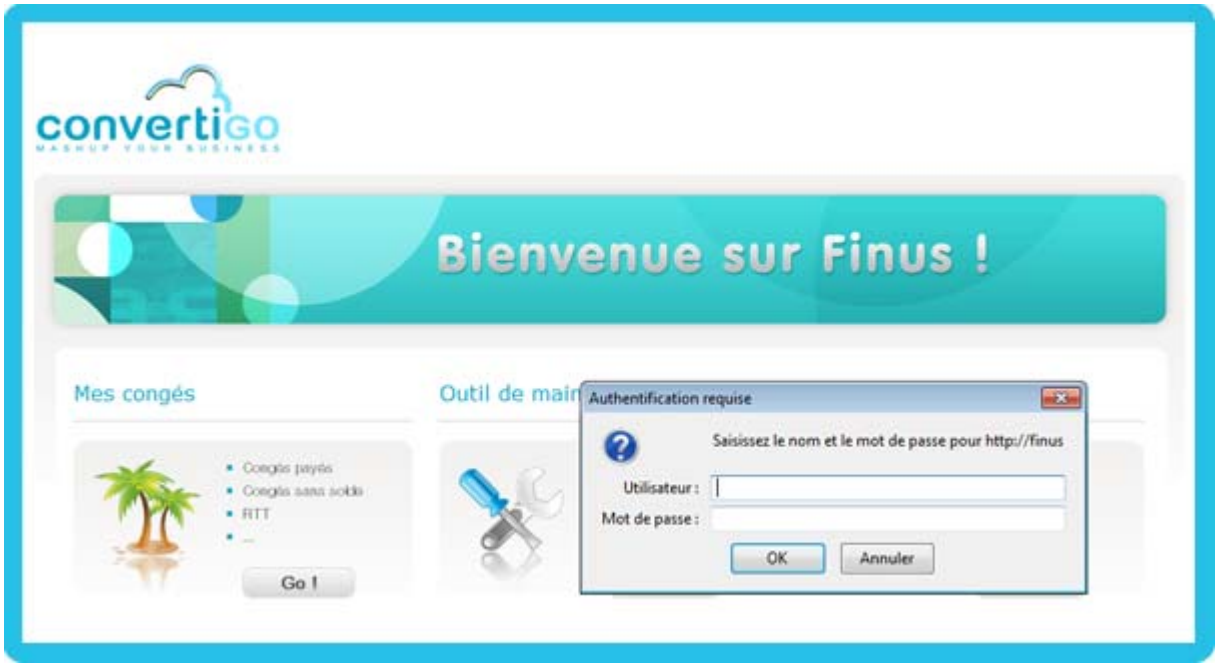


Figure 2 - 349: Credentials statement - Intranet website with basic authentication

A `GoToHolidaysApp` transaction, which defines two variables named `username` and `password`, is created to access the holidays management application for an employee. This transaction:

- connects to Convertigo Intranet website,
- access the holidays application by clicking on the corresponding "Go" button. When doing so, the basic authentication window pops up, as shown on previous figure.

For Convertigo to handle this authentication, a *Credentials* statement is created in the transaction start handler with the following parameters:

```
Credentials [
   user=username
   password=password
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

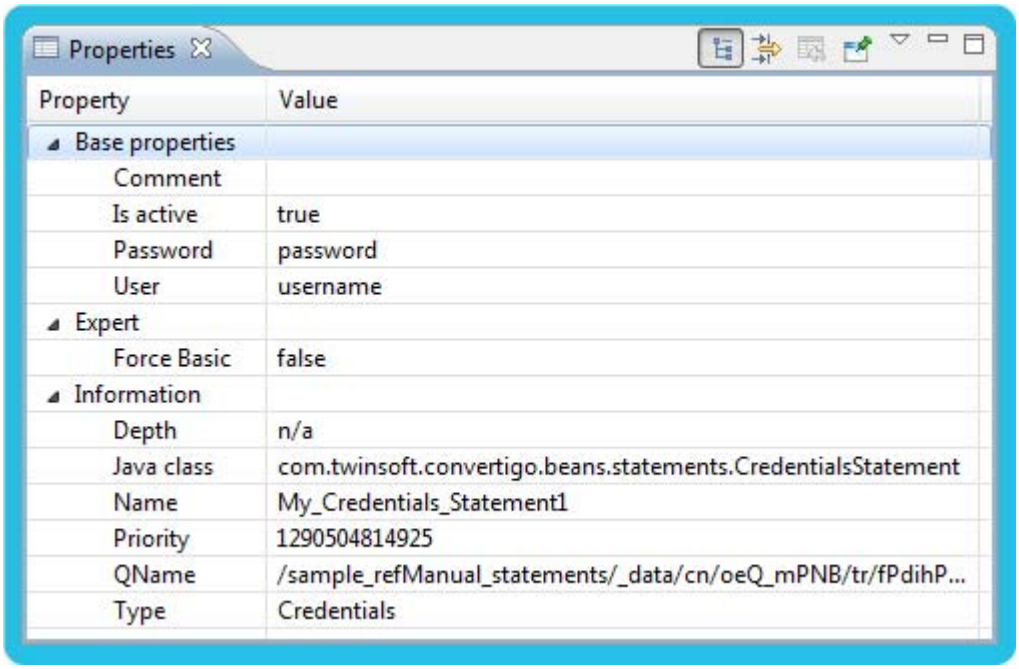


Figure 2 - 350: Credentials statement - Configuration example

This statement is created in the **Functions** folder of the transaction, under the transaction start handler and appears as follows in the **Projects** view:

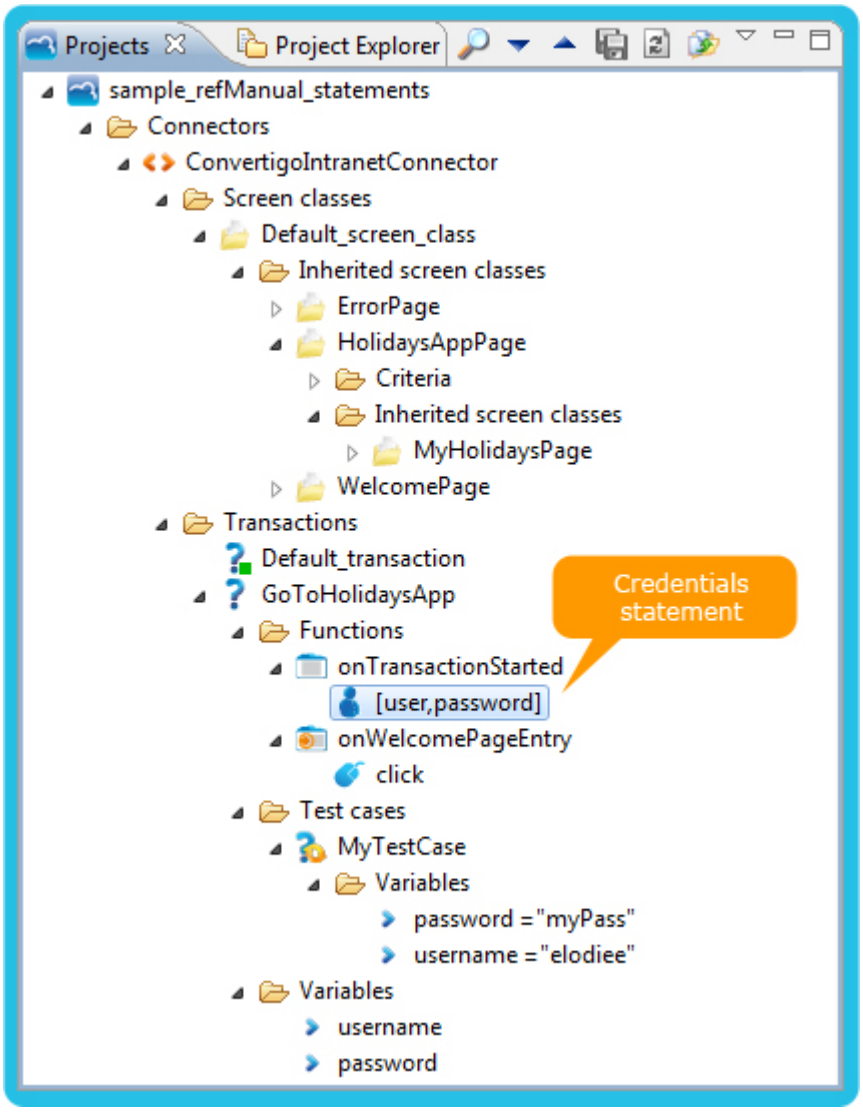


Figure 2 - 351: Credentials statement - Object in Projects view

When executing the test case defined for the `GoToHolidaysApp` transaction, user authentication values are automatically send by Convertigo to basic authentication. Then the transaction accesses the user's holidays management application.

BROWSER PROPERTY CHANGE 

OBJECT DESCRIPTION

Allows changing some properties of the embedded browser.

This statement allows a transaction to change some of the properties of the Mozilla-based Convertigo HTML decoder. For each property, you can specify if the property has to be modified and the new value of this property, or if the property keeps its unchanged value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attachment enable value	AttachmentMode	standard	Forces the **Attachment retrieval** property on (enabled) or `off` (disabled), or leaves it unchanged. By default, attachment files are not downloaded when transfer dialogs open, in Studio or Server. Using this statement, **Attachment retrieval** property value can be changed and has the following effects: • If set to `force on`, attachment files are automatically downloaded when transfer dialogs open and are recoverable by the *Get attachment* extraction rule. • If set to `force off`, attachment files are never downloaded when transfer dialogs open. • If set to `no change`, it keeps the default value (disabled) or the previously set value, if another *Browser property change* statement changed the value previously in the same context.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Convertigo mode	ConvertigoMode	expert	Applies changes on Studio, Engine, or both. Defines in which mode the browser setting must be active: • `Studio mode`: settings only valid in Convertigo Studio, • `Engine mode`: settings only valid in Convertigo Server, • `Both modes`: settings valid in Convertigo Server as well as in Studio (default value).

Property	Type	Category	Description
Image rendering enable value	ImageMode	standard	Forces the browser **Image rendering** property on (enabled) or `off` (disabled), or leaves it unchanged. By default, the **Image rendering** property value depends on the environment: • in Studio, **Image rendering** property is enabled by default, • in Server, the default value is based on a property configured in Administration Console *Configuration* page, in advanced properties of *HTML parser configuration* tab. Using this statement, **Image rendering** property value can be changed and has the following effects: • If set to `force off`, the HTML decoder does not render images, which speeds up the process. It is used mostly for production server mode, so another way to do the same behavior is to always set this property to `no change` with a default value configured to `false` in the Administration Console. • If set to `force on`, the HTML decoder always renders images, which slows down a little the process. • If set to `no change`, it keeps the default value or the previously set value, if another *Browser property change* statement changed the value previously in the same context.
Is active	boolean	standard	Defines whether the statement is active.
JavaScript enable value	JavascriptMode	standard	Forces the browser **JavaScript** property on (enabled) or `off` (disabled), or leaves it unchanged. By default, the JavaScript code present in target HTML pages is executed, in Studio or Server, and the resulting DOM reflects the JavaScript execution. Using this statement, **JavaScript** property value can be changed and has the following effects: • If set to `force on`, embedded JavaScript code in target HTML pages is executed. • If set to `force off`, embedded JavaScript code in target pages is not executed. • If set to `no change`, it keeps the default value (enabled) or the previously set value, if another *Browser property change* statement changed the value previously in the same context. In most cases, **JavaScript** property should be enabled but for specific Web Clipping projects, we recommend disabling JavaScript to prevent a double JavaScript execution (one in the HTML decoder and one in the user's browser).

Property	Type	Category	Description
Plugin enable value	PluginMode	standard	Forces the browser **Plugin feature** property `on` (enabled) or `off` (disabled), or leaves it unchanged. By default, the **Plugin feature** property value depends on the environment: • in Studio, **Plugin feature** property is enabled by default, • in Server, the default value is based on a property configured in Administration Console *Configuration* page, in advanced properties of *HTML parser configuration* tab. Using this statement, **Plugin feature** property value can be changed and has the following effects: • If set to `force off`, the plugins such as Flash player are disabled, which speeds up the process. It is used mostly for production server mode, so another way to do the same behavior is to always set this property to `no change` with a default value configured to `false` in the Administration Console. • If set to `force on`, the plugins such as Flash player are enabled, which slows down a little the process. • If set to `no change`, it keeps the default value or the previously set value, if another *Browser property change* statement changed the value previously in the same context.
Remove cookies	boolean	standard	Removes all cookies from the browser. This property is not exactly a browser setting. When this property is set to `true`, all cookies are removed from the Convertigo context and HTML decoder.
Window open enable value	WindowOpenMode	standard	Forces the browser **Window open** property on (enabled with option to open "in new window" or "in same window"), off (disabled) or leaves it unchanged. JavaScript `window.open` function is disabled by default to prevent pop-up windows to be displayed. Using this statement, **Window open** property value can be changes and has the following effects. If **Window open** property is set to `force on`, pop-up windows are enabled. If enabled, two cases are possible: • set to `Force on same window`: pop-up windows are opened replacing the parent window in the HTML decoder (only one browser window opened), • set to `Force on new window`: pop-up windows are opened as new tabs next to the parent window in the HTML decoder (several tabs opened in parallel in the same browser). In this case, tabs opened can be accessed and manipulated thanks to *Tab management* statement. If **Window open** property is set to `force off`, pop-up windows are disabled. If **Window open** property is set to `no change`, it keeps the default value (disabled) or the previously set value, if another *Browser property change* statement changed the value previously in the same context.

EXAMPLES

Let's consider the `searchGoogleWithLimit` transaction set in the context of the "*Starting With Convertigo Web Integrator*" *Quick Guide*. This transaction, which defines two variables named `keyword` and `maxPages`, searches for the keyword in Google search engine and accumulates the results of `maxPages` pages into an XML structure thanks to a *Table* extraction rule.

In order to accelerate the execution of this transaction, we can define a *Browser property change* statement disabling the images rendering. This statement is created with the following parameters:

```
Browser property change [
   image rendering enable value=force off
   convertigo mode=both modes
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

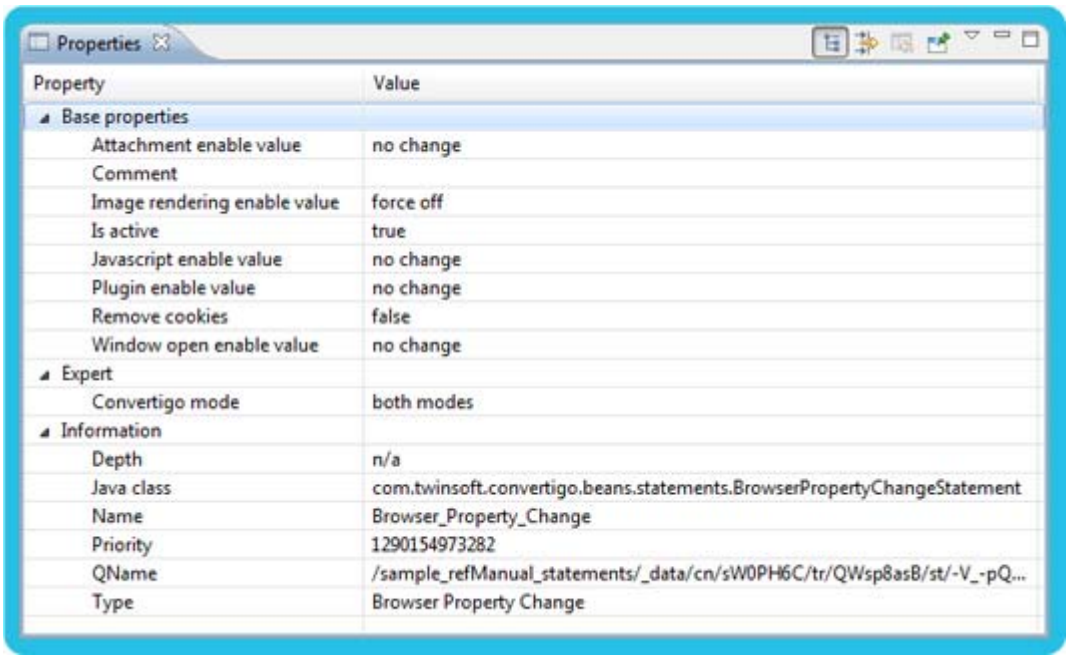


Figure 2 - 352: Browser property change statement - Configuration example

The statement is created in the **Functions** folder of the transaction, under the transaction start handler and appears as follows in the **Projects** view:

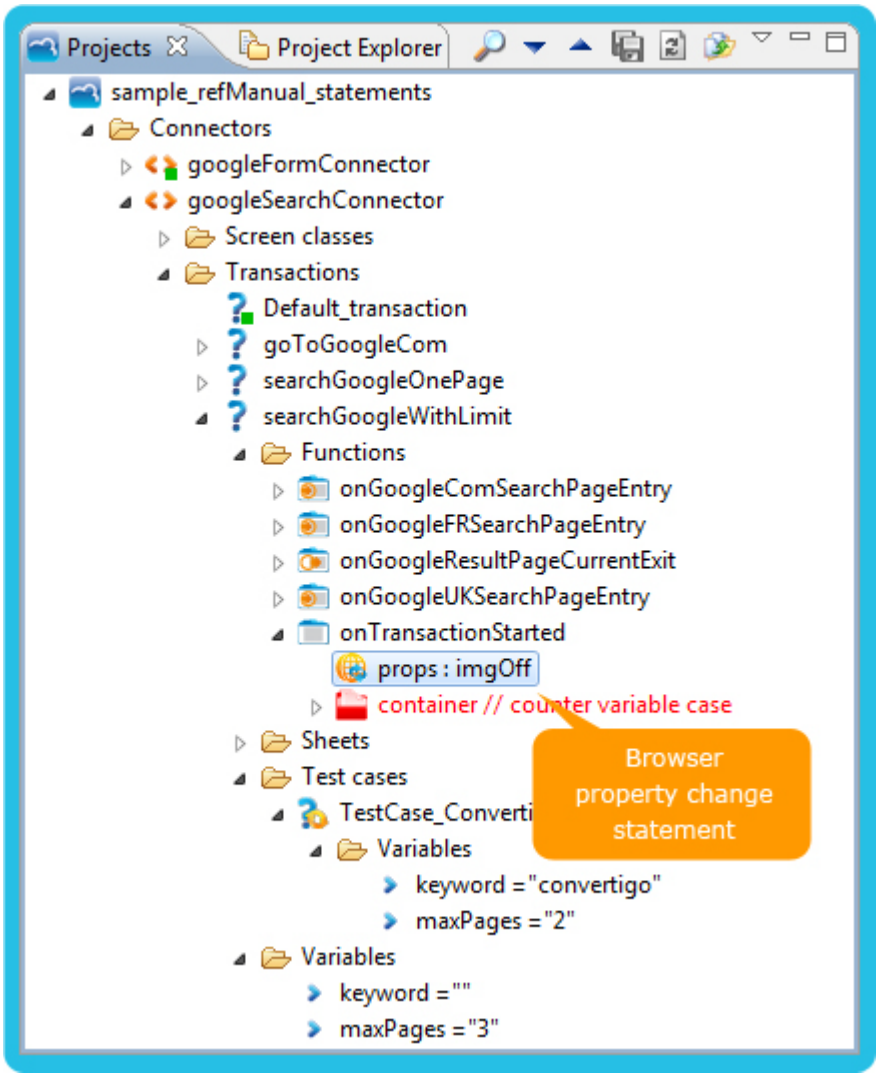


Figure 2 - 353: Browser property change statement - Object in Projects view

When executing the test case defined for the transaction, the connector is reconnected to Google website, with the images rendering disabled. It is visible in the **Connector editor** of the Convertigo studio throughout the execution of the transaction:

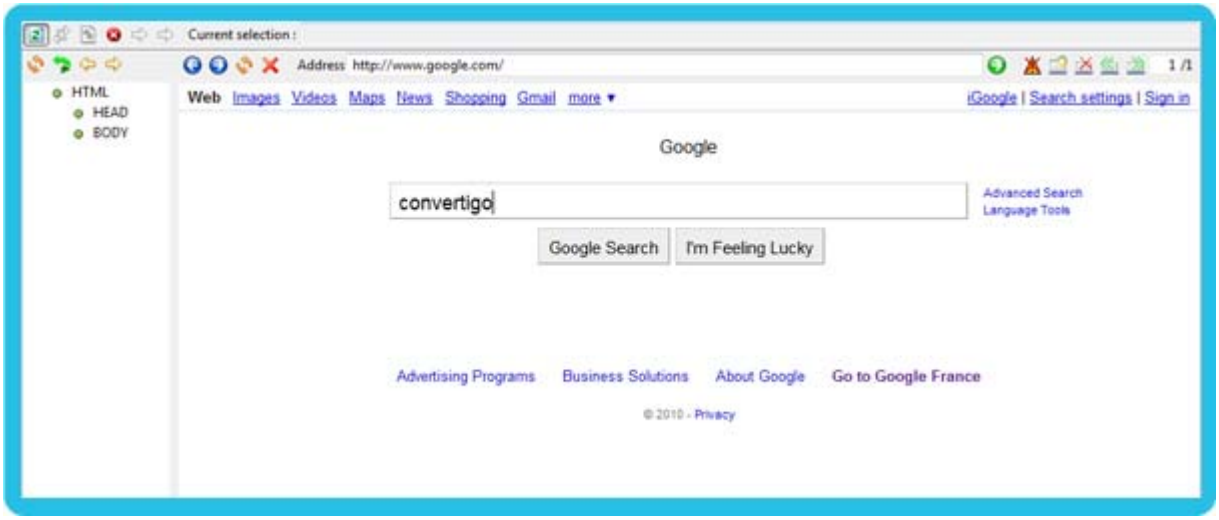


Figure 2 - 354: Browser property change statement - Connecting to Google website with images rendering disabled

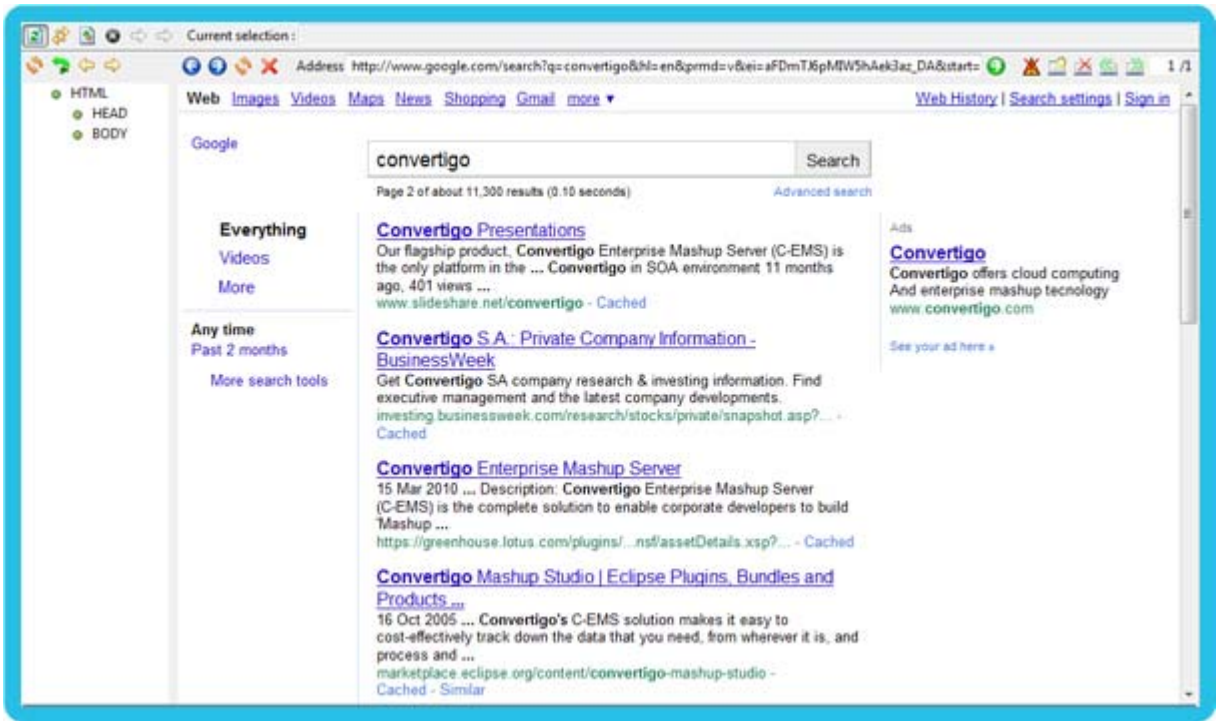


Figure 2 - 355: Browser property change statement - Executing transaction and extracting data with images rendering disabled

NAVIGATION BAR 

OBJECT DESCRIPTION

Simulates a navigation bar action in the browser.

This statement allows Convertigo to navigate in the target web application using the usual navigation bar tools of its internal browser. The action to perform is defined thanks to the **Action** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	standard	Navigation bar action to perform in browser. This property defines the navigation bar tool to use. Following actions are available: • `backward`: goes back to the last visited page, • `forward`: goes forward to the last visited page, • `goTo`: accesses the web page which URL is defined in the **JavaScript URL** property, • `refresh`: reloads the currently displayed page, • `stop`. Note: The `goTo` action requires the **JavaScript URL** property to be set.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
JavaScript URL	JS expression	standard	Defines the URL of the page to access when `goTo` action is set in **Action** property. This property is a JavaScript expression evaluated during the transaction execution.

Property	Type	Category	Description

Synchronizer	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

TAB MANAGEMENT 

OBJECT DESCRIPTION

Manages browser tabs.

Pop-ups opening in the internal browser can be opened as tabs (depending on the value of the browser **Window open** setting). This statement allows to open, close, move, etc. browser tabs.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	standard	Defines the action to be performed on browser tabs. Available actions are the following: • close: closes the current tab, • new: opens a new tab, • getnbtab: retrieves the number of open tabs in a JavaScript variable, • getindex: retrieves the current tab index in a JavaScript variable, • next: set focus on the next tab, • previous: set focus on the previous tab, • setindex: places focus on the tab of specified index.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
JS index	JS expression	standard	Defines the JavaScript expression resulting in the index of the tab to be used. This property has to be filled when the **Action** property is set to an action that is concerned by a tab index value.
Variable name	String	standard	Defines the name of the variable in which data is retrieved. This property has to be filled when the **Action** property is set to an action that is concerned by retrieving a value in a JS variable (retrieving the current tab index, "getindex" action, or number of tab, "getnbtab" action).

COOKIES GET 

OBJECT DESCRIPTION

Gets cookies from the Convertigo context.

This statement gets cookies from the Convertigo context and sets the value in a variable.

The value can be a string or an array of strings, depending on **Separator** property value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
Separator	String	standard	Defines the separator between cookies (empty for an array). This property allows to get a string concatenating all cookies with a separator. When no character is defined, the cookies are retrieved as an array of strings.
Variable	String	standard	Defines the variable in which the cookies will be retrieved. This variable can be a string or an array of strings, depending on **Separator** property value.

EXAMPLES

Let's consider a transaction similar to `GoToGoogleCom` transaction set in the context of the "*Starting With Convertigo Web Clipper*" *Quick Guide*, but defined in a Web Integrator project.

This transaction connects to Google website and, depending on the client country, navigates to Google.com page. When finally arrived on Google.com page, the transaction retrieves the cookies sent by the website and, for each cookie retrieved, displays a log line with its content.

To retrieve the cookies sent by Google website, a *Cookies Get* statement is created with the following parameters:

```
Cookies Get [
  separator=""
  variable=googleCookie
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

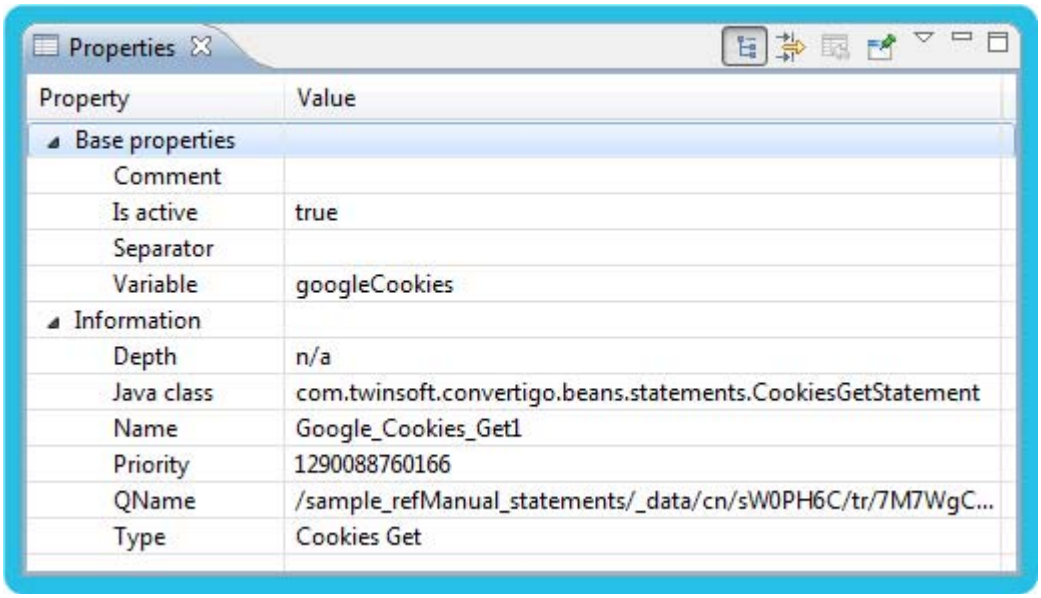


Figure 2 - 356: Cookies Get statement - Configuration example

The **Separator** property is set to an empty string, so the cookies are retrieved in the variable as an array of strings.

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and appear as follows in the **Projects** view:

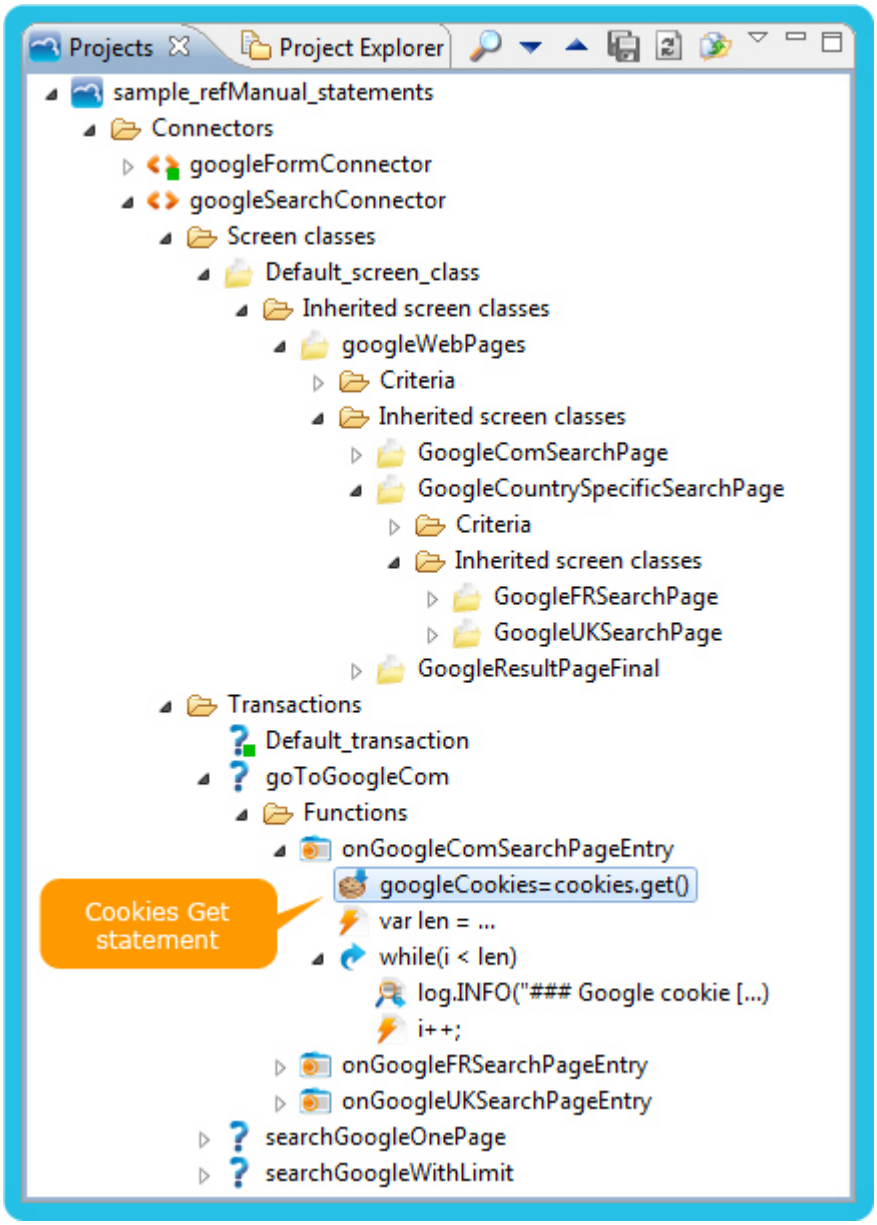


Figure 2 - 357: Cookies Get statement - Object in Projects view

When executing the `GoToGoogleCom` transaction, cookies are retrieved from the website and the Convertigo engine log displays the following lines:

Figure 2 - 358: Cookies Get statement - Log displaying retrieved cookies

Cookies Add 

Object Description

Adds cookie(s) in the Convertigo context.

This statement adds cookie(s) in the Convertigo context for them to be used between Convertigo and the remote website.

Several cookies can be added, each cookie must match the following format: "`$Domain=<my.domain>;$Path=</the/path/>;$Secure=true|false;$Date=<expiry date>;myName=myValue`".

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the JavaScript expression evaluated to give the cookie(s). This expression generates a string value (one cookie) or an array of string values (several cookies). Each cookie must match the following format: "`$Domain=<my.domain>;$Path=</the/path/>;$Secure=true\|false;$Date=<expiry date>;myName=myValue`".
Is active	boolean	standard	Defines whether the statement is active.

ADOPT CLIENT COOKIES 

OBJECT DESCRIPTION

Adopts client cookie(s) in the Convertigo context.

This statement adds all client cookie(s) sent by the client browser in the Convertigo context for them to be used between Convertigo and the remote website.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.

INJECT JS IN BROWSER 

OBJECT DESCRIPTION

Injects JavaScript code under the node matching the specified XPath.

This statement enables Convertigo to inject any JavaScript code in the target web page. It can then dynamically invoke any existing JavaScript code from the target page.

The *Inject JS in browser* statement dynamically creates a `SCRIPT` tag under the node designated by the **XPath** property, containing the JavaScript code specified by the **JS code** property, and executes it.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.
JS code	String	standard	Defines the JavaScript code to inject and invoke. Any JavaScript expression that will be valid for the page currently accessed by the HTML connector. This script is dynamically added as a `SCRIPT` tag under the node specified by the **XPath** property and evaluated. For calling a JavaScript function already defined in the page, do it using this syntax: `functionToBeCalled();`. You can also replace an existing JavaScript function by writing another JavaScript function with a similar name. This can be useful if a website features JavaScript functions unsupported by Convertigo's Mozilla/Firefox HTML parser. You can also pass Convertigo variables to your JavaScript code by using the **Variables** property.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

Property	Type	Category	Description
Variables	XMLVector	standard	Declares and initializes variables from Convertigo in injected JavaScript code. This property allows passing transaction's scope variables to the JavaScript code injected in the web page. These variables will be initialized with a value resulting from a JavaScript expression evaluated during the transaction execution (for example using transaction variables). These variables must be of standard types (for example `int`, `string`, etc.), complex types are not supported (for example `Array` or `DOM`). For each variable, three columns have to be set: • **Variable**: the variable name, • **Comment**: a comment to illustrate this variable, • **JS Expression**: the JavaScript expression to execute to give the variable value. Note: A new variable can be added to the list using the blue keyboard icon. The variables defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

GET URL 

OBJECT DESCRIPTION

Extracts the current page URL into a variable in JavaScript scope.

The *Get URL* statement extracts the web page URL and sets a JavaScript variable with this string.

Note: The JavaScript variable defined in scope thanks to this statement is a *String*.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
Variable name	String	standard	Defines the name of the JavaScript variable. If this variable exists in scope, its value is overridden. If the variable doesn't exist in scope, it is created.

EXAMPLES

Let’s consider the US directory website. Every page of this site contains a search form on its top:

Figure 2 - 359: Get URL statement - US directory website pages with search form

A `searchBusiness` transaction, which defines three variables named `business`, `cityZIPcode`, and `state`, sets those three values into search form input fields and launch a search by clicking on the `Search` button.

The transaction is able to fill the search form on the three pages of the US directory website that are identified by screen classes in the project. The actions to perform on every screen are defined thanks to a *Function* statement named `setSearchInputs` as follows:

- set Business input,
- set City / ZIP code input,
- select State in combobox,
- click on the `Search` button.

If the transaction starts on a page that is not recognized as belonging to a screen class defined in the project, we want to test whether the web page URL belongs to US Directory website or not. This would give the transaction the ability to display a custom error message depending on the test result.

To retrieve the page URL, a *Get URL* statement is created with the following parameters:

```
Get URL [
   variable name=pageUrl
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

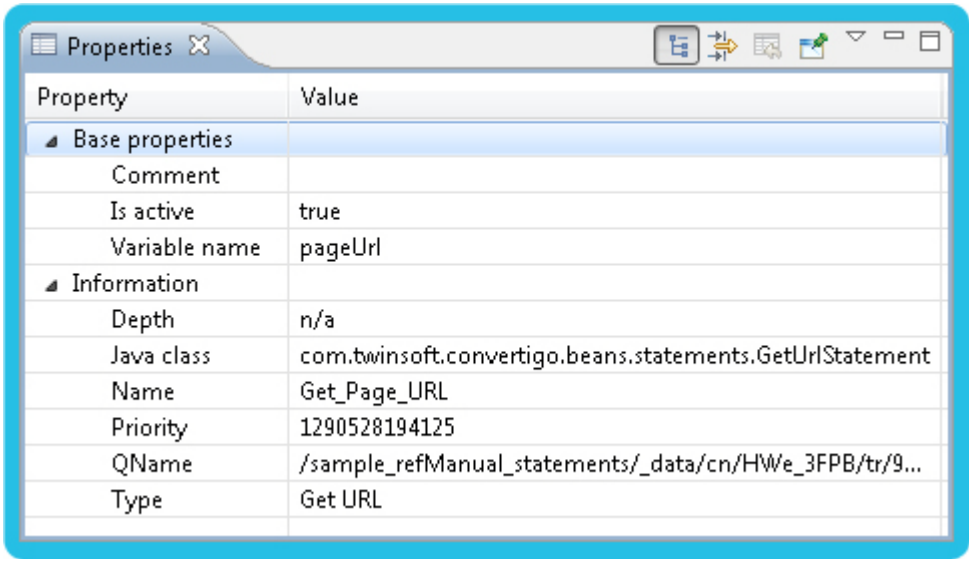


Figure 2 - 360: Get URL statement - Configuration example

It is created in the **Functions** folder of the transaction, in the `Default_screen_class` entry handler. In fact, this screen class will match any web page not belonging to project-defined screen classes. This statement is created next to other statements used to implement the transaction behavior described above and appears as follows in the **Projects** view:

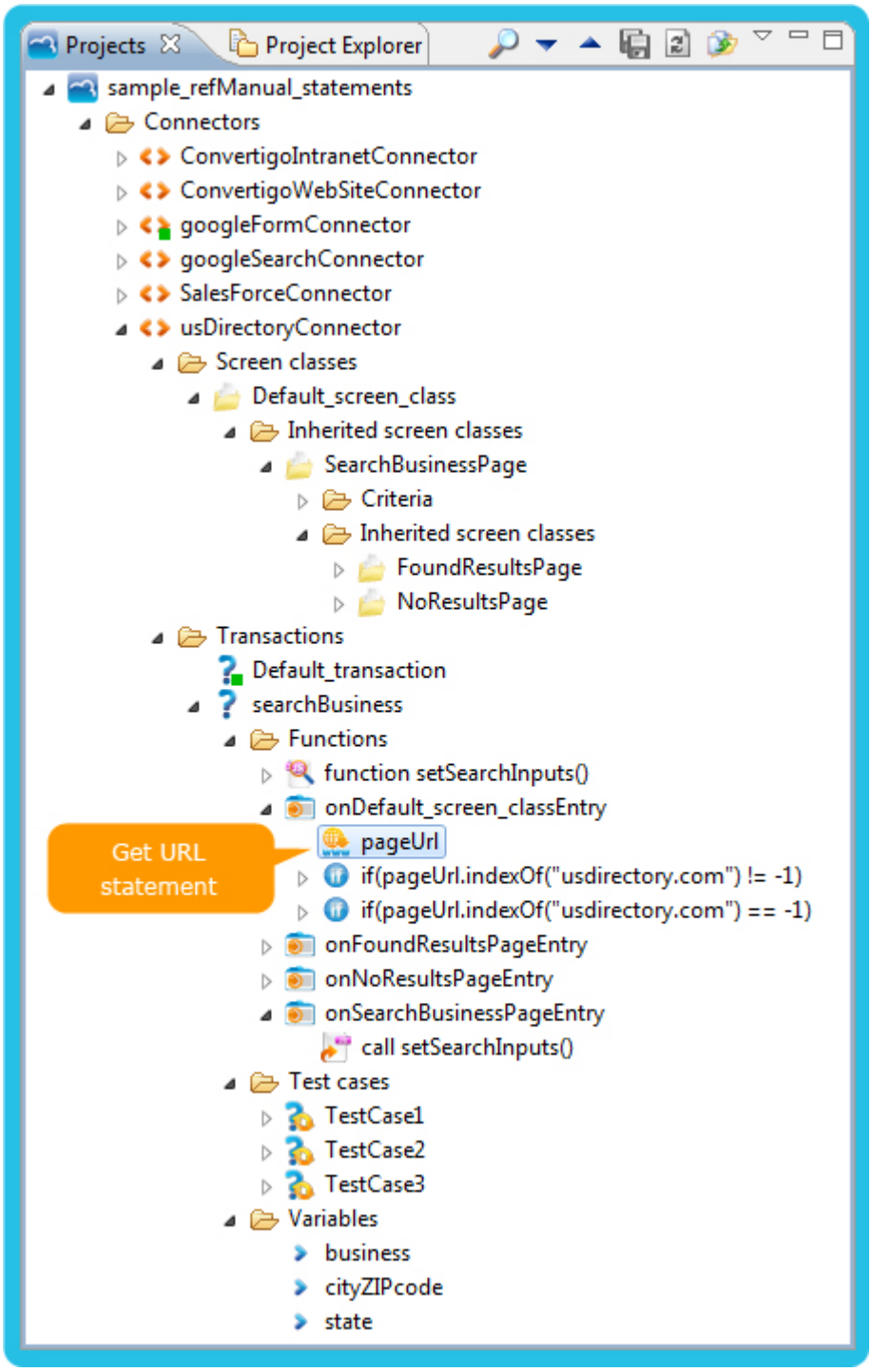


Figure 2 - 361: Get URL statement - Object in Projects view

When diplaying Google search page in the web browser and then executing one of the test cases defined for the transaction, the statements are executed so that an error message is displayed in the transaction XML output.

GET ATTACHMENT 

OBJECT DESCRIPTION

Downloads an attachment file.

This statement instructs Convertigo to wait for an attachment to be fully downloaded. This statement requires the Convertigo internal browser to be set so as to receive attachments, by using an appropriate *Browser property change* statement.

The required property must be changed prior to executing the action that triggers the download (click on a download link for example). The property change can be included either in a previous transaction or in a handler executed prior to the download.

Important note: *Get attachment* statement must be executed immediately after the action that triggers the download.

Once the download is complete, the final destination for the retrieved document depends on the **Attachment recovery policy** property value:

- If the property is set to `localfile_<whatever_value>`, the file is stored locally in a temporary file, using the path defined by the **File path** property.
- If the property is set to `base64`, the file is stored in memory encoded in base 64.

Note: The file can be sent back to the client afterwards as binary data with the correct MIME type: the client should request Convertigo `.bin` requester (see the *"Interfaces to Convertigo"* chapter of the Reference Manual, *"HTTP protocol interface to Convertigo"* section, *"Convertigo URLs">"Convertigo requesters"* paragraph).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attachment recovery policy	Policy	standard	Defines how and where to recover the attachment file. This property can take one of the following values: • `localfile_increment`: stores the downloaded file locally on the server, using the **File path** property; if a file with the same name has already been downloaded, it increments the name of the file with a number in order not to lose previous version, • `localfile_override`: stores the downloaded file locally on the server, using the **File path** property; if a file with the same name has already been downloaded, it overrides the previously downloaded file with the new version, • `base64`: stores the downloaded file in memory, encoded in base 64. The `localfile_override` policy is the default value for this property.

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
File path	JS expression	standard	JavaScript expression defining the file path, including the file name, of the file to get (optional). If set, this property allows to override the initial file name and to define the path to the directory where the file must be get. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. If not set, a default path is automatically generated: `.//downloads/<original_name_from_the_server>`. This path automatically creates a `downloads` folder in the project (if not existing) and stores in it the file with its original name suggested by the server.
Is active	boolean	standard	Defines whether the statement is active.
Threshold	long	standard	Defines the downloading activity threshold in millisecond. The *HTML connector* cannot know when a file download is terminated. On the other side, the *HTML connector* knows about downloading activity. This property allows to define a maximum time of inactivity to wait after the last downloading activity. If this threshold time is reached, the file download is considered as finished.
Timeout	long	standard	Defines the download timeout in millisecond. This property allows to define the maximum time to wait for the file to be downloaded. If the file download is not finished at the end of this time, the download is aborted.

EXAMPLES

Let's consider the Convertigo website.

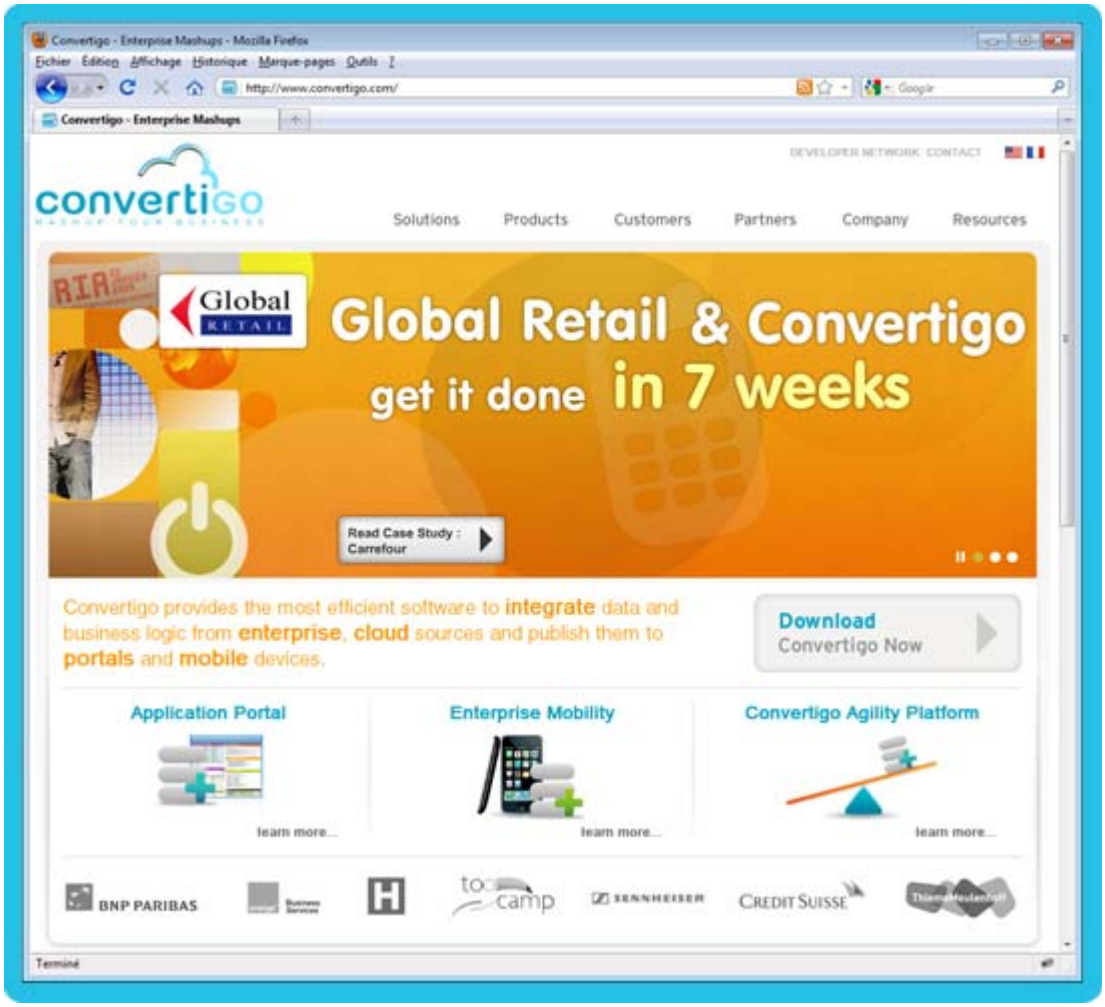


Figure 2 - 362: Get attachment statement - Convertigo website

A `downloadOperatingGuidePDFdocumentation` transaction is developed to:

- connect to Convertigo website,
- navigate through the Developer Network pages,
- and download the Operating Guide documentation PDF file, storing it to the Convertigo project.

The navigation through the website is performed thanks to several statements. While arrived on the target **Convertigo Documentation** page, the transaction initiates the file download by clicking on the appropriate link thanks to a *Mouse action* statement. This statement is synchronized thanks to a `Download Started` synchronizer. For more information, see *Mouse action* statement documentation and examples.

Then, to download the PDF file, a *Get attachment* statement is created with the following parameters:

```
Get attachment [
   attachment recovery policy:localfile_override
   file path:
   timeout=300000
   threshold=1000
```

]

These parameters are edited in the **Properties** view of the Convertigo Studio:

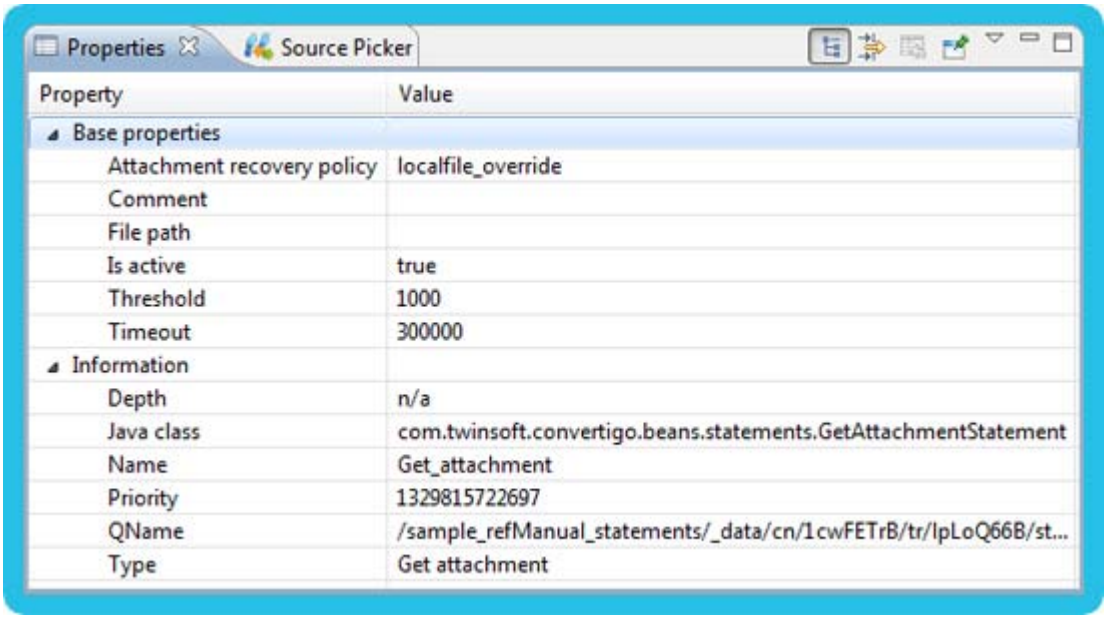


Figure 2 - 363: Get attachment statement - Configuration example

It is created in the **Functions** folder of the transaction, in the appropriate screen class entry handler. This statement is created just after the *Mouse action* statement that initiates the download, as described above, and appears as follows in the **Projects** view:

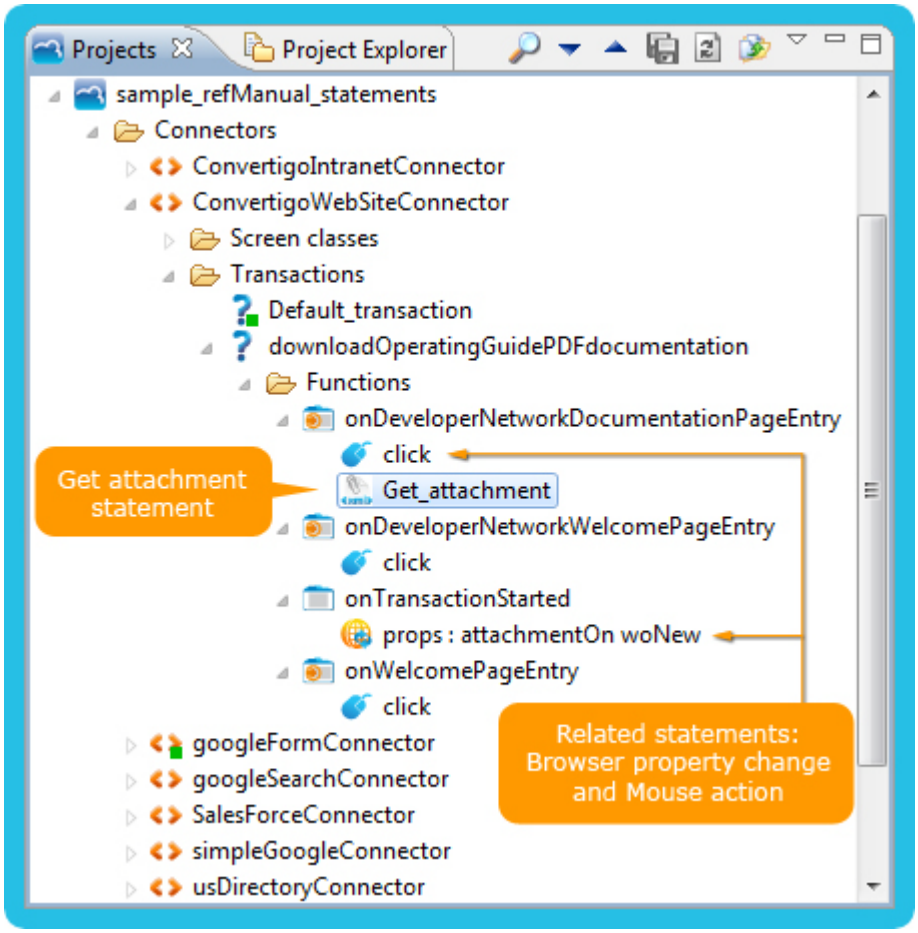


Figure 2 - 364: Get attachment statement - Objects in Projects view

When starting the transaction, on the `onTransactionStarted` handler, a *Browser property change* statement is added in order to change Convertigo internal Web browser's `attachment download` property. This property set in the Convertigo internal Web browser is mandatory for any transaction that downloads an attachment thanks to the *Get attachment* statement. For more information, see the *Browser property change* statement documentation and examples.

Executing the transaction automatically downloads the PDF file in default `downloads` folder at project's root, as no **File path** property is set for this *Get attachment* statement. Switch to the **Project Explorer** view, the `downloads` folder as well as the file are present.

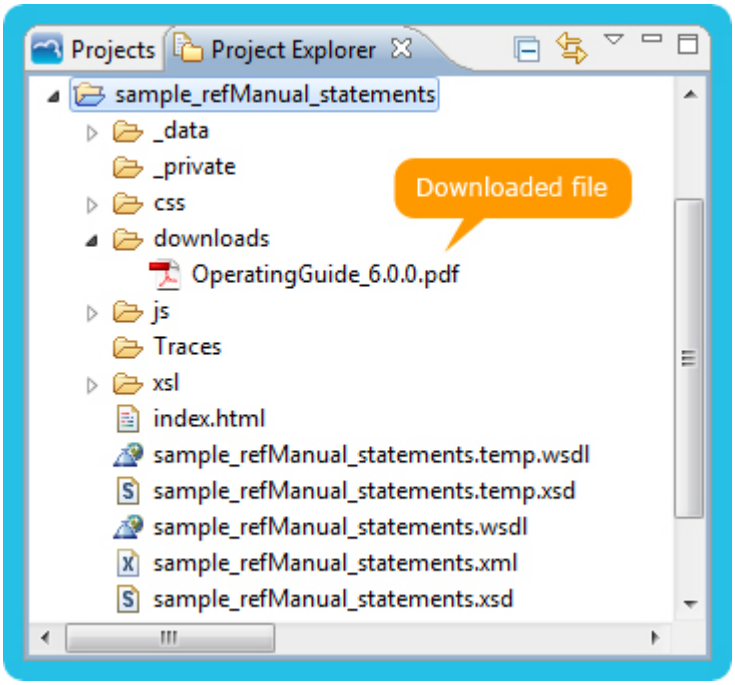


Figure 2 - 365: Get attachment statement - Downloaded file in Project Explorer view

OTHERS

HTTP UPLOAD REQUEST 

OBJECT DESCRIPTION

Sends an HTTP upload request.

This statement simulates an HTTP `POST` request on the target application with a content type `multipart/form-data` to upload a file.

Note: You can add *HTTP statement variable* objects to this statement, they will be sent as HTTP request parameters (for more information see *HTTP single-valued variable* and *HTTP multi-valued variable* documentation).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
File path	JS expression	standard	JavaScript expression defining the file path, including the file name, of the file to upload. This file must be a local file. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder.
HTTP Filename	JS expression	standard	JavaScript expression defining the HTTP file name of the file to upload. If empty, the name of the local file is sent as HTTP name.
HTTP headers	XMLVector	expert	Defines the HTTP headers to be used in the HTTP request. This property allows the user to define specific HTTP headers for the HTTP request. Each header is defined with the following items: • **Variable**: HTTP header name (ex: `Content-Type`). • **Value**: HTTP header value (ex: `application/x-www-from-urlencoded`). Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
Host	String	standard	Defines the host address. By default, *HTTP request* statements use the **Host** property set in the connector object. This property can be overridden by setting the **Host** property here. The **Host** property can contain a DNS host name or be a simple IP address.
Is HTTPS	boolean	standard	Defines whether the connection is secured (HTTPS).

Property	Type	Category	Description
Is active	boolean	standard	Defines whether the statement is active.
Port	int	standard	Defines the server port. By default, *HTTP request* statements use the **Port** property set in the connector object. This property can be overridden by setting the **Port** property here.

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

URI	JS expression	standard	Defines as a JavaScript expression the URI to be used in the HTTP request. This property is a JavaScript expression that is evaluated during the transaction execution and gives the URI string to be used in the HTTP request.

Property	Type	Category	Description
URL charset encoding	String	expert	Defines the charset encoding to use for the variable values sent as parameters in HTTP request. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is blank. If blank, the parent transaction's **URL charset encoding** property value is used.

HTTP REQUEST 

OBJECT DESCRIPTION

Sends an HTTP request.

This statement simulates an HTTP `POST` or `GET` request on the target application.

Note: You can add *HTTP statement variable* objects to this statement, they will be sent as HTTP request parameters (for more information see *HTTP single-valued variable* and *HTTP multi-valued variable* documentation).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
HTTP headers	XMLVector	expert	Defines the HTTP headers to be used in the HTTP request. This property allows the user to define specific HTTP headers for the HTTP request. Each header is defined with the following items: • **Variable**: HTTP header name (ex: `Content-Type`). • **Value**: HTTP header value (ex: `application/x-www-from-urlencoded`). Note: A new HTTP header can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon.
HTTP verb	int	standard	Allows to choose the HTTP verb to use for this HTTP request: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `TRACE`, `OPTIONS` or `CONNECT`. For more information about HTTP verbs, you can visit the following RFC page: *http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*.property.https.display_name=Is HTTPS
Host	String	standard	Defines the host address. By default, *HTTP request* statements use the **Host** property set in the connector object. This property can be overridden by setting the **Host** property here. The **Host** property can contain a DNS host name or be a simple IP address.
Is HTTPS	boolean	standard	Defines whether the connection is secured (HTTPS).
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Port	int	standard	Defines the server port. By default, *HTTP request* statements use the **Port** property set in the connector object. This property can be overridden by setting the **Port** property here.

Synchronization	TriggerXMLizer	expert	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

URI	JS expression	standard	Defines as a JavaScript expression the URI to be used in the HTTP request. This property is a JavaScript expression that is evaluated during the transaction execution and gives the URI string to be used in the HTTP request.

Property	Type	Category	Description
URL charset encoding	String	expert	Defines the charset encoding to use for the variable values sent as parameters in HTTP request. This property allows to define the charset encoding used to URL-encode the parameter values: • GET parameters for the query string, • POST parameters in case of `application/x-www-form-urlencoded` content-type. Default value is blank. If blank, the parent transaction's **URL charset encoding** property value is used.

EXAMPLES

This example is based on a connector, reaching Google website, defined in a project named `sample_refManual_variables`.

*You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open **Samples > Reference Manual examples > Variables examples** in the **New Project** wizard.*

Let's consider an *HTML transaction*, named `searchYahoo`, similar to `searchGoogle` transaction developed in the context of the "*Starting with Convertigo Web Integrator*" Quick Guide, but performing the research on Yahoo website instead of Google.

It declares an *HTTP single-value variable*, named `keyword`, with a **Default value** property set to "`Convertigo`". For more information about *HTML transaction* or *HTTP single-valued variable* object, see "*HTML transaction*" and "*HTTP single-valued variable*" documentation and example.

The `searchYahoo` transaction appears as follows in the **Projects** view of the Convertigo Studio:

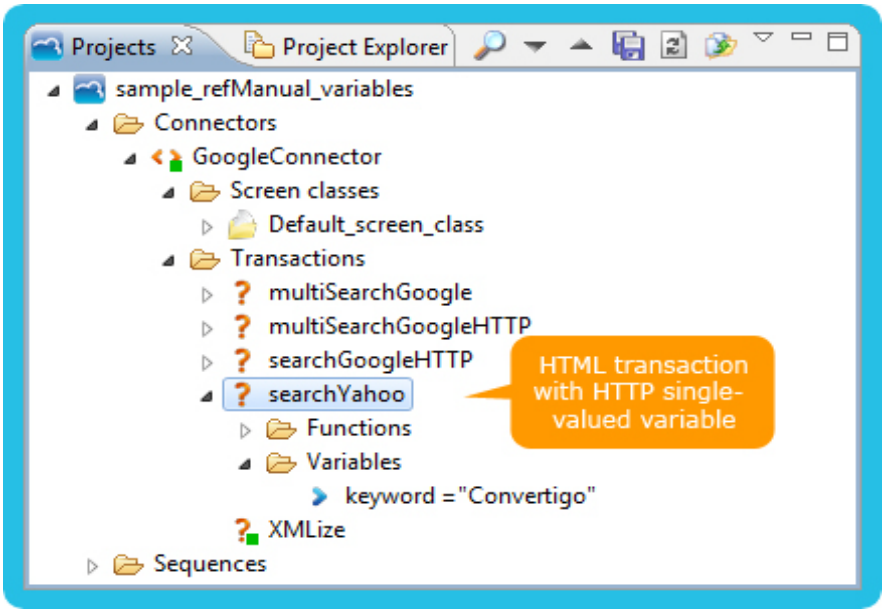


Figure 2 - 366: HTTP request statement - HTML transaction with HTTP variable in Projects view

The `searchYahoo` transaction is created with its **Subpath** property left empty, so it starts executing its statements after the connector has established the connection to target website.

Being in a connector reaching Google website, the transaction has to redirect to Yahoo website when Google search page is loaded. Then, it makes the research on the given keyword.

The Yahoo request URL for a research is:

`http://fr.search.yahoo.com/search?p=<keyword>`

where p is the keyword HTTP parameter.

To perform the redirection, an *HTTP request* statement is created on the *Entry handler* matching Google search page, with the following parameters:

```
HTTP request [
   isHTTPs=false
   port=80
   host=fr.search.yahoo.com
   URI="/search"
   synchronization=[document completed: 1, timeout= 60000 ms]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

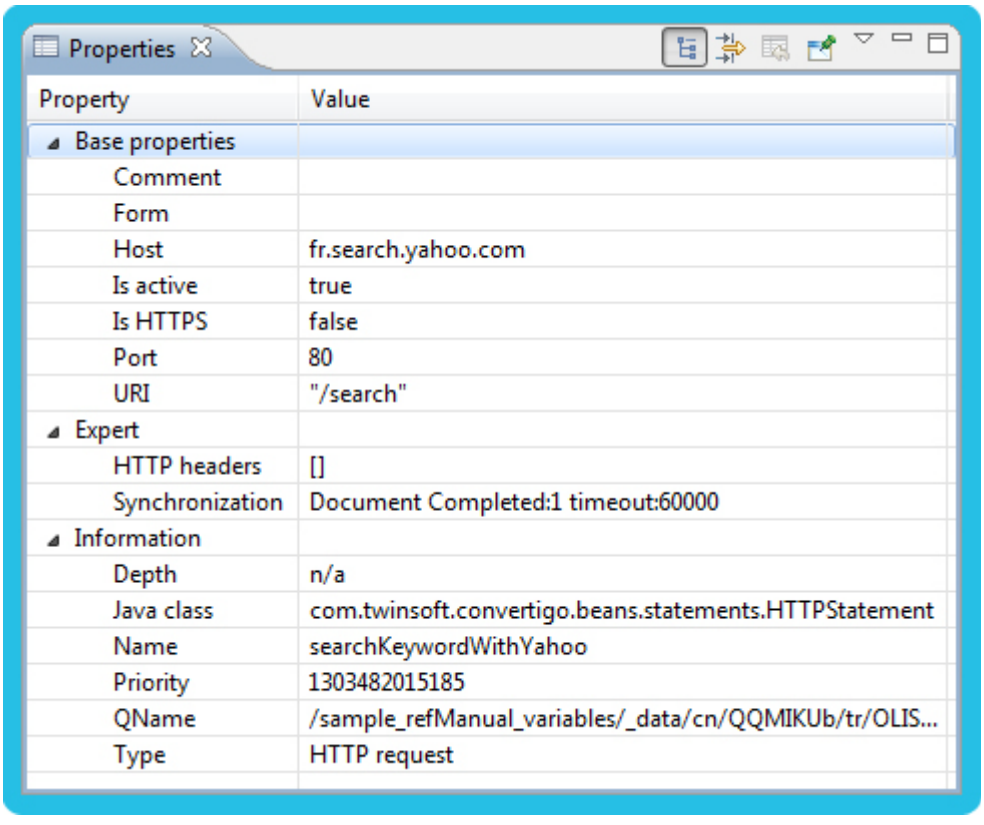


Figure 2 - 367: HTTP request statement - Configuration example

The **URI** property is appearing between double-quotes because this property is a JavaScript expression.

The **Synchronization** property indicates that, after triggering the action, the statement waits

for *one* document to be fully loaded before continuing the transaction execution. This property is edited in the **Trigger editor**:

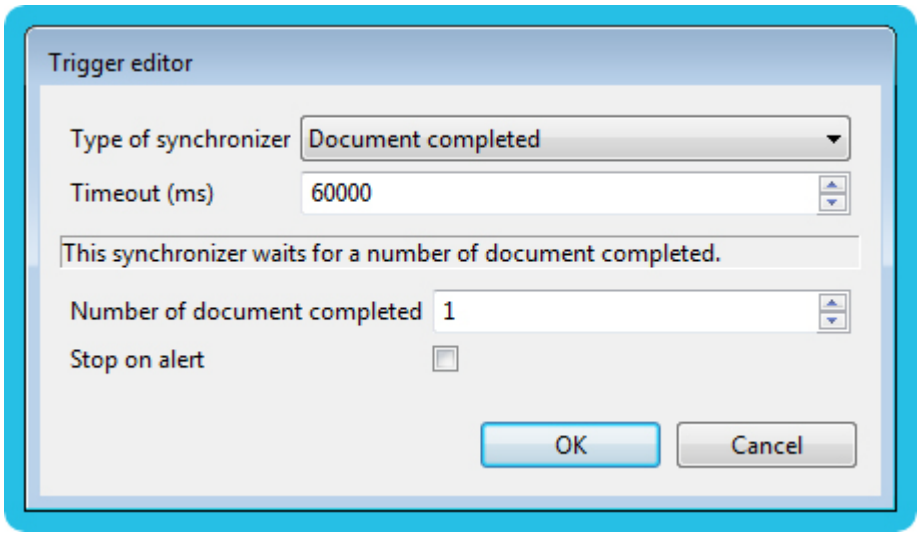


Figure 2 - 368: HTTP request statement - Synchronization property edition

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and appears as follows in the **Projects** view:

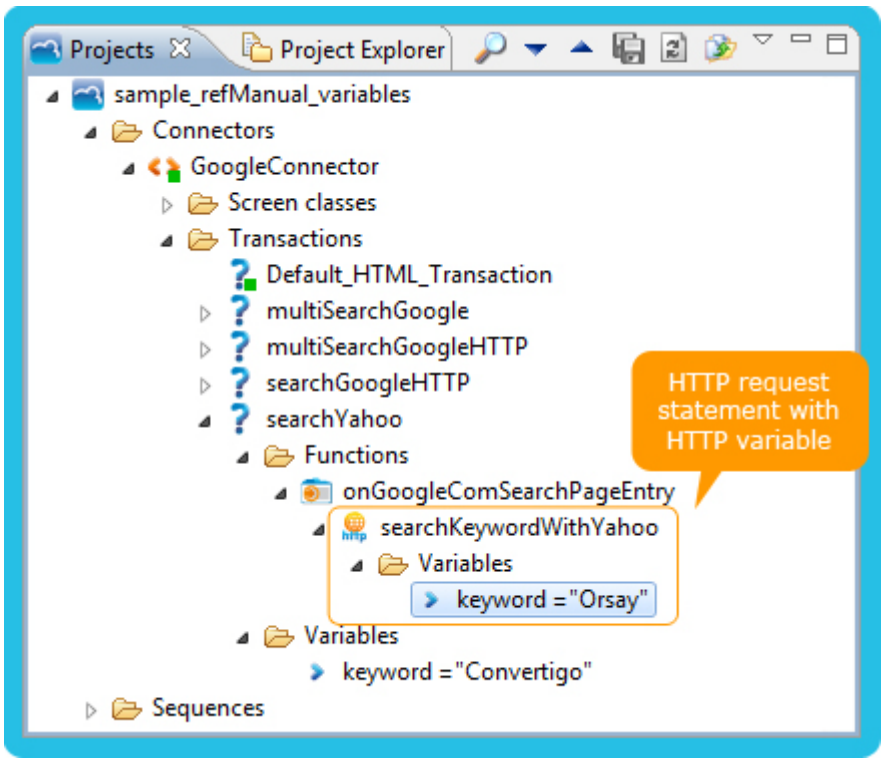


Figure 2 - 369: HTTP request statement - Object in Projects view

The *HTTP request* statement may declare variables, in order to send them as request parameters. In this case, the statement declares an *HTTP single-valued variable*, named `keyword`, in order to send the searched keyword as an HTTP request parameter to the Yahoo research. For more information about the *HTTP single-valued and multi-valued variables* for statements, see "*HTTP single-valued variable*" and "*HTTP multi-valued variable*" documentation and examples.

The transaction is implemented and can be tested in the Studio. To do so, the first step is to open the connector editor by double-clicking on the *HTML connector* in the **Projects** view (if not already open). The internal browser connects to `http://www.google.com/` as seen in **Design** tab of the editor:

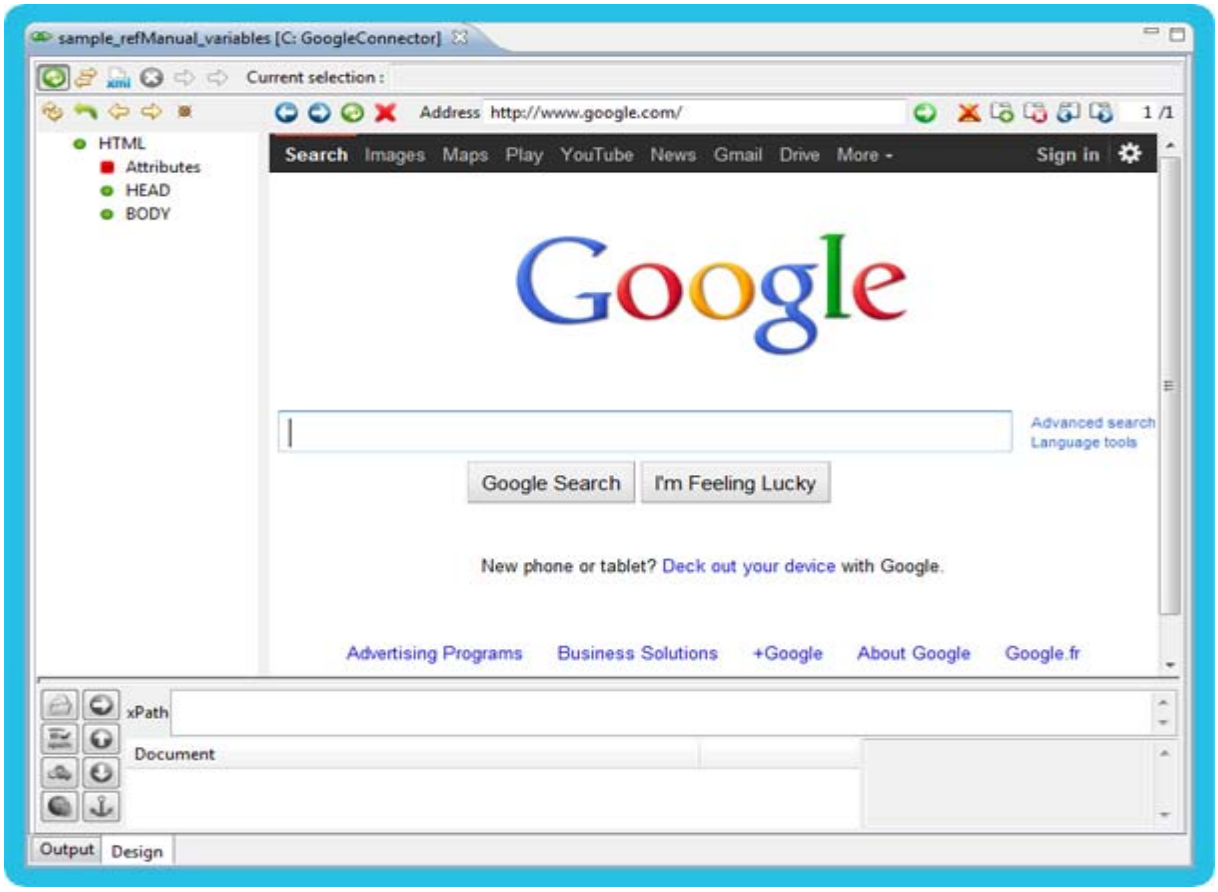


Figure 2 - 370: HTTP request statement - Connector editor

Now execute the `searchYahoo` transaction by pressing F5 key on the transaction object in the **Projects** view. The internal browser connects to `http://fr.search.yahoo.com/search?p=Convertigo` as seen in **Design** tab of the editor:

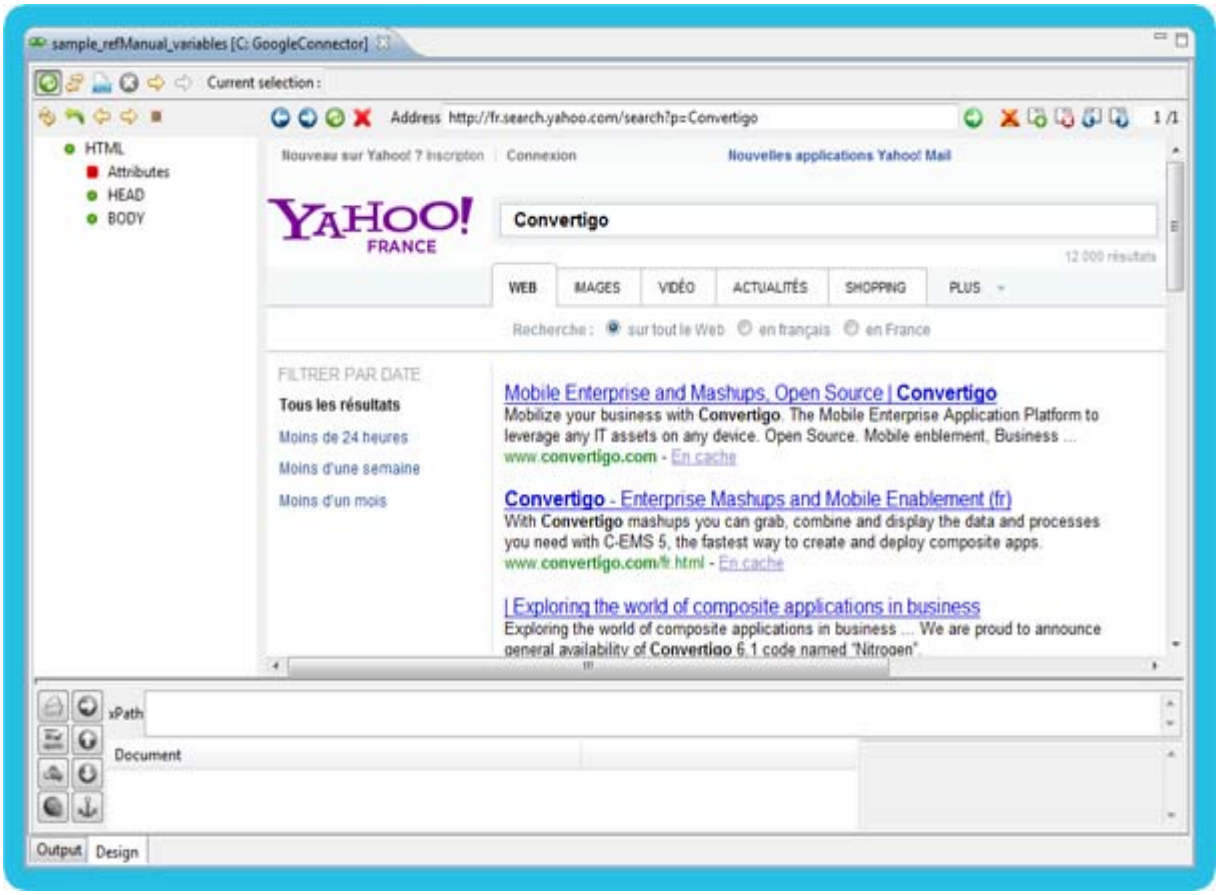


Figure 2 - 371: HTTP request statement - Connector editor after transaction execution

WHAT HAPPENED?

At the start of the transaction, its `keyword` variable was added to the JavaScript scope.

The connector built the request URL by concatenating the server address, the transaction **Subpath** (empty) and an empty query string because the `keyword` variable hasn't specified any parameter name through its **HTTP name** property. Thus, it simply requested `http://www.google.com/`.

The Google search page screen class was detected thanks to its criteria and the corresponding entry handler was executed.

The *HTTP request* statement built the request URL by concatenating its **Host** value, **URI** value and a query string. This query string was formed from statement's variables details.

EXCEPTION 

OBJECT DESCRIPTION

Raises a Convertigo Engine exception.

In some circumstances, it is necessary to explicitly raise a Convertigo Engine exception. This is reflected as a `SoapException` for SOAP web service callers or by an XML error for any other caller.

Expression property can be set to a complex JavaScript expression, mixing text strings and data from variables. This expression will be evaluated during the transaction execution and will build a dynamic message output in the raised exception.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the statement value. This property is a JavaScript expression that is evaluated during the transaction execution and gives the statement's result.
Is active	boolean	standard	Defines whether the statement is active.

EXAMPLES

Let’s consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

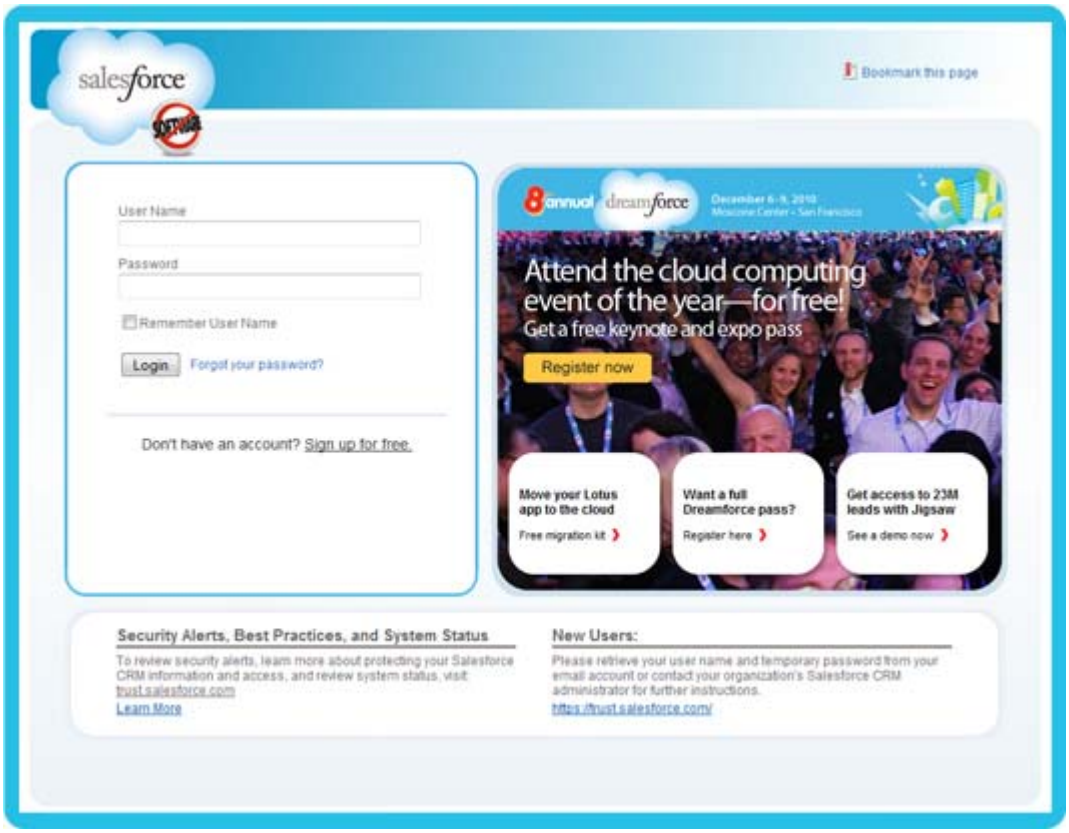


Figure 2 - 372: Exception statement - SalesForce website authentication page

A `Login` transaction, which defines two variables named `username` and `password`, is written to authenticate the user on SalesForce website. This transaction:

- fills the User Name and Password fields using the `user` / `password` variables,
- submits the authentication form to access to SalesForce welcome page.

When the authentication is not correct, SalesForce website displays an error message, recognized by Convertigo as the `LoginFailedPage` screen class.

When the username / password is not provided or the authentication is not correct, we want to raise a Convertigo Engine Exception. To do so, two *Exception* statements are created in the transaction, with the following parameters:

- on the transaction start handler, when username is not received or empty:

```
Exception [
   expression: "Authentication impossible: username empty or null"
]
```

- on the `LoginFailedPage` screen class handler, when the authentication has failed:

```
Exception [
   expression: "Authentication error: "+
               errorMessage.item(0).getNodeValue()
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

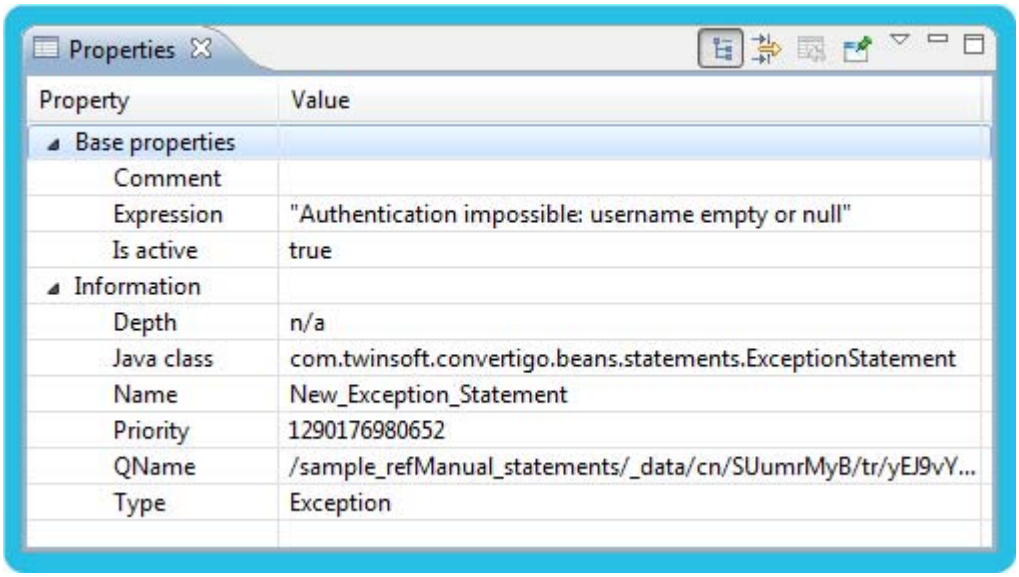


Figure 2 - 373: Exception statement - Configuration example

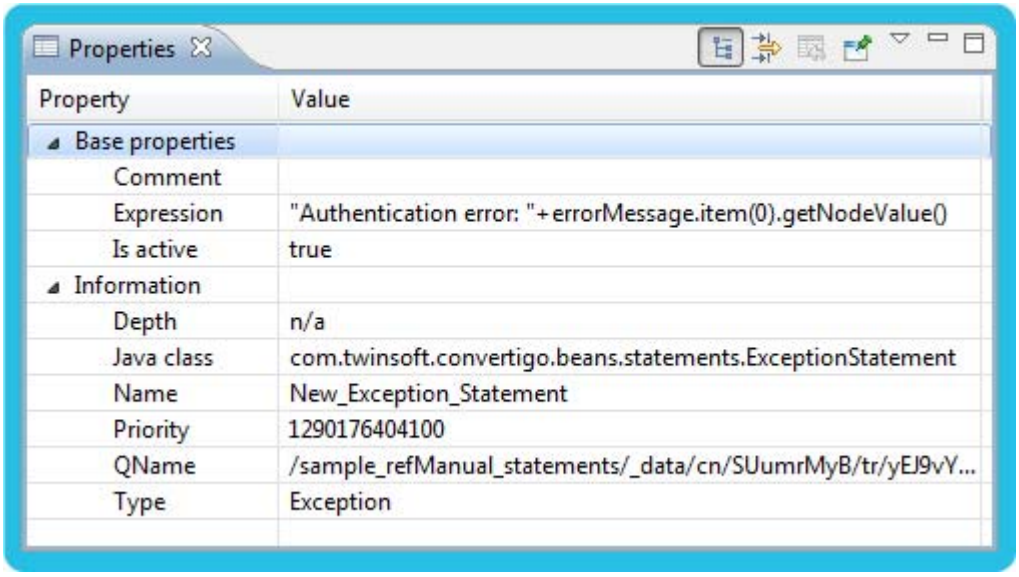


Figure 2 - 374: Exception statement - Configuration example

The **Expression** property is set to a simple text JavaScript expression for the first statement, that will be evaluated unchanged during the transaction execution. For the other statement, the **Expression** property is set to a more complex JavaScript expression, mixing a text string and data from a variable, in order to build a dynamic message when evaluated during the transaction execution.

The statements are created in the **Functions** folder of the transaction, under the corresponding handlers and other statements, and appear as follows in the **Projects** view:

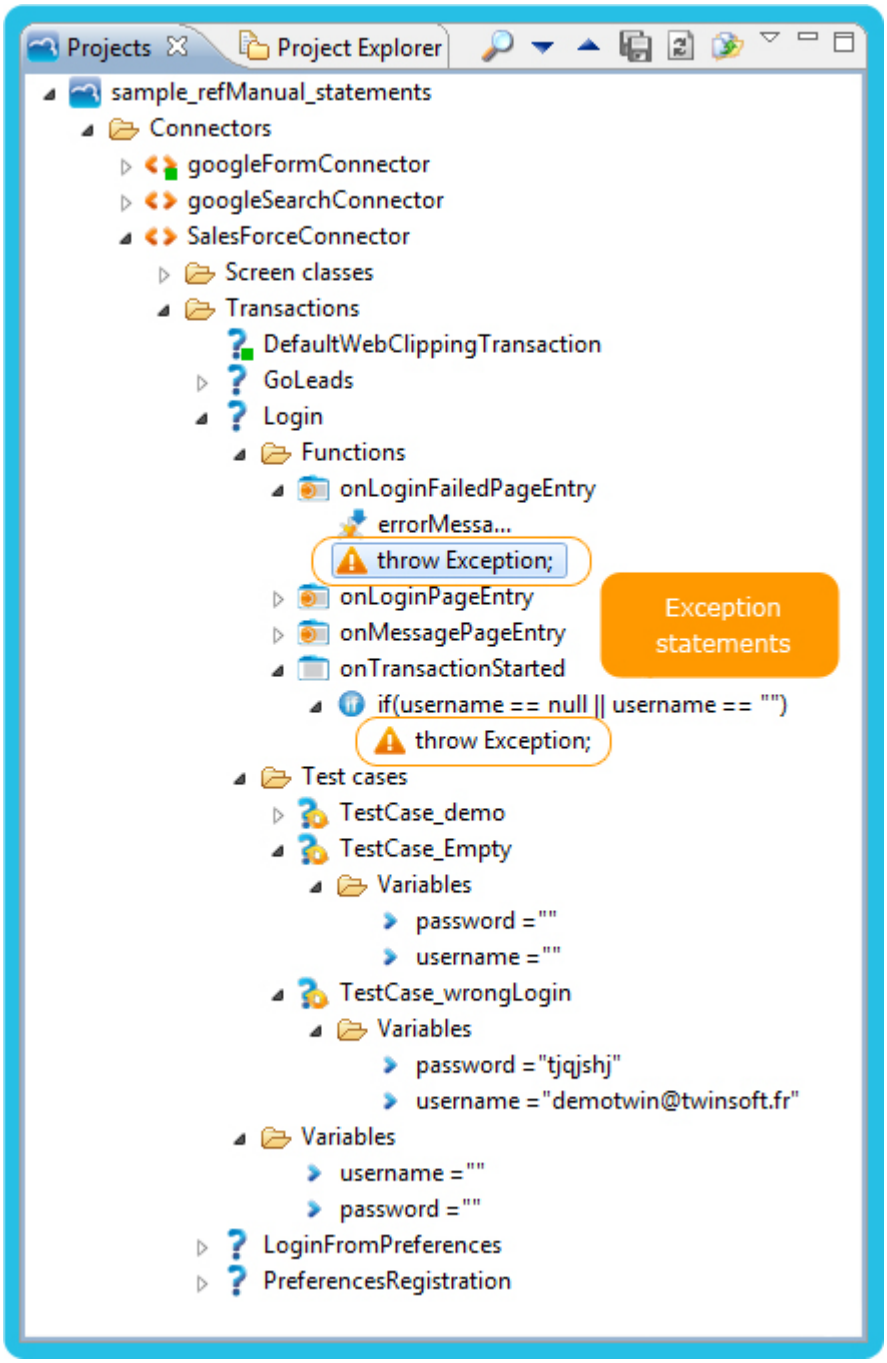


Figure 2 - 375: Exception statement - Objects in Projects view

When executing the test cases named `TestCase_Empty` or `TestCase_wrongLogin` defined for the `Login` transaction, the transaction raises an Engine Exception with the appropriate message. They are visible in the Engine log:

```
[com.twinsoft.convertigo.engine.EngineException] A statement exception has been raised
java.lang.Throwable: Username empty or null: authentication to SalesForce not possible
     at com.twinsoft.convertigo.beans.statements.ExceptionStatement.execute(ExceptionStatement.java:55)
     at com.twinsoft.convertigo.beans.core.StatementWithExpressions.executeNextStatement(StatementWithExpressions.java:324)
     at com.twinsoft.convertigo.beans.statements.BlockStatement.executeNextStatement(BlockStatement.java:87)
     at com.twinsoft.convertigo.beans.statements.IfStatement.execute(IfStatement.java:67)
     at com.twinsoft.convertigo.beans.core.StatementWithExpressions.executeNextStatement(StatementWithExpressions.java:324)
     at com.twinsoft.convertigo.beans.statements.FunctionStatement.executeNextStatement(FunctionStatement.java:77)
     at com.twinsoft.convertigo.beans.statements.FunctionStatement.execute(FunctionStatement.java:64)
     at com.twinsoft.convertigo.beans.transactions.HtmlTransaction.executeSimpleHandlerCore(HtmlTransaction.java:829)
     at com.twinsoft.convertigo.beans.core.Transaction.executeHandlerCore(Transaction.java:382)
     at com.twinsoft.convertigo.beans.transactions.HttpTransaction.executeHandlerCore(HttpTransaction.java:247)
     at com.twinsoft.convertigo.beans.transactions.HtmlTransaction.executeHandlerCore(HtmlTransaction.java:720)
     at com.twinsoft.convertigo.beans.core.Transaction.executeHandler(Transaction.java:340)
     at com.twinsoft.convertigo.beans.core.Transaction.handleRequestableEvent(Transaction.java:211)
     at com.twinsoft.convertigo.beans.core.RequestableObject$RequestableThread.run(RequestableObject.java:725)
```

Figure 2 - 376: Exception statement - Exception and message visible in Engine log

```
[com.twinsoft.convertigo.engine.EngineException] A statement exception has been raised
java.lang.Throwable: Authentication error:
                     Your login attempt has failed. The username or password may be incorrect, or your location or login time may be restricted.
                     Please contact the administrator at your company for help.
    at com.twinsoft.convertigo.beans.statements.ExceptionStatement.execute(ExceptionStatement.java:55)
    at com.twinsoft.convertigo.beans.core.StatementWithExpressions.executeNextStatement(StatementWithExpressions.java:324)
    at com.twinsoft.convertigo.beans.statements.FunctionStatement.executeNextStatement(FunctionStatement.java:77)
    at com.twinsoft.convertigo.beans.statements.FunctionStatement.executeNextStatement(FunctionStatement.java:79)
    at com.twinsoft.convertigo.beans.statements.FunctionStatement.execute(FunctionStatement.java:64)
    at com.twinsoft.convertigo.beans.transactions.HtmlTransaction.executeHandlerCore(HtmlTransaction.java:765)
    at com.twinsoft.convertigo.beans.core.Transaction.executeHandler(Transaction.java:340)
    at com.twinsoft.convertigo.beans.transactions.HtmlTransaction.runCore(HtmlTransaction.java:459)
    at com.twinsoft.convertigo.beans.core.RequestableObject$RequestableThread.run(RequestableObject.java:728)
```

Figure 2 - 377: Exception statement - Exception and message visible in Engine log

GET NODES 

OBJECT DESCRIPTION

Extracts nodes from current HTML into a variable in JavaScript scope.

The *Get nodes* statement extracts a list of nodes from the web page DOM and sets a JavaScript variable in the current executed transaction JavaScript scope. This variable contains a `Java NodeList` object, i.e. a list of XML nodes, extracted from the HTML page thanks to the execution of an XPath on the page DOM. This XPath is defined in **XPath** property.

The variable is named after the **Variable name** property value. It exists while the transaction is running.

If only one node matches, the variable is also a `NodeList` containing only one `Node` (index is 0). If no node matches, the variable is finally an empty `NodeList`, containing no `Node` (`var_name.getLength() = 0`).

Notes:

- The variable contains a list of node elements of the DOM. To access one (`Node`) of the list, use the following syntax in a statement: `var_name.item(index)`.
- To access one element's text content (`String`), use the `element.getTextContent()` method, to retrieve the text of the element, or the `element.getNodeValue()` method, which result depends on the node type (will extract a text only if the `Node` is of `Text` or `Attribute` type).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Variable name	String	standard	Defines the name of the JavaScript variable. If this variable exists in scope, its value is overridden. If the variable doesn't exist in scope, it is created.
XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

EXAMPLES

Example 1

Let's consider a `googleSearch` transaction that implements a simple Google research on Google website. It is created in a `simpleGoogleConnector` connector, containing only two screen classes (`googleSearchForm` and `googleResults`).

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

The `googleSearch` transaction, defining a variable named `keyword`, has for purpose to input this keyword into the search input field in the `googleSearchForm` screen class and validate the research by clicking on the **Google Search** button. Then, it extracts the results in a table XML structure from `googleResults` screen class, and:

- if present, clicks on the **Next** link to pass to the next page and accumulate the data,
- if not present, stops.

In order to test the presence of the **Next** link on the `googleResults` screen class before clicking on it, a *Get nodes* statement is created with the following parameters:

```
Get nodes [
   variable name=nextButtonNodeList
   xpath='//A[@id="pnnext"]'
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

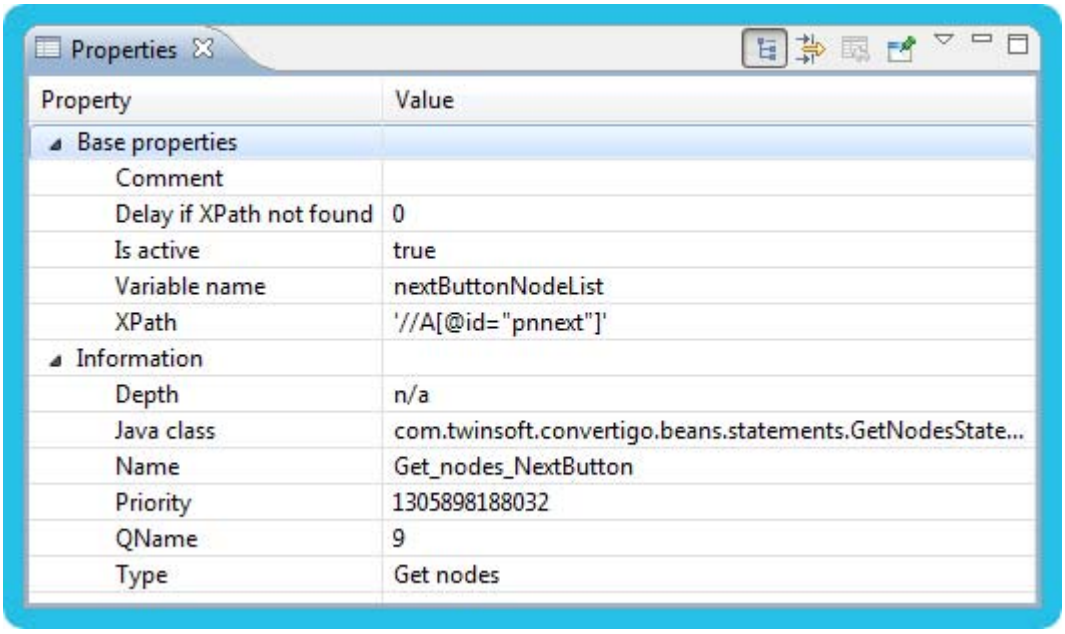


Figure 2 - 378: Get nodes statement - Configuration example

The **Xpath** property is set to a simple String expression matching an `A` link node with an id attribute that is `"pnnext"`. The **Variable name** property is set to `nextButtonNodeList` for the programmer to keep in mind that the generated variable is a Java `NodeList`.

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler, with various other statements used to implement the transaction behavior described above. It appears as follows in the **Projects** view:

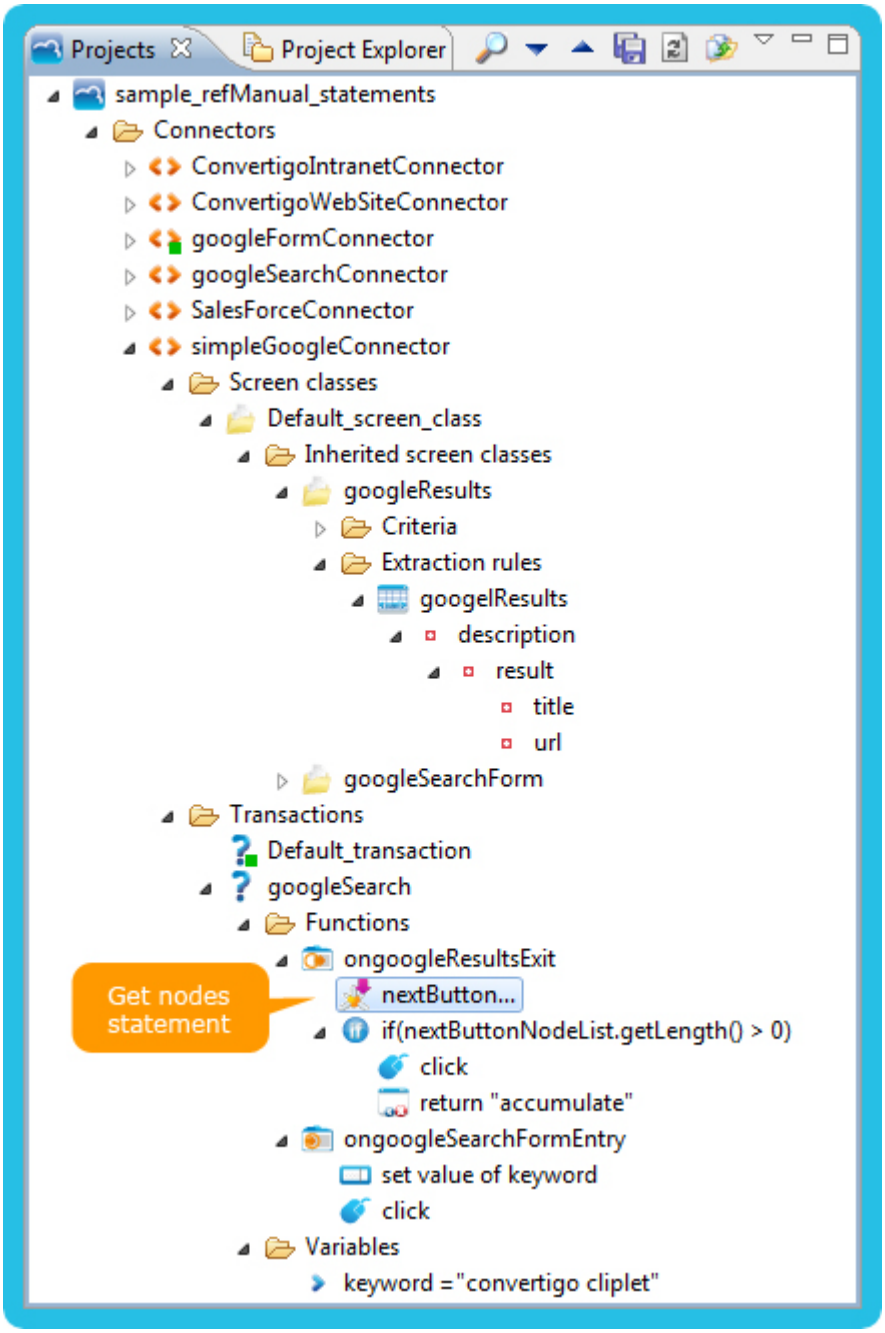


Figure 2 - 379: Get nodes statement - Object in Projects view

On each screen class detection, after the variable is set in current JavaScript scope by the execution of this statement, the next statement tests if the variable contains a node or not in order to:

- click on the link and continue by accumulating data,
- or stop the transaction, as previously explained.

This *If* statement uses the following syntax to test the length of the extracted `NodeList` object: `nextButtonNodeList.getLength() > 0`. When the length is superior to 0, it contains at least one `Node`: the transaction can continue.

The variable defined for the transaction contains a default value "`convertigo cliplet`" that gives only a few pages of results on Google. When executing the transaction, it searches for

the keyword, accumulates the 2 or 3 pages of results and stops.

Example 2

Let's consider a `fillForm` transaction, which defines one multivaluated variable named `inputs`. This transaction sets input field values into a web page containing a `FORM` HTML element. It is created in a `googleFormConnector` connector, containing only one screen class (`Form_Screen_class`).

You can find the complete example project in the Studio. To open this project, refer to the procedure "Opening a sample project from the Studio", choosing to open ***Convertigo Samples and Demos*** *>* ***Reference Manual examples*** *>* ***HTML connector statements examples*** *in the* ***New Project*** *wizard.*

As the transaction doesn't know in advance the number of inputs to fill in the `FORM`, it is implemented to loop on each element of the `inputs` variable, dynamically received from the caller, and to set each value in the nth HTML `INPUT` element of the web page. When there is no more `INPUT` element to fill, the transaction exits the loop and ends.

For each iteration of the loop, the transaction has to check if an nth `INPUT` element exists in the page. To do so, a *Get nodes* statement is created to try getting the nth `INPUT` element in a JavaScript variable. This statement is created with the following parameters:

```
Get nodes [
   variable name=inputNodeList
   xpath='//INPUT[not(@type) or (@type!="submit" and @type!="hidden")]
   [' + iterator+1 + ']'
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

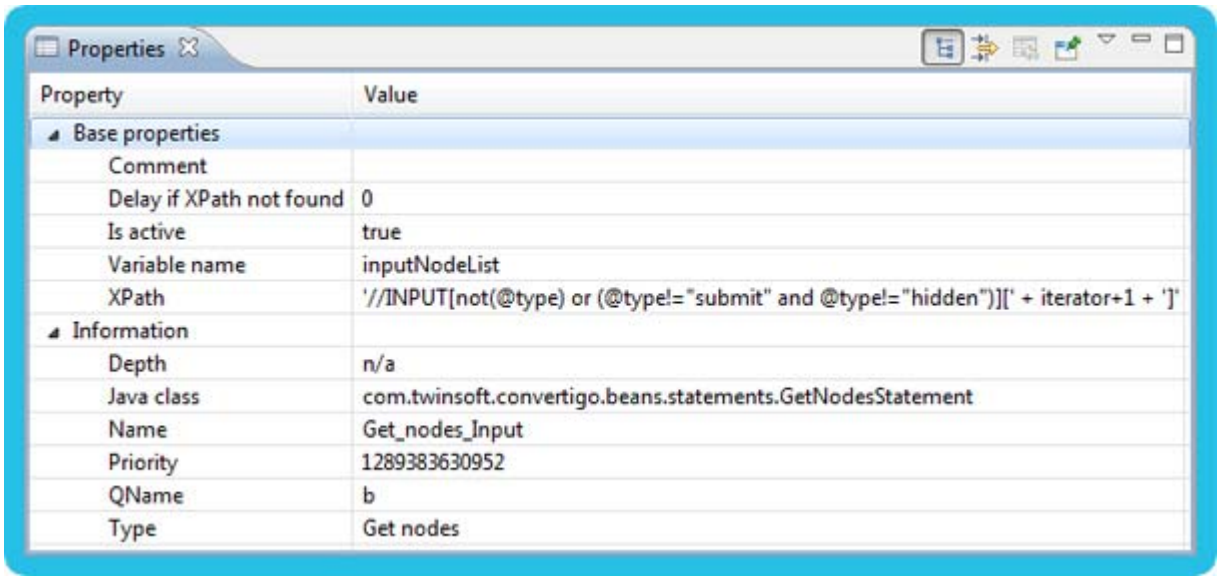


Figure 2 - 380: Get nodes statement - Configuration example

The **Xpath** property is set to a complex Javascript expression, mixing strings and variables in order to build an Xpath of the following form when evaluated for each iteration of the loop:

```
//INPUT[not(@type) or (@type!="submit" and @type!="hidden")][1]
```

This xpath will allow the statement to extract on each iteration a `NodeList` containing one node, or none when no nth `INPUT` exists.

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and various other statements used to implement the transaction behavior described above. It appears as follows in the **Projects** view:

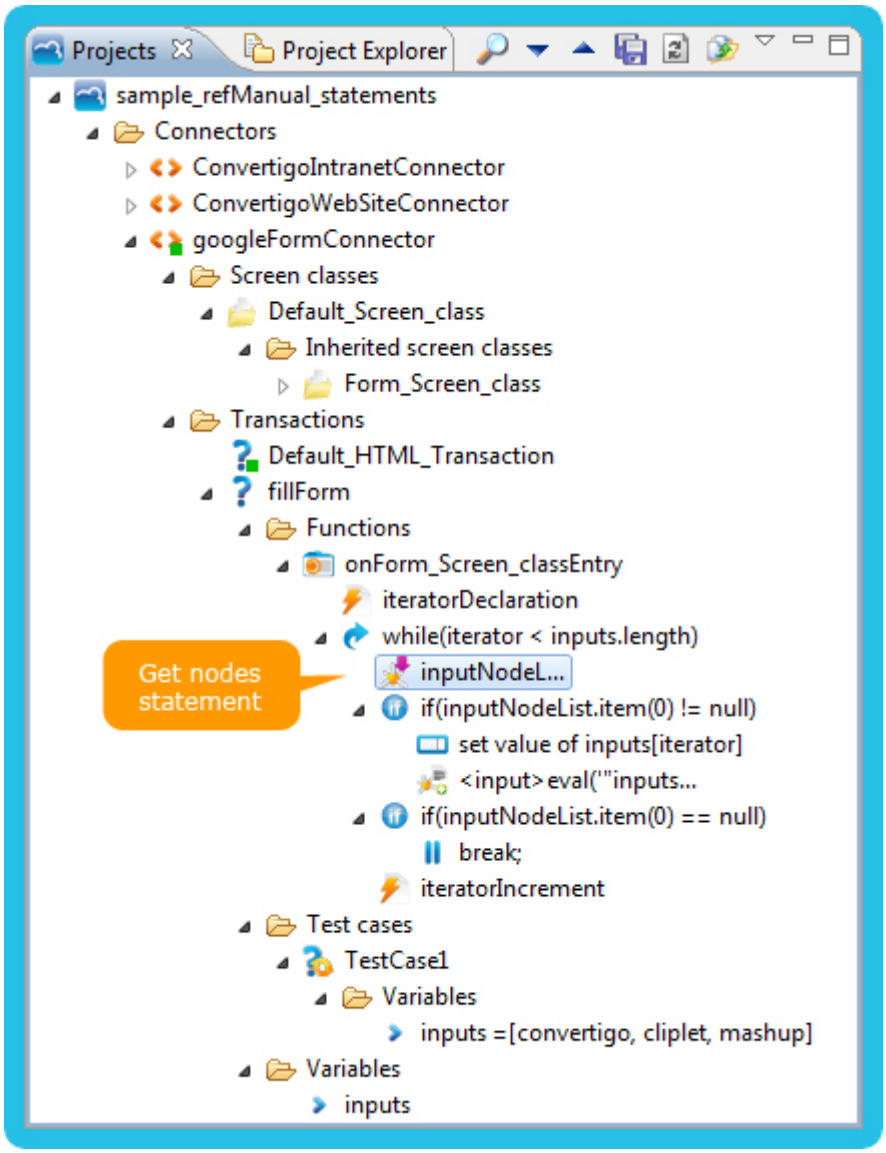


Figure 2 - 381: Get nodes statement - Object in Projects view

On each iteration, after the variable is created by the execution of this statement, the transaction tests if the variable contains a node or not in order to continue or stop, as previously explained.

This *If* statements use the following syntaxes to test if the extracted `NodeList` contains a `Node` or not :

- `inputNodeList.item(0) != null`: there is a `Node` in the `NodeList`,
- `inputNodeList.item(0) == null`: the `NodeList` is empty.

The *Test Case* defined for the transaction contains three values in `inputs` variable. When executing it on Google search page, containing a `FORM` element with only one `INPUT` to fill, the transaction sets the first value of `inputs` variable into the `INPUT` element of the web page and exists the loop as no second `INPUT` element is found for the next value.

GET TEXT 

OBJECT DESCRIPTION

Extracts a string from current HTML into a variable in JavaScript scope.

The *Get text* statement gets a single node from the web page DOM and sets a JavaScript variable in the current executed transaction JavaScript scope. This variable contains a `String`, extracted from the HTML page thanks to the execution of an XPath on the page DOM. This XPath is defined in **XPath** property.

The variable is named after the **Variable name** property value. It exists while the transaction is running.

If no node matches, the variable is `null`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delay if XPath not found	long	standard	Defines the maximum delay the statement waits if the XPath doesn't currently exists. When no node in the page DOM matches the XPath defined in **XPath** property, the statement waits for it to match up to this delay, set in milliseconds. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the statement continues its action. Note: It is equivalent to defining a statement *Wait synchronization* with an `XPath` synchronizer before this statement, waiting for the same XPath.
Is active	boolean	standard	Defines whether the statement is active.
Variable name	String	standard	Defines the name of the JavaScript variable. If this variable exists in scope, its value is overridden. If the variable doesn't exist in scope, it is created.
XPath	JS expression	standard	Defines the XPath expression of elements on which the statement applies. Depending on the statement, the execution of this **XPath** on the web page DOM can result in a single `Node` or a `NodeList`.

CONTEXT GET 

OBJECT DESCRIPTION

Gets the object bound with the specified key in the context.

This statement gets an object from the context identified by a key and sets it in a JavaScript variable.

This object was previously set in the context, bound with its key, thanks to *Context Set* statement.

If no object is bound with this key in the context, the JavaScript variable is set to `null`.

Note: The action is similar to a *Simple statement* with the following line of code:

```
myVar = context.get("key").
```

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
Key	String	standard	Defines the key identifying the object to retrieve. **Key** property is similar to the parameter of `context.get` JavaScript method.
Variable	String	standard	Defines the JavaScript variable into which the object will be retrieved. **Variable** property is similar to the name of the result variable of `context.get` JavaScript method.

EXAMPLES

Let's consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

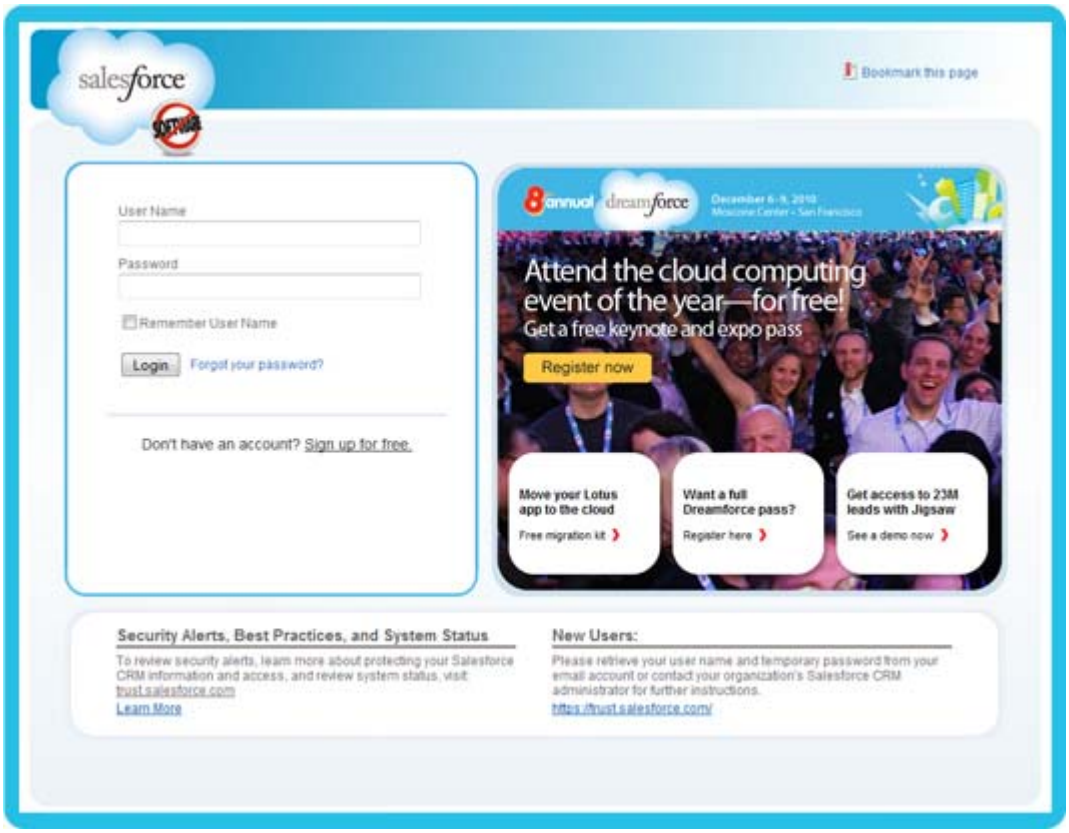


Figure 2 - 382: Context Get statement - SalesForce website authentication page

In the context of a mashup application including a SalesForce widget, we want the user to be able to customize its user / password. A `PreferencesRegistration` transaction, which defines two variables named `username` and `password`, allows to register in Convertigo context the user authentication values.

A `LoginFromPreferences` transaction, which does not define any variable, is written to authenticate the user on SalesForce website using the previously saved username and password. This transaction:

- retrieves user / password from the context,
- fills the User Name and Password fields using these values,
- submits the authentication form to access to SalesForce welcome page.

To do so, a *Context Get* statement is created in the *transaction start Handler* for each data to be retrieved from the context. For the `username`, the statement is created with the following parameters:

```
Context Get [
   key="user"
   variable=username
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

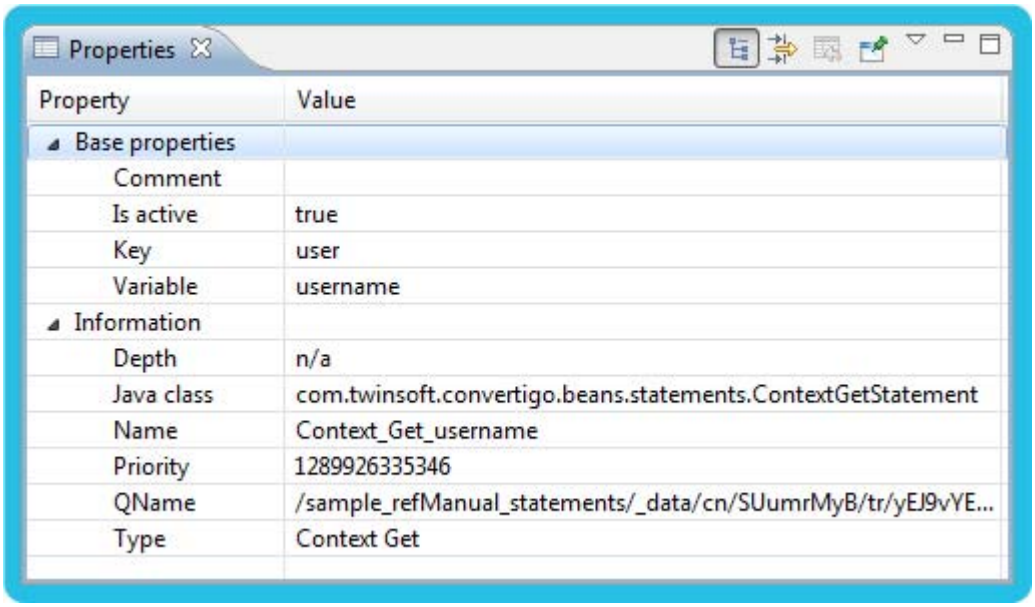


Figure 2 - 383: Context Get statement - Configuration example for username

For the `password`, the statement is created with the following parameters:

```
Context Get [
  key="pass"
  variable=password
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

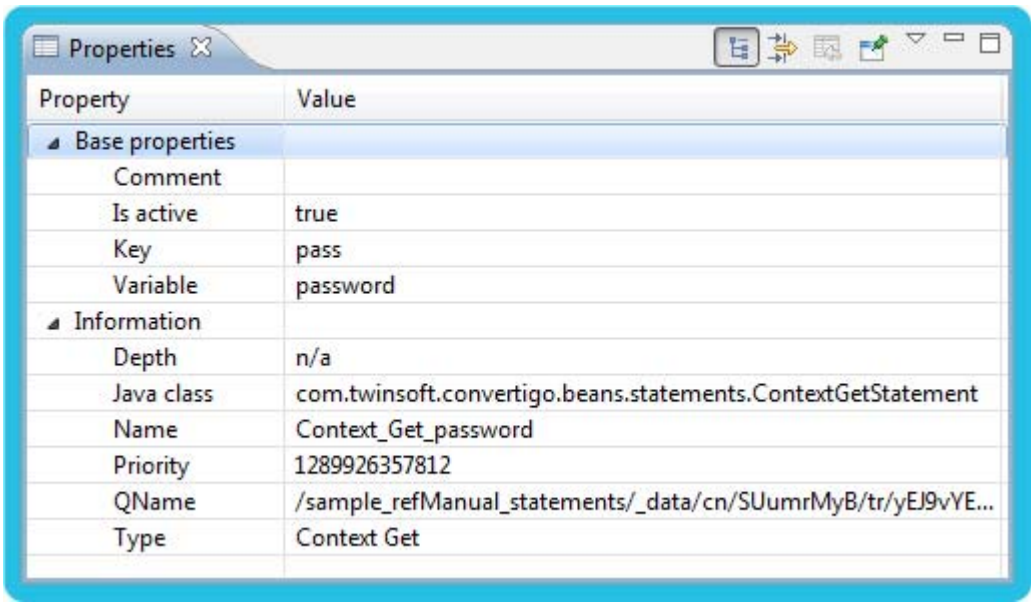


Figure 2 - 384: Context Get statement - Configuration example for password

The statements are created in the **Functions** folder of the transaction, under the *transaction start Handler* and appear as follows in the **Projects** view:

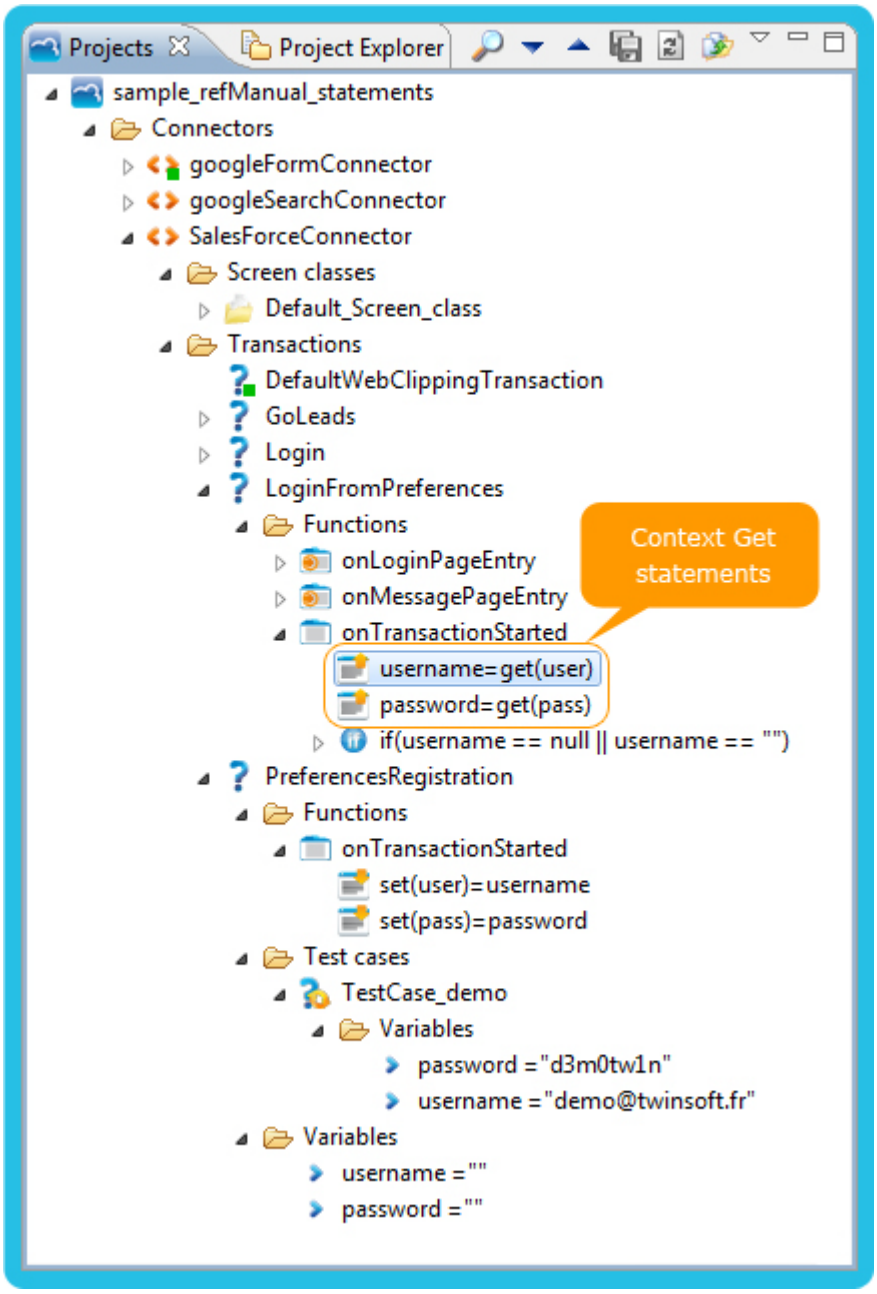


Figure 2 - 385: Context Get statement - Objects in Projects view

When executing the test case defined for the `PreferencesRegistration` transaction, user authentication values are saved in the context. Then the `LoginFromPreferences` transaction can be executed: it authenticates and connects to SalesForce website.

CONTEXT SET 

OBJECT DESCRIPTION

Stores an object identified by a key in the context.

Context Set statement stores an object in the context identified by a key.

This object can be retrieved later identified by its key thanks to *Context Get* statement.

Note: The action is similar to a *Simple statement* with the following line of code:

`context.set("key", object)`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the object to set. This property is a JavaScript expression that is evaluated during the transaction execution and gives the object to set. **Expression** property is similar to the second parameter of `context.set` JavaScript method.
Is active	boolean	standard	Defines whether the statement is active.
Key	String	standard	Defines the key to identify the object to be set. **Key** property is similar to the first parameter of `context.set` JavaScript method.

EXAMPLES

Let's consider the SalesForce website, famous CRM SaaS application. This website needs an authentification thanks to a user / password:

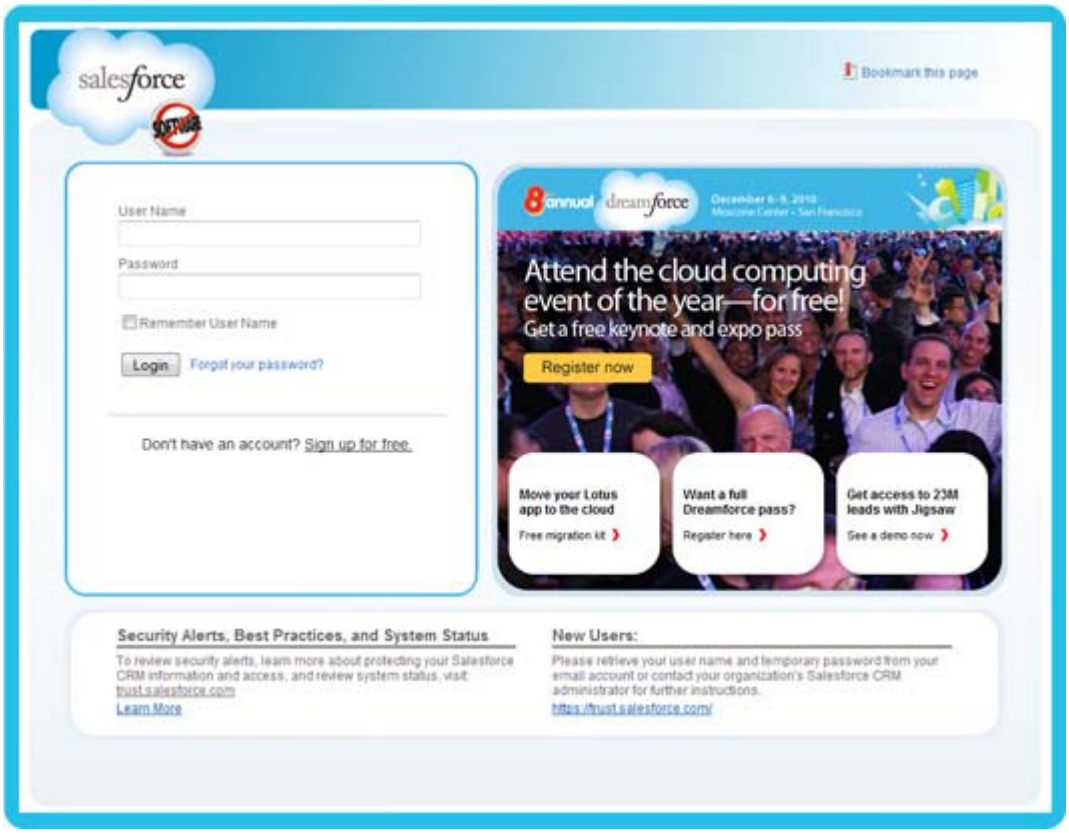


Figure 2 - 386: Context Set statement - SalesForce website authentication page

In the context of a mashup application including a SalesForce widget, we want the user to be able to customize its user / password. A `PreferencesRegistration` transaction, which defines two variables named `username` and `password`, is written in order to register in Convertigo context the user authentication values.

To do so, a *Context Set* statement is created in the transaction start Handler for each variable to save in the context. For the `username` variable, the statement is created with the following parameters:

```
Context Set [
  key="user"
  expression=username
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

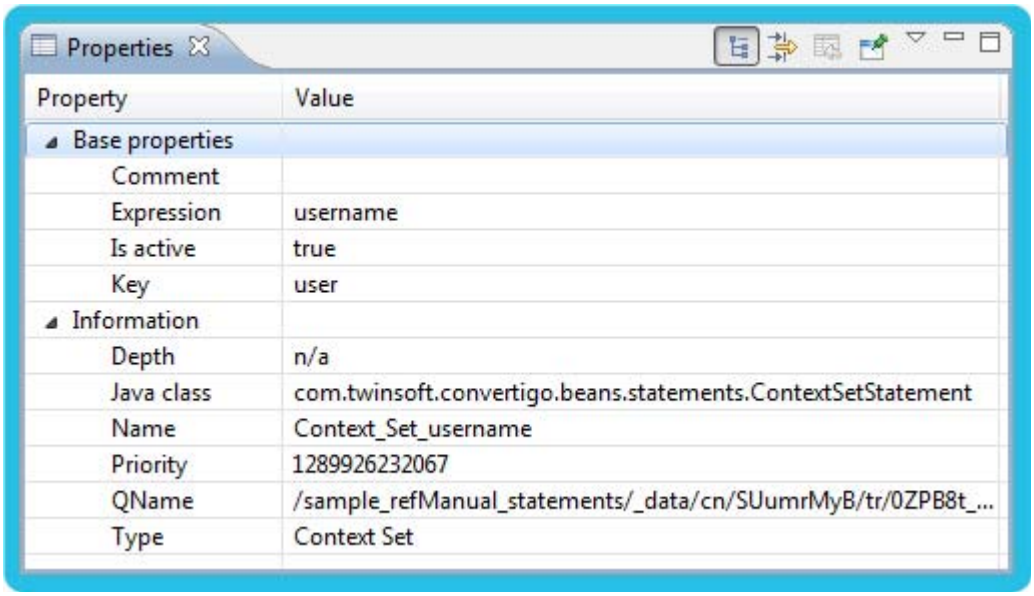


Figure 2 - 387: Context Set statement - Configuration example for username

For the `password` variable, the statement is created with the following parameters:

```
Context Set [
  key="pass"
  expression=password
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

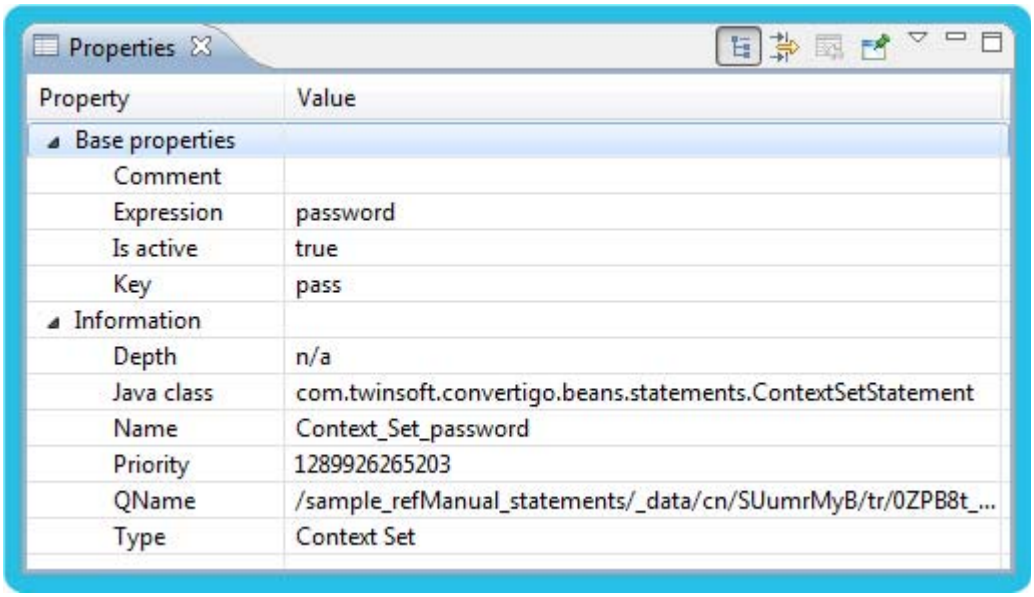


Figure 2 - 388: Context Set statement - Configuration example for password

The statements are created in the **Functions** folder of the transaction, under the *transaction start Handler* and appear as follows in the **Projects** view:

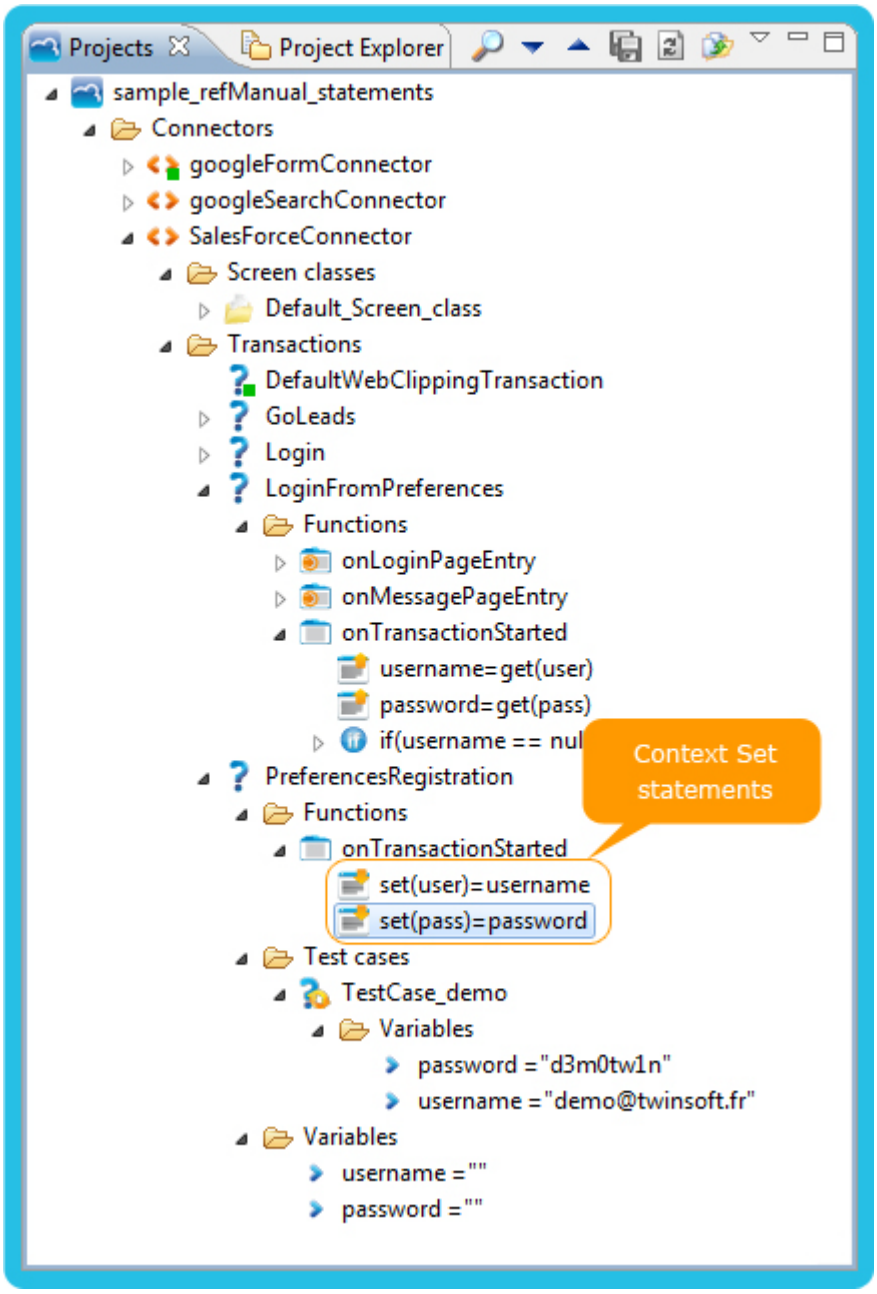


Figure 2 - 389: Context Set statement - Objects in Projects view

A `LoginFromPreferences` transaction, which does not define any variable, is written to authenticate the user on SalesForce website using the previously saved username and password. This transaction:

- retrieves user / password from the context,
- fills the User Name and Password fields using these values,
- submits the authentication form to access to SalesForce welcome page.

When executing the test case defined for the `PreferencesRegistration` transaction, user authentication values are saved in the context. Then the `LoginFromPreferences` transaction can be executed: it authenticates and connects to SalesForce website.

CONTEXT ADD TEXT NODE 

OBJECT DESCRIPTION

Adds a text node to transaction XML output.

This statement adds a node under the document root of the transaction XML output, containing a text.

Note: The action is similar to that of a *Simple statement* with:

`context.addTextNodeUnderRoot(expression)`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	standard	Defines the expression evaluated to give the text to output. This property is a JavaScript expression that is evaluated during the transaction execution and gives the text string to output in the generated element.
Is active	boolean	standard	Defines whether the statement is active.
Tag name	String	standard	Defines the tag name of the generated XML element. This property can contain any name, no words are reserved, and must follow the rules on XML naming: • it can contain letters, numbers, and other characters, • it cannot start with a number, • it cannot contain spaces nor punctuation character.

EXAMPLES

Let's consider a `fillForm` transaction, which defines one multivaluated variable named `inputs`. This transaction sets input field values into a web page containing a `FORM` HTML element.

As the transaction doesn't know in advance the number of inputs to fill in the `FORM`, it is implemented to loop on each element of the `inputs` variable, dynamically received from the caller, and to set each value in the nth HTML `INPUT` element of the web page. When there is no more `INPUT` element to fill, the transaction exits the loop and ends.

For each `INPUT` filled, we can add a text node in the transaction output XML to know at the end of the execution which values have been filled into `INPUT` elements. To do so, a *Context Add text node* statement is created in case the nth `INPUT` element is filled with the nth value

of the `inputs` variable. This statement is created with the following parameters:

```
Context Add text node statement [
   tag name="input"
   expression: "inputs["+iterator+"]='"+inputs[iterator]+"'"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

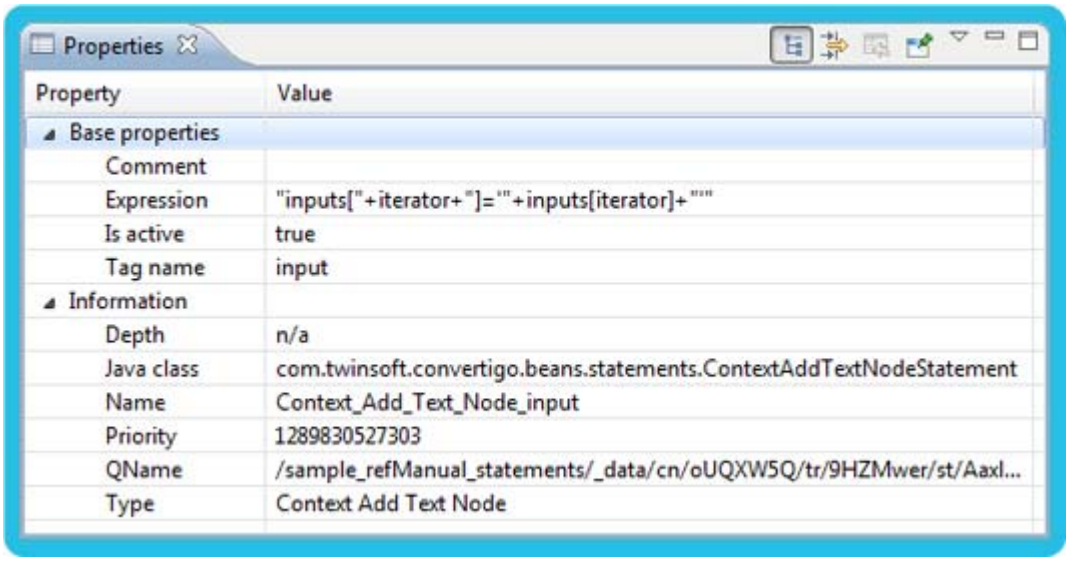


Figure 2 - 390: Context Add text node statement - Configuration example

The **Expression** property is set to a complex JavaScript expression, mixing strings and variables in order to build a string of the following form when evaluated:

```
inputs[2]='mashup'
```

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and various other statements used to implement the transaction behavior described above. It appears as follows in the **Projects** view:

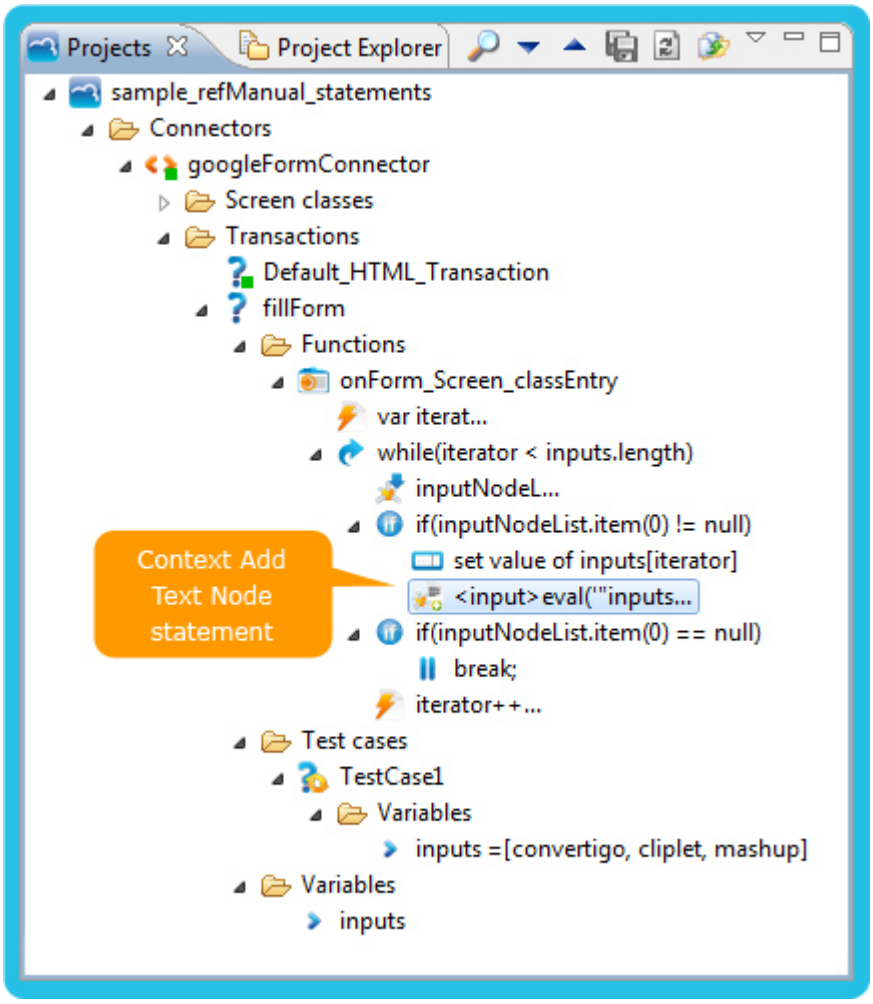


Figure 2 - 391: Context Add text node statement - Object in Projects view

The test case defined for the transaction contains three values in `inputs` variable. When executing it on Google search page, containing a `FORM` element with only one `INPUT` to fill, the transaction sets the first value of `inputs` variable into the `INPUT` element of the web page and exists the loop as no `INPUT` element is found for the next value.

After execution, the transaction XML output contains the one text node that has been added thanks to the statement:

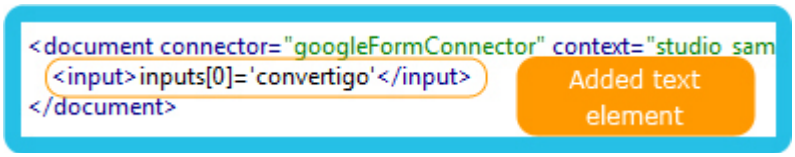


Figure 2 - 392: Context Add text node statement - Resulting XML after executing fillForm transaction on Google search page

LOG (WEB) 

OBJECT DESCRIPTION

Produces output data in log file.

This statement outputs a message in context or engine logger (defined thanks to the **Engine** property), for the log level defined in the **Level** property.

The message to output is generated from the JavaScript expression defined in **Expression** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Engine	boolean	standard	Defines if the log is to be output in `Engine` logger or default `Context` logger. This property allows to choose the `logger` on which the log applies. If set to `true`, the message will be seen as output by the Convertigo Engine. If set to `false` (default value), the message will be seen as output by the running Context.
Expression	JS expression	standard	Defines the expression evaluated to give the text to output. This property is a JavaScript expression that is evaluated during the transaction execution and gives the text string to output in log file.
Is active	boolean	standard	Defines whether the statement is active.
Level	String	standard	Defines the `log level` on which the log applies. This property defines the minimum level of log for which the message has to be output. The message will be output for any log level superior or equals to this property value. Log levels possible values are the following, by ascending order: • `ERROR`, • `WARN`, • `INFO`, • `DEBUG`, • `TRACE`.

EXAMPLES

Let's consider a transaction similar to `GoToGoogleCom` transaction set in the context of the "*Starting With Convertigo Web Clipper*" *Quick Guide*, but defined in a Web Integrator project.

This transaction connects to Google website and, depending on the client country, navigates to Google.com page. When finally arrived on Google.com page, the transaction retrieves the cookies sent by the website and, for each cookie retrieved, displays a log line with its content.

To output cookies value in log, a *Log* statement is created with the following parameters:

```
Log [
  expression: "### Google cookie ["+i+"]: "+googleCookies[i]+" ###"
  engine=false
  level=INFO
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

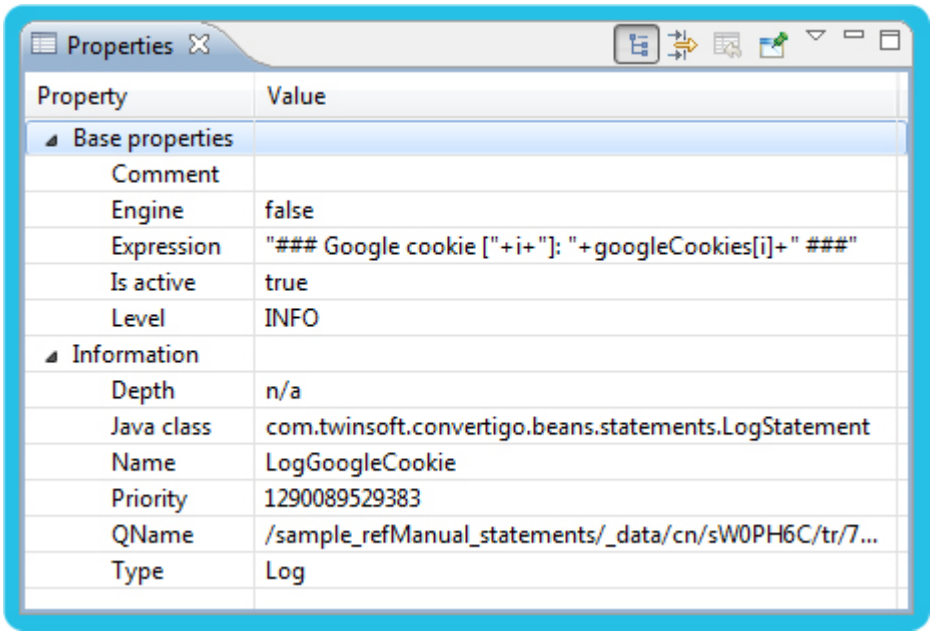


Figure 2 - 393: Log statement - Configuration example

The **Expression** property is set to a complex Javascript expression, mixing text strings and data from the variable in which the cookies have been retrieved, in order to build a dynamic message when evaluated during the transaction execution.

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler and appear as follows in the **Projects** view:

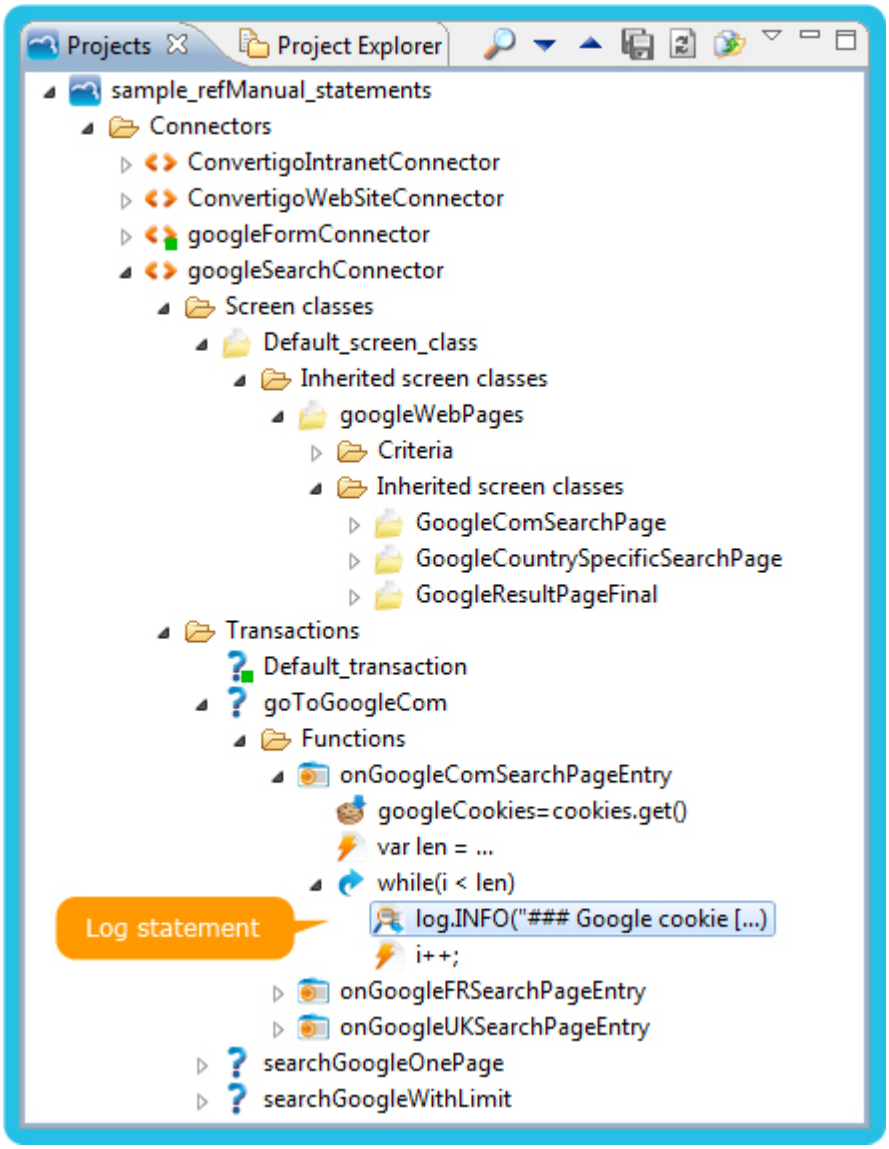


Figure 2 - 394: Log statement - Object in Projects view

When executing the `GoToGoogleCom` transaction, cookies are retrieved from the website and the Convertigo engine log displays the following lines:

Figure 2 - 395: Log statement - Log displaying retrieved cookies

WAIT SYNCHRONIZATION 

OBJECT DESCRIPTION

Checks the defined synchronization and waits for it.

A synchronization is defined thanks to the **Synchronization** property. This statement does no action but waits for the synchronizer to return.

There are four types of synchronizers, see their definitions in **Synchronization** property description.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description

Synchronization	TriggerXMLizer	standard	Defines how to synchronize the statement.
			A synchronizer states how and when accessed pages are considered fully loaded. Only then are data extracted and new pages re-detected.
			There are several types of synchronizers, that are described hereafter:
			• `Document completed`: The synchronizer waits for a number of documents to be completed. Specify here how many "document completed" events Convertigo has to wait for before assuming that the page is complete. In many cases, when the target application uses HTTP META redirects or JavaScript redirects, the document is loaded several times. You can monitor
			`==== start parse ======`
			and
			`==== Parse end ==(XXXms )==========`
			traces in the Engine console (debug mode) to count the number of "document completed" events needed for the synchronizer. The `Document completed` synchronizer can be configured to also stop on alert messages that could pop up. Alert messages do not trigger a "document completed" event and are not detected by this synchronizer. To activate this option, check the **Stop on alert** checkbox.
			• `XPath`: The synchronizer waits until a specified XPath is found. Convertigo tries to evaluate the specified XPath while receiving a web page or executing JavaScript in it. Once the XPath matches at least one node of the page, the synchronizer returns.
			• `Wait time`: The synchronizer waits until a specified time is reached (in ms, set via the **Timeout** property).
			• `Screen Class`: The synchronizer waits for one of the selected screen classes to be detected. Several screen classes can be selected to be waited for. The synchronizer returns when one of them is reached.
			• `Download started`: The synchronizer waits for a download request. This is the perfect synchronizer to use before a **Get attachment** statement.
			• `No wait`: The synchronizer doesn't wait and execution proceeds directly.
			For all synchronizer types, the maximum waiting time is set using the **Timeout** property.

Continue with Site Clipper 

Object Description

Redirects to a *Site Clipper connector*.

Continue with Site Clipper statement ends an *HTML transaction* by redirecting the context to a *Site Clipper connector*. It is set at the end of an handler to end the *HTML transaction*, which results in that no other handler nor statement from the transaction is executed after it.

This statement specifies a redirection URL to its parent connector in the transaction's XML output. This rewritten URL is an absolute URL pointing to the current Convertigo project, with a particular syntax:

- it starts with the usual project's path,
- it then specifies the Convertigo context and the *Site Clipper connector* to use,
- it ends with the `.siteclipper` extension,
- after the extension, the target resource URL is concatenated, replacing the '`://`' symbols after the target resource protocol, (`http://` for example, by a '`/`' character.

This gives the following URL form:

```
http://<convertigo_server_host>:<convertigo_server_port>/convertigo/
projects/<project_name>/
context=<context_name>&connector=<connector_name>.siteclipper/
<target_resource_protocol>/<target_resource_host>/
<target_resource_URI>
```

The *Site Clipper connector* accessed thanks to this URL then relays all HTTP messages between the client and the target server. For more information about Site Clipper, see *Site Clipper connector* and related objects documentation.

Note: The *Site Clipper connector* to which redirect is in the same project. Thus, the *HTML transaction* and the *Site Clipper connector* can share the same context.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.

Property	Type	Category	Description
Site Clipper connector	String	standard	Define the *Site Clipper connector* to which redirect. The target connector can be chosen among the connectors from the same project as the *Continue with Site Clipper* statement. Indeed, the HTML transaction including the *Continue with Site Clipper* statement and the *Site Clipper* connector must share the same context.

EXAMPLES

Let's consider the Convertigo administration website. This website needs an authentication with a user / password:

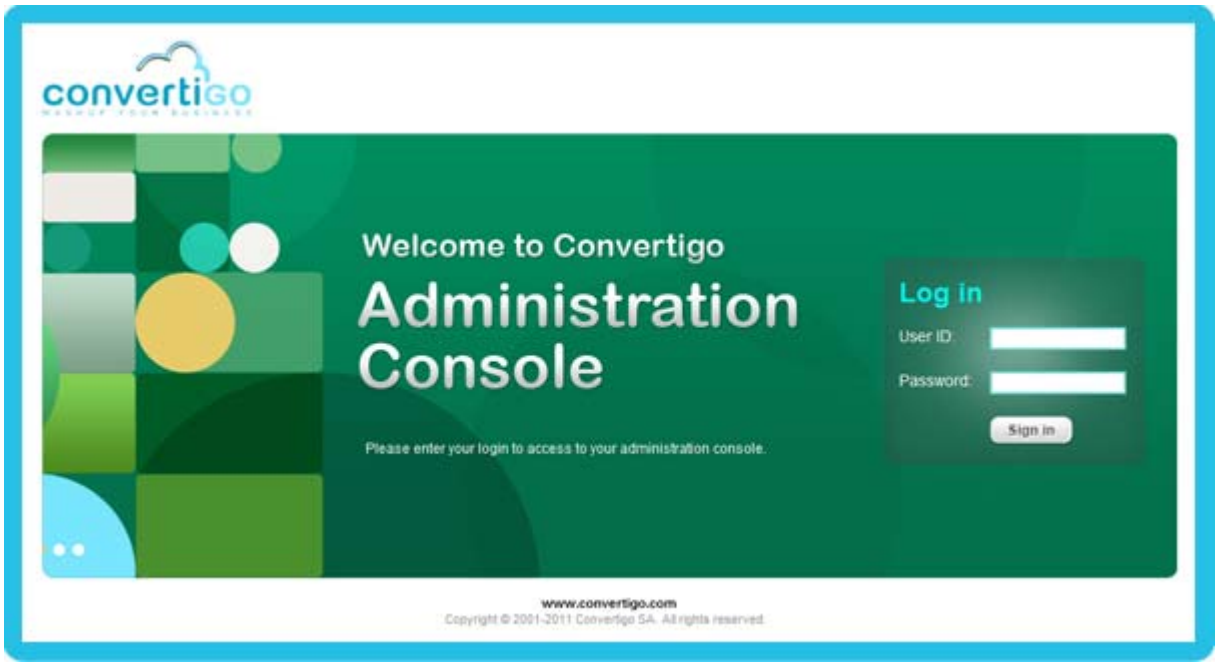


Figure 2 - 396: Continue with Site Clipper statement - Login page of Convertigo administration website

In the context of this website clipping, we want the login phase to be automated and then the user to be able to get the browsing control back on the administration website through Convertigo.

An HTML transaction, named `LoginAdmin`, is created to automate the access to the Configuration page of Convertigo administration website. Defining two variables named `username` and `password`, this transaction:

- connects to Convertigo administration website,
- enter the username and password in the corresponding fields,
- validates the authentication by clicking on the "Sign in" button,
- clicks on the "Configuration" menu button to access Configuration page.

After the transaction accesses the Configuration page of Convertigo administration website, we want the user to get back the control of the browsing, through Convertigo.

To do so, a *Continue with Site Clipper* statement is created at the end of the transaction for Convertigo to switch to the Site Clipper connector that is defined in the same project. This

statement is set with the following parameters:

```
Continue with Site Clipper [
   site clipper connector=ConvertigoAdminConnector
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

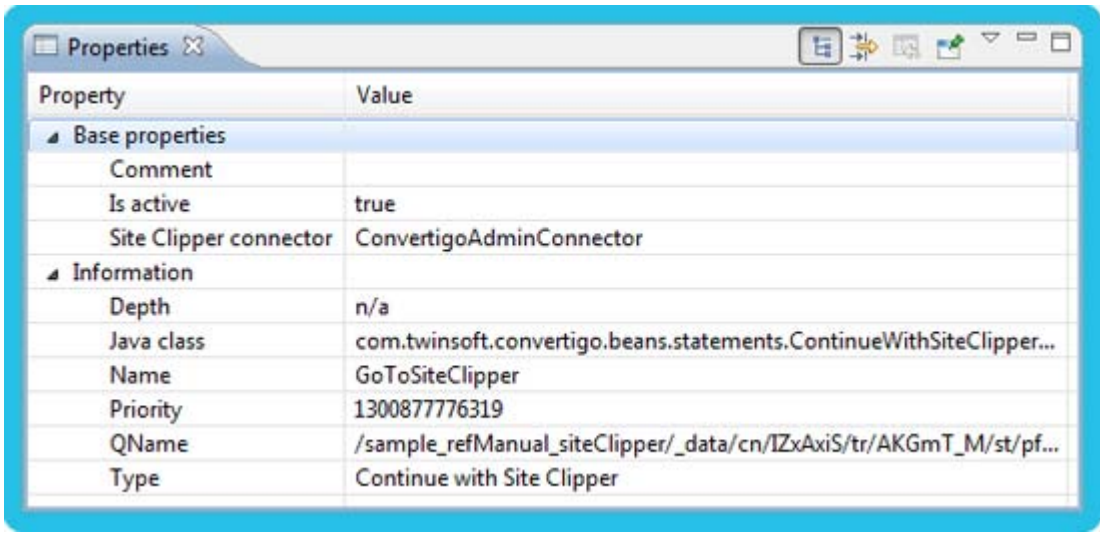


Figure 2 - 397: Continue with Site Clipper statement - Configuration example

The statement is created in the **Functions** folder of the transaction, under the corresponding Screen class handler, next to other statements that implement the transaction behavior described above. It appears as follows in the **Projects** view:

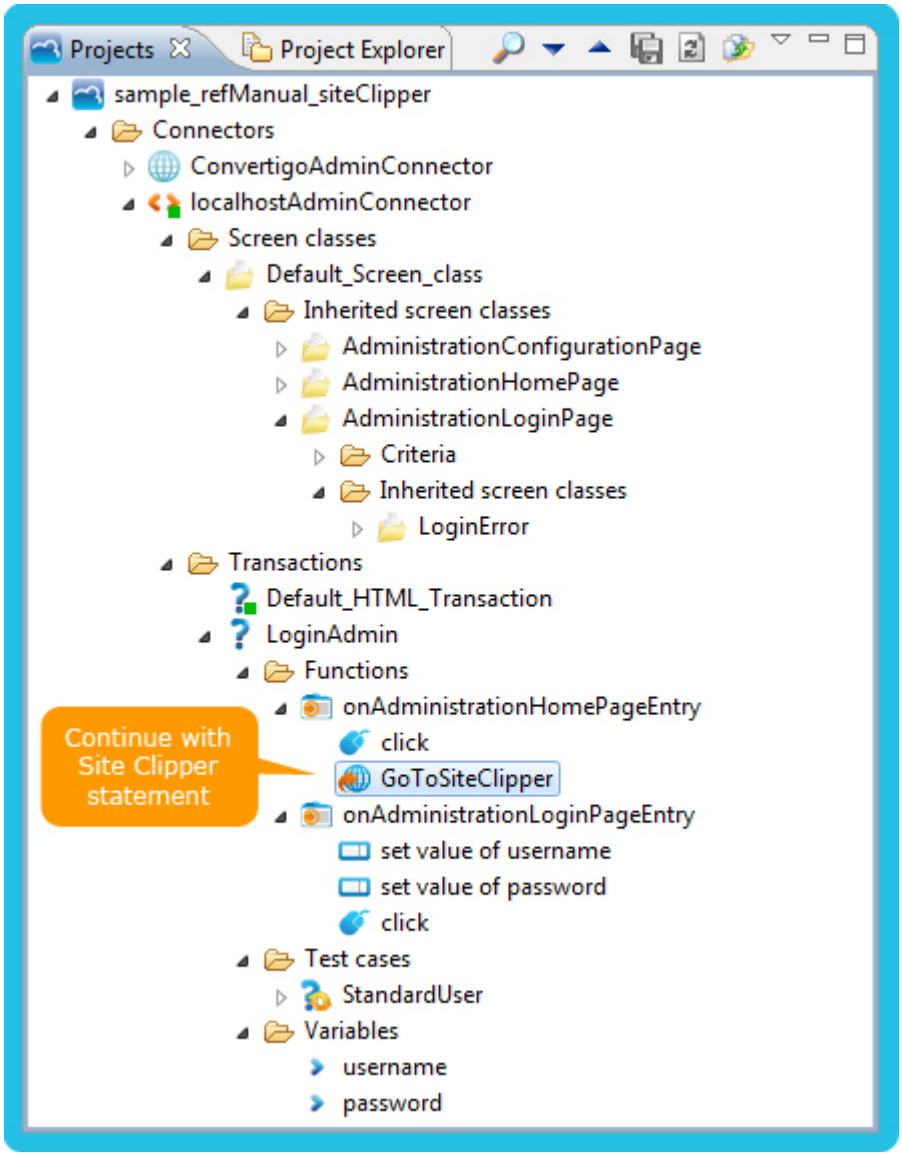


Figure 2 - 398: Continue with Site Clipper statement - Object in Projects view

When executing the test case defined for the `LoginAdmin` transaction in the test platform, authentication values are automatically set by Convertigo in corresponding fields and the transaction accesses Configuration page of Convertigo administration website. Then, the transaction automatically continues on the *Site Clipper connector*. The user accesses the Configuration page of Convertigo administration and can navigate on the website through Convertigo:

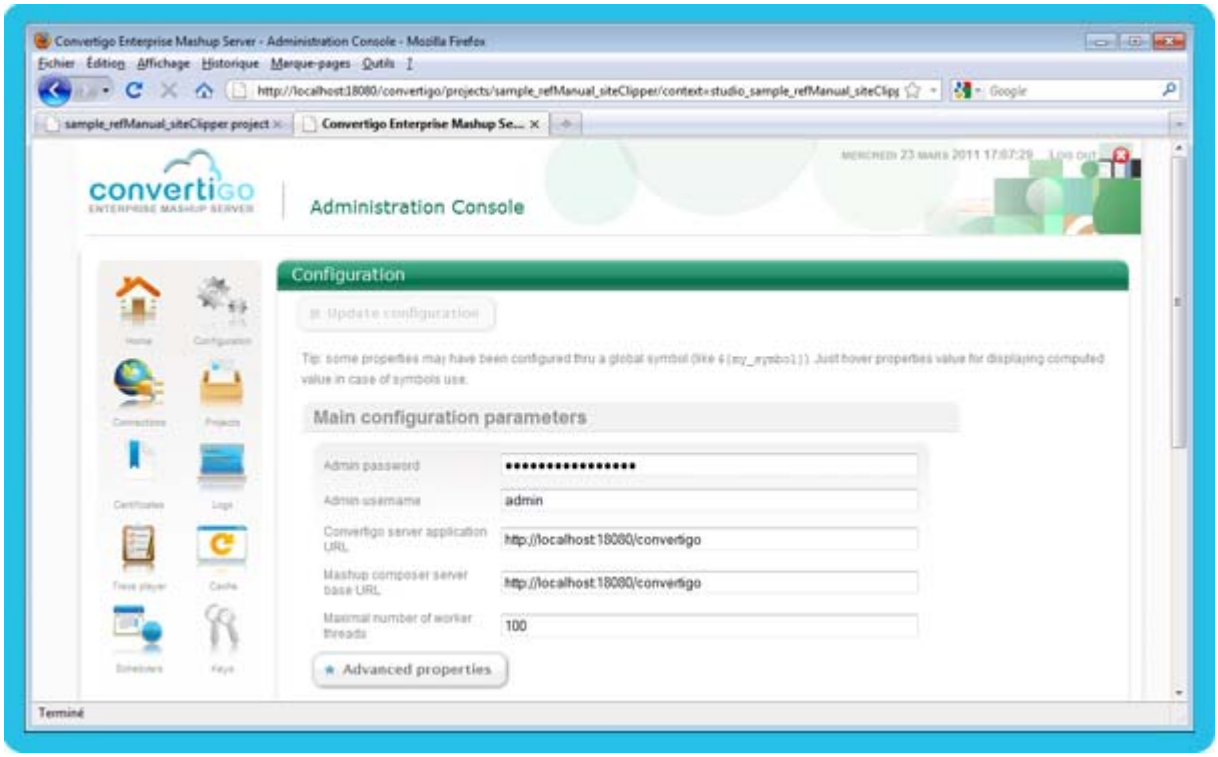


Figure 2 - 399: Continue with Site Clipper statement - Return of transaction execution

Recorder for Site Clipper 

Object Description

Starts recording HTML connector HTTP responses to reuse them with *Continue with Site Clipper* statement.

Recorder for Site Clipper statement enables the recording of all HTTP responses that match the *URL RegEx filter* made by the *HTML connector* and disable cache for those responses.

The recording stop when a *Continue with Site Clipper* is executed or if a stateless *HTML transaction* starts. Those recorded HTTP responses are reuse when the *Site Clipper connector* response the same resources and prevent to ask the resources again to the remote website.

This can be important on some page where a browser refresh doesn't provide the same content, like a POST form result or query that modify server side data (such a shopping cart).

Record response consume memory, so use it with caution and try to set a *URL RegEx filter* if possible. A recorded response is destroyed after the *Site Clipper connector* use it or when its *response lifetime (sec)* is expired.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	standard	Defines whether the statement is active.
Response lifetime	int	standard	Define the time-to-live (in seconds) of recorded responses. Recorded responses are kept in memory a maximum of time corresponding to the **Response lifetime** property value. If a recorded response is not used by a *Site Clipper connector* during its lifetime, it is automatically destroyed when its time-to-live expires.
URL regexp filter	String	standard	Defines a regular expression for response URL filtering. All HTTP responses should not be recorded by this statement. The **URL regexp filter** property allows defining a regular expression as a string pattern to find in the URL of HTTP responses. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

Let's consider the website "www.loancalculator.ws" that proposes to calculate the amount of each period and interest payment of loans. It uses an HTML form where loan data are entered:

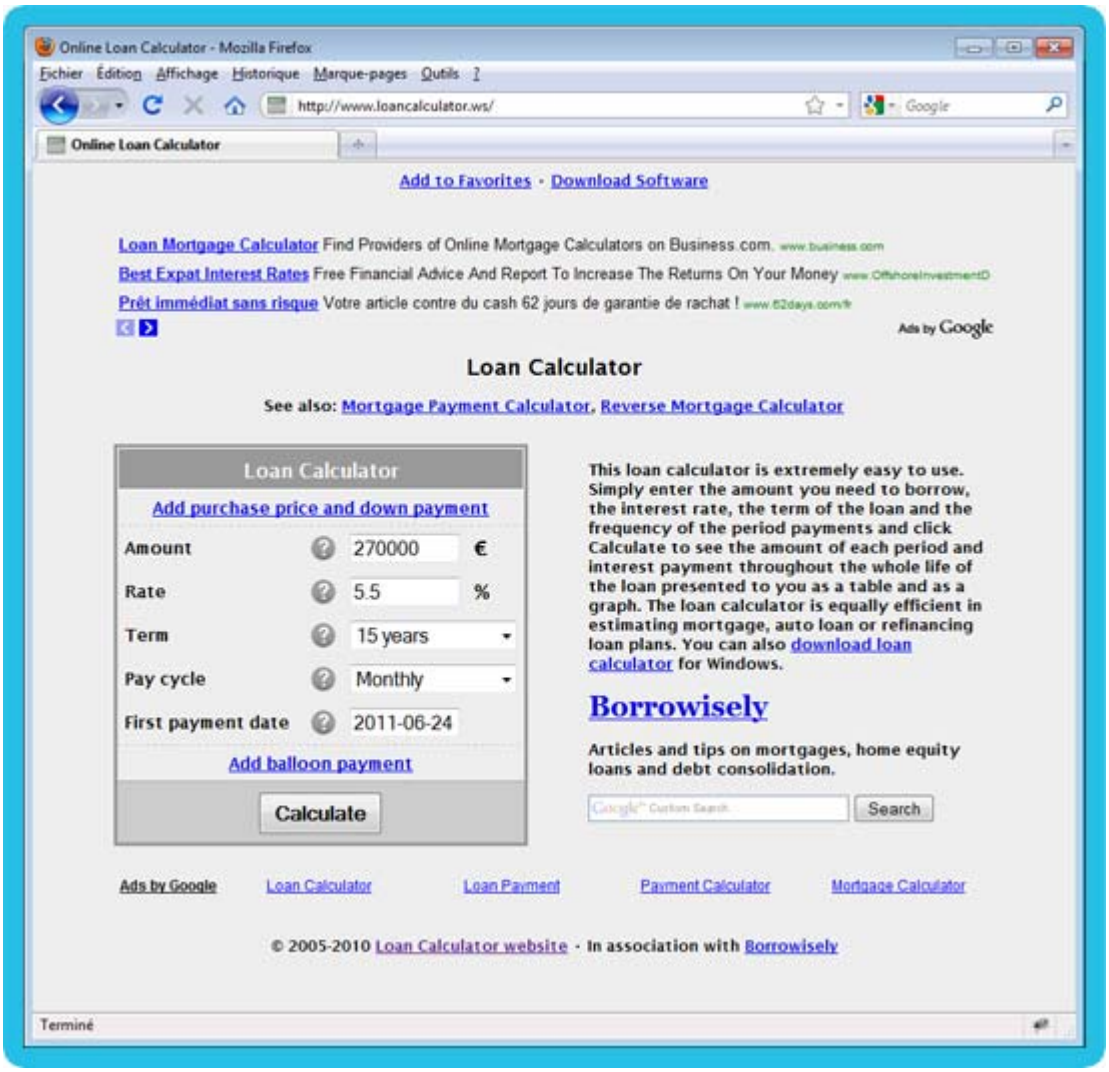


Figure 2 - 400: Recorder for Site Clipper statement - Form page of LoanCalculator website

When validated, this form submits its fields as POST data to the response page, which URL is "`http://www.loancalculator.ws/free/`", and displays a result page of the following form:

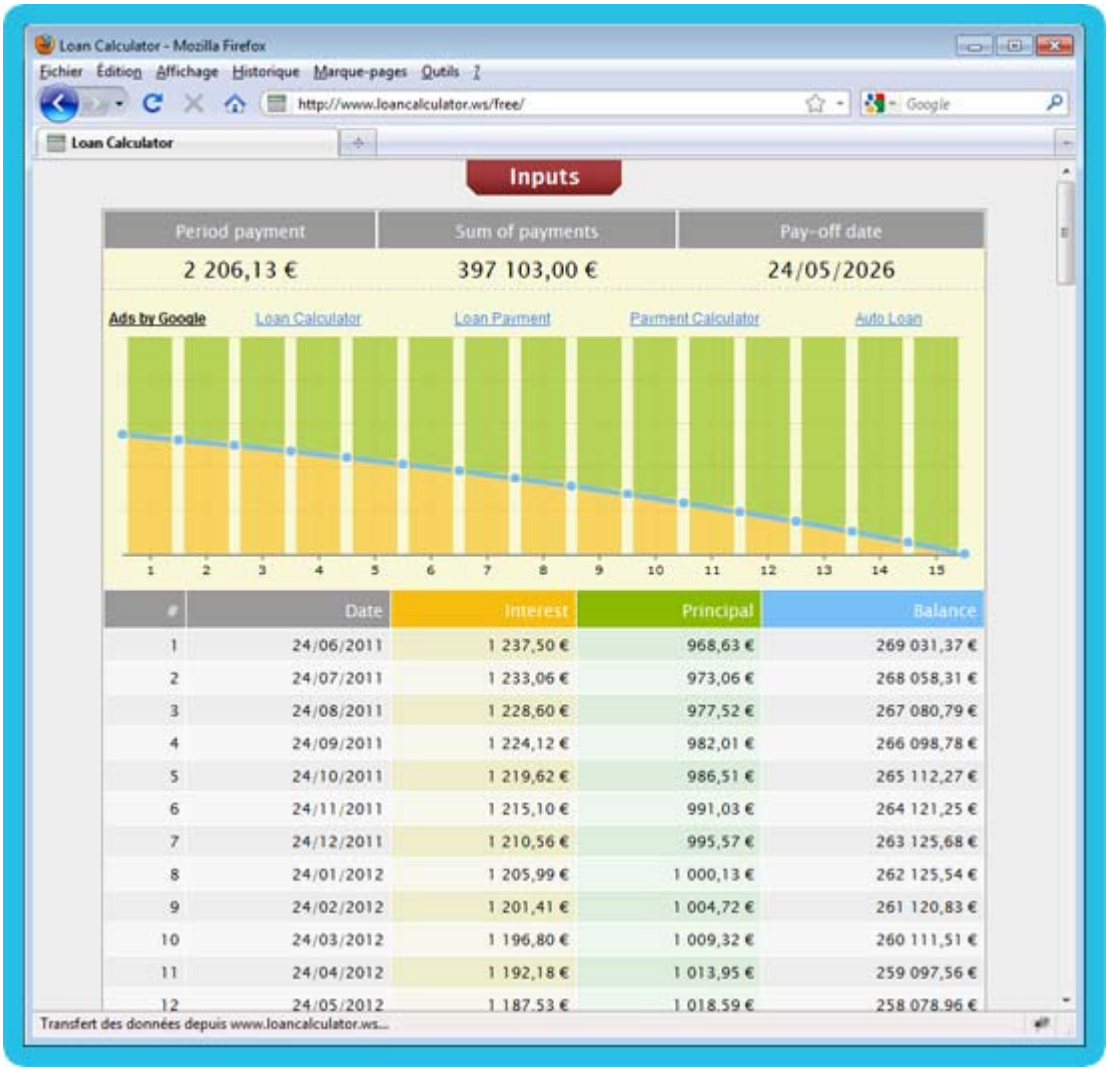


Figure 2 - 401: Recorder for Site Clipper statement - Result page of LoanCalculator website

An HTML transaction, named `LoanSimulation`, is developed to perform a loan simulation with data retreived from its variables. It fills all fields with input data and submits the form.

When the result page is displayed, the transaction's purpose is to give the control back to the user through a *Site Clipper connector* for the user to see the results of its loan calculation.

To do so, the transaction uses a *Continue with Site Clipper* statement on the results screen class. For more information about this statement, see "*Continue with Site Clipper*" statement documentation and examples.

Switch to a browser displaying the test platform for this project. The transaction declares default values for its variables, so it can be executed directly.

Executing the `LoanSimulation` transaction (in a new tab thanks to the **Execute Full Screen** button) sets correctly all data in the form and submits it.

Then, when the result page is displayed, the context is passed to the *Site Clipper connector* and the user gets the control back on the website accessed through Convertigo.

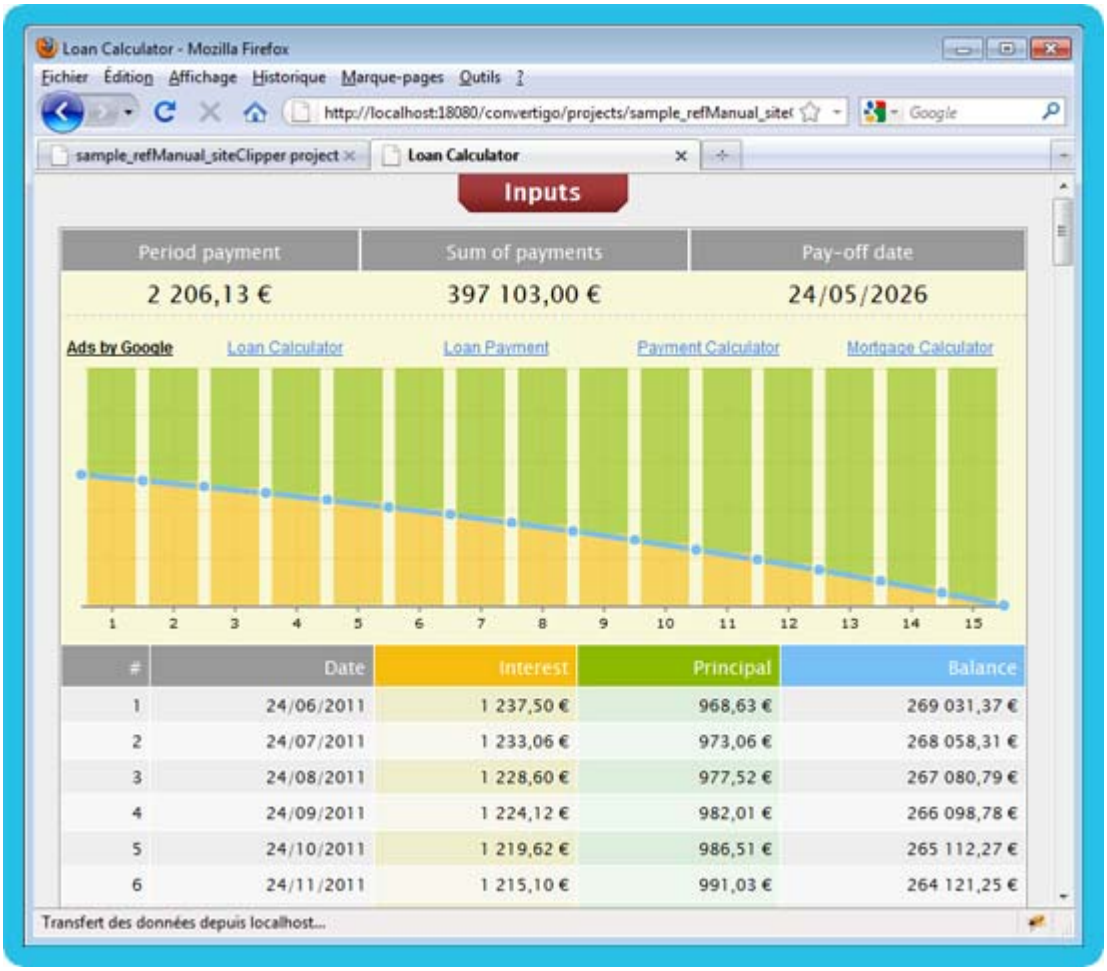


Figure 2 - 402: Recorder for Site Clipper statement - Result page through Site Clipper after the transaction execution

We can see that the result page the user gets back doesn't display the same data as the web browser included in the HTML connector editor of Convertigo Studio.

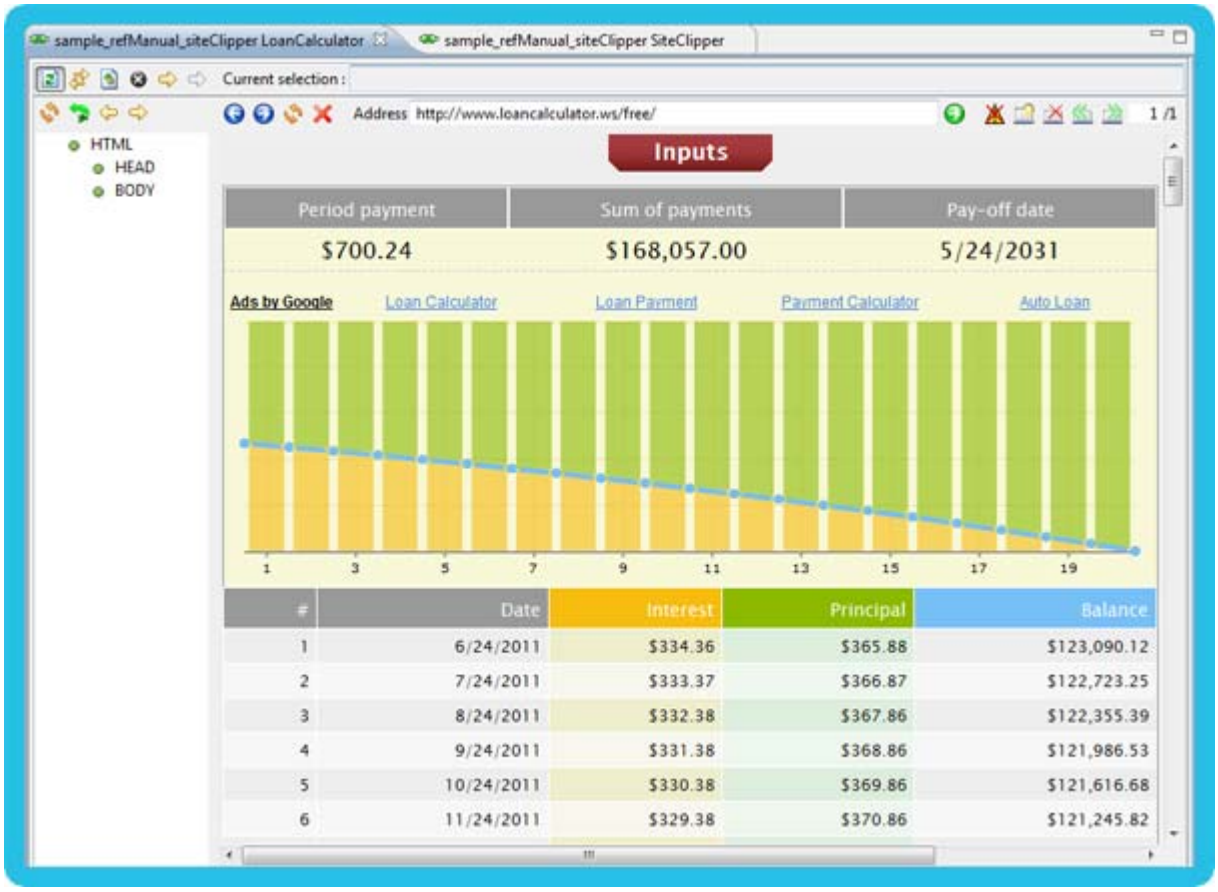


Figure 2 - 403: Recorder for Site Clipper statement - Result page in connector editor after transaction execution

Actually, the Convertigo Studio web browser displays the results corresponding to data entered in the form by the *HTML transaction* (values from transaction input variables).

When passing the context to the *Site Clipper connector*, the request asks for "`http://www.loancalculator.ws/free/`" URL, which is the URL of the result page, but without sending again the `POST` data sent by the form.

In other words, when passed to the *Site Clipper connector*, the result page is refreshed and displays the website's default results.

In order to prevent this behavior, the *HTML transaction* can use a a *Recorder for Site Clipper* statement before submitting the form. This statement is set with the following parameters:

```
Recorder for Site Clipper [
  URL regexp filter=.*
  response lifetime=120
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

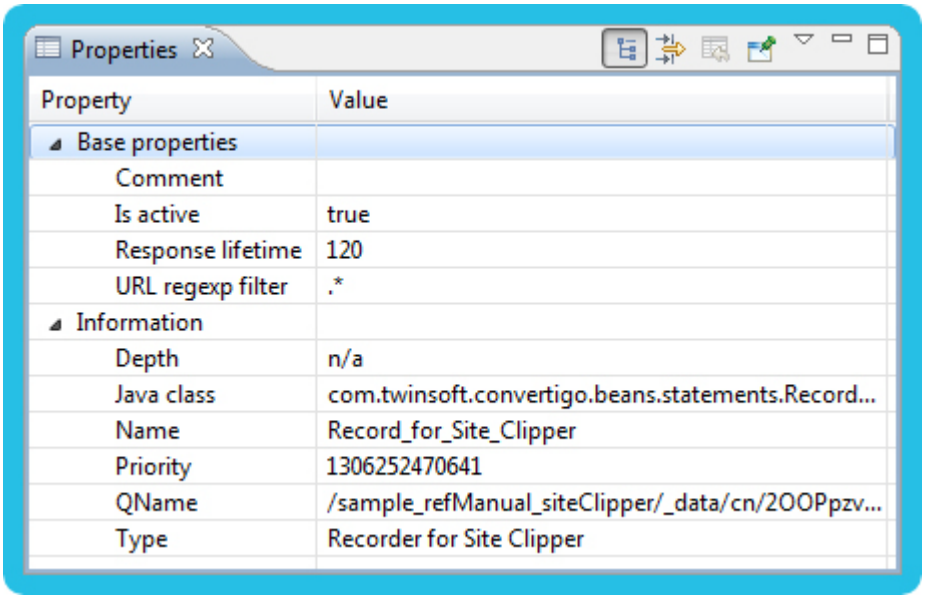


Figure 2 - 404: Recorder for Site Clipper statement - Configuration example

The statement is created in the **Functions** folder of the transaction, in the form page screen class handler, before the *Mouse action* statement that performs the click validating the form. It appears as follows in the **Projects** view:

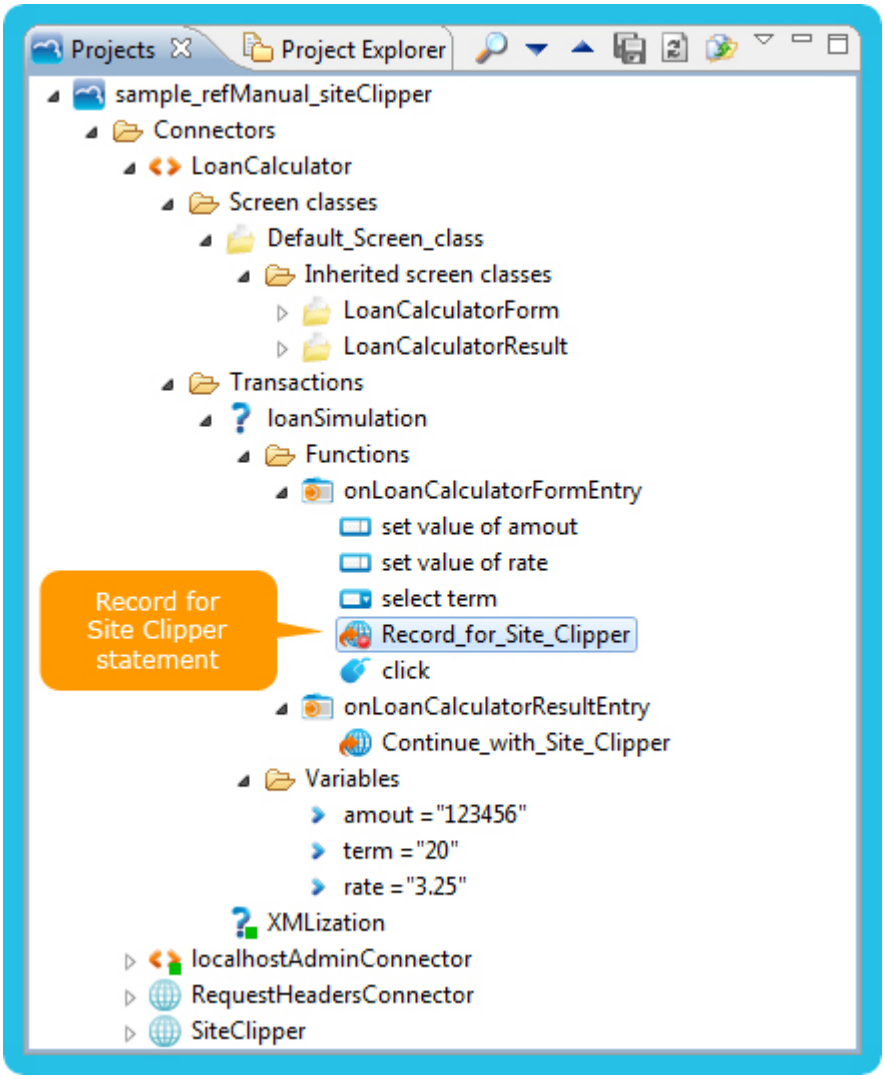


Figure 2 - 405: Recorder for Site Clipper statement - Object in Projects view

Switching back to the browser displaying the test platform for this project. Executing again the `LoanSimulation` transaction (in a new tab thanks to the **Execute Full Screen** button) sets correctly all data in the form, starts recording for Site Clipper and submits the form. Then, when the result page is displayed, the context is passed to the *Site Clipper connector* and the user gets the control back on the correct result page from the website accessed through Convertigo:

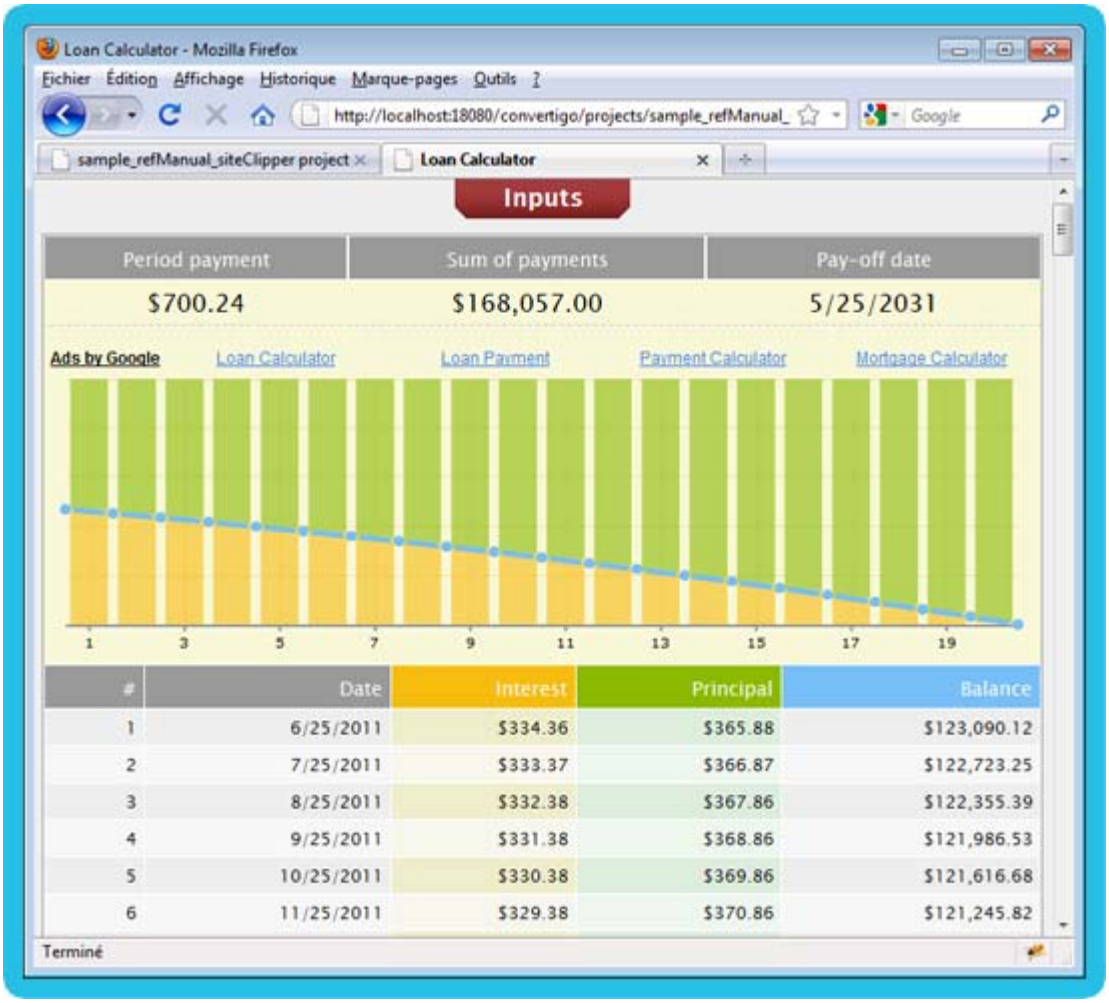


Figure 2 - 406: Recorder for Site Clipper statement - Return of transaction execution

WHAT HAPPENEND?

Convertigo recorded the HTTP response received by the *HTML connector* when submitting the form, as the *Recorder for Site Clipper* statement was set before the click that submits the form.

Then, when the *Site Clipper connector* asked for the result page URL, Convertigo retrieved the HTTP response recorded from *HTML connector* and the *Site Clipper connector* got the same result as the *HTML connector*.

2.9 Legacy

2.9.1 Main objects

JAVELIN CONNECTOR 

OBJECT DESCRIPTION

Establishes connections with a legacy screen application (`IBM 3270`, `IBM 5250`, `Bull DKU 7107` or `Videotex`).

A *Javelin connector*, also named *Legacy connector*, represents a connection to a legacy system, based on a terminal emulation session. It allows Convertigo to connect to a mainframe application to perform transactions, that is to say navigate through legacy screens and either:

- extract data into a proper XML document (CLI),
- on-the-fly webize legacy screens (CLP).

Javelin connector is needed by Convertigo to connect to legacy applications. Once connected, all tasks (screen classes detection, data extraction, browsing, etc.) associated with the *Javelin connector* can be carried out as defined in the project thanks to several objects:

- Screen classes,
- Criteria,
- Extraction rules,
- Javelin transactions,
- Screen classes handlers.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Connection address	String	standard	Defines the connection address. The connection address of a *Javelin connector* is composed of: • a **destination address**, as a hostname (or IP adress) and optionally a port, • a **connection type**: most often `DIR` for direct connection, it can also take `EIC` or `TCP` as value, • a **connection parameter**, optional. The **connection parameter** has different meanings according to the emulator: • 3270: TN3270 device name, • 5250: TN5250 device name, • DKU: MAILBOX, • Minitel: service code (e.g. '3615SNCF'). It can be defined using an automatic numbering syntax managed by Convertigo engine: `PREFIX<POOL:x-y/z>SUFFIX`. This syntax will automatically generate a pool of "auto-numbered connection parameters", the *Javelin connector* will use one of them when a new connection starts. This syntax is composed of the following elements: • `PREFIX`: any prefix string to start the device name or service code, • `<POOL:x-y/z>`: incremental number from `x` to `y` on `z` digits (for example `<POOL:1-99/2>` meaning an incremental number from 1 to 99 on 2 digits, i.e. from `01` to `99`), • `SUFFIX`: any suffix string to end the device name or service code. Notes: • When a connection using an "auto-numbered connection parameter" is closed, the parameter is released in the pool and can be used again. • This pool of "auto-numbered connection parameters" works like a round robin: when released, an "auto-numbered connection parameter" is queued at the end of the pool. It will be re-used only when all others have been used before. • The pool of "auto-numbered connection parameters" will not work in Convertigo Studio context, only works in Convertigo server.
Connection synchronization code	String	expert	Defines the code to execute for synchronization purposes after connecting the emulator to the host. This property allows the developer to program a code to be executed to synchronize the emulator after its connection, before executing any transaction. It uses JavaScript code as *Javelin transaction* core.
Emulator	String	standard	Defines the emulator associated with the connector. This property takes one of the following values: • `Bull DKU 7107`, • `IBM 3270`, • `IBM 5250 (AS/400)`, • `Videotex (Minitel)`, • `Unix (VT220)`.

Property	Type	Category	Description
Enable SSL	boolean	expert	Defines whether a SSL connection should be used.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.
IBM terminal type	String	expert	Defines the IBM terminal type. This property allows to override the value of the **TerminalType** configuration property. Depending on the **Emulator** property value, this overridden configuration property is present in different files, and this **IBM terminal type** property can take different values. For `IBM 3270` emulator: • the **TerminalType** configuration property is defined in "TerminalSNA.txt" configuration file, • its default value is positioned to "IBM-3278", • it can be overridden by **IBM terminal type** property to `IBM-3278` (corresponding to old 3270), • or it can be overridden by **IBM terminal type** property to `IBM-3279` (default value for 3270). For `IBM 5250` emulator: • the **TerminalType** configuration property is defined in "TerminalAS400.txt" configuration file, • its default value is positioned to "IBM-5250", • it can be overridden by **IBM terminal type** property to `IBM-3179` (default value for 5250).
Language	int	expert	Defines the language used within the emulator. This property value has to be chosen amongst a list of available values.
Trust all SSL server certificates	boolean	expert	Defines if all server certificates should be automatically trusted for SSL connections.
Virtual server	String	expert	Defines the name of the virtual server to use (if left empty, the primary virtual server is used).

EXAMPLES

The following is an example of *Javelin connector* set for connecting to a local IBM 5250 legacy application:

```
Javelin connector [
  connection address=[connection parameter="", host name="localhost",
  port=23, connection type=DIR]
  emulator=IBM 5250 (AS/400)
  connection synchronization code="javelin.waitForDataStable
  (timeout, threshold);"
  carioca authentication=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

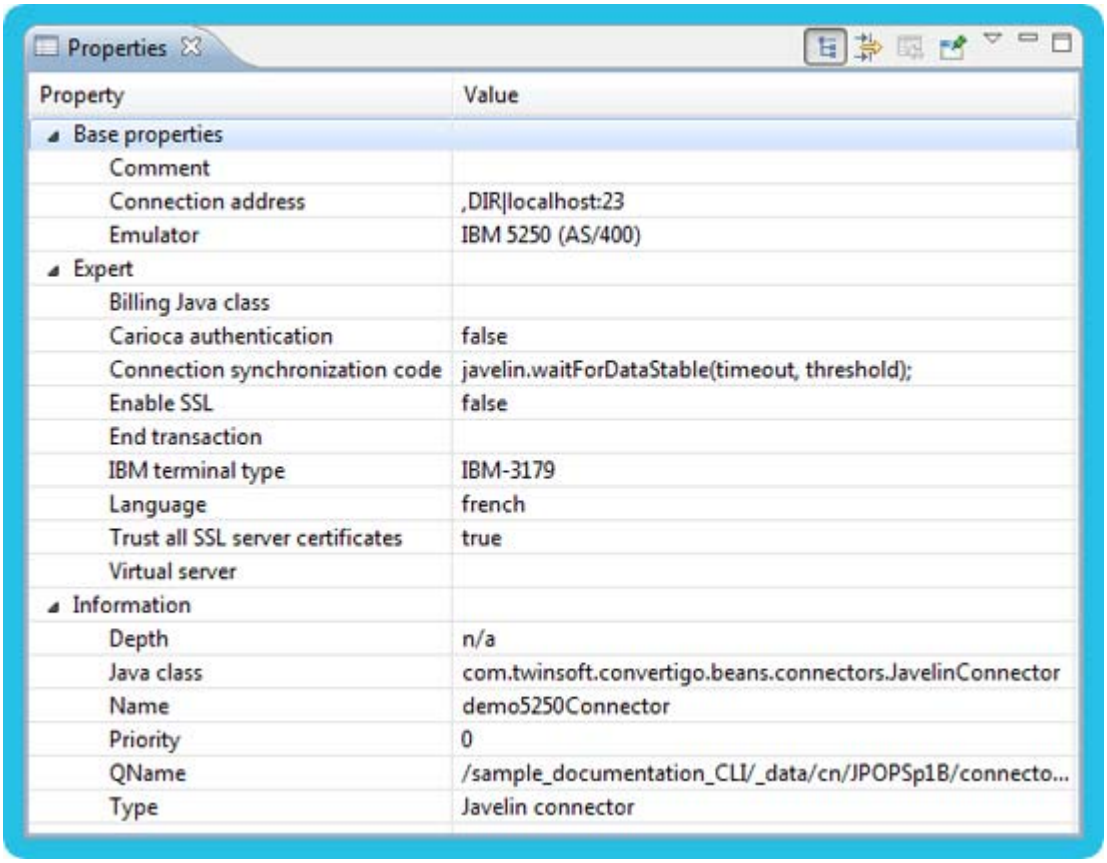


Figure 2 - 407: Javelin connector - Configuration example

Connection address property is edited in the associated editor:

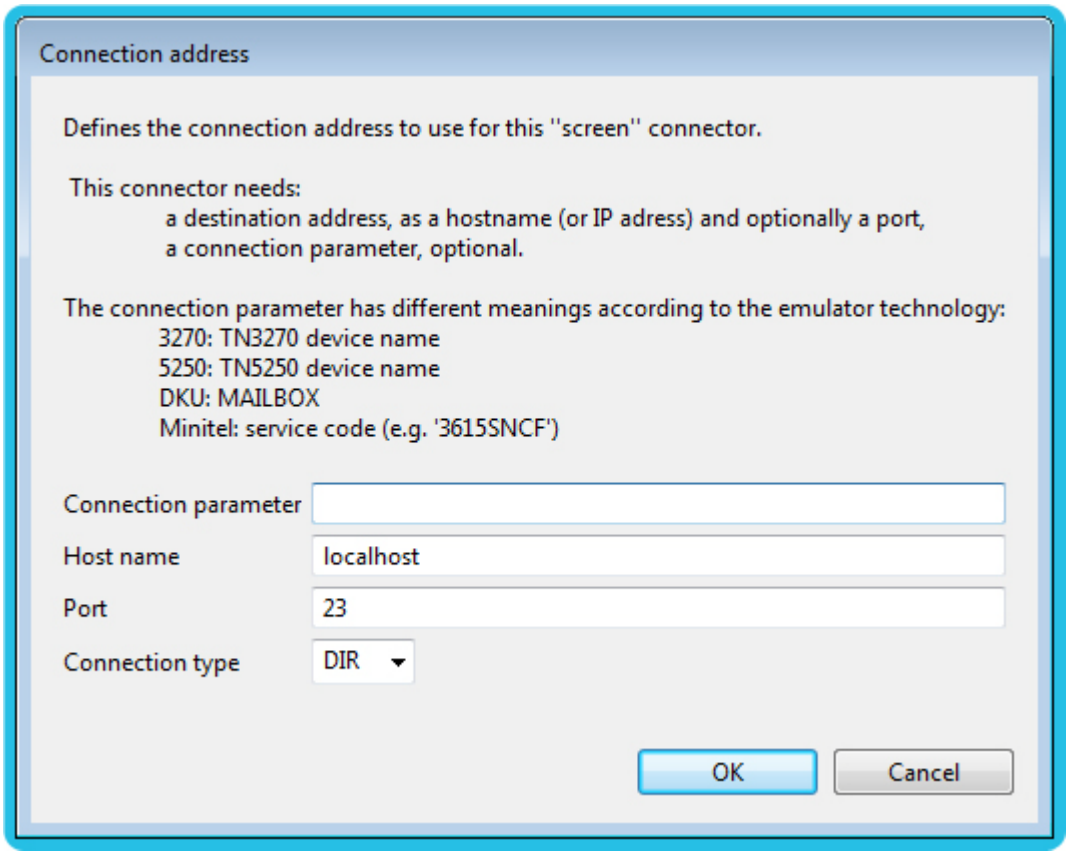


Figure 2 - 408: Javelin connector - Connection address property edition

In the Convertigo Studio, the **Javelin connector** editor displaying the emulator connected to the legacy application and the generated XML is displayed as follows:

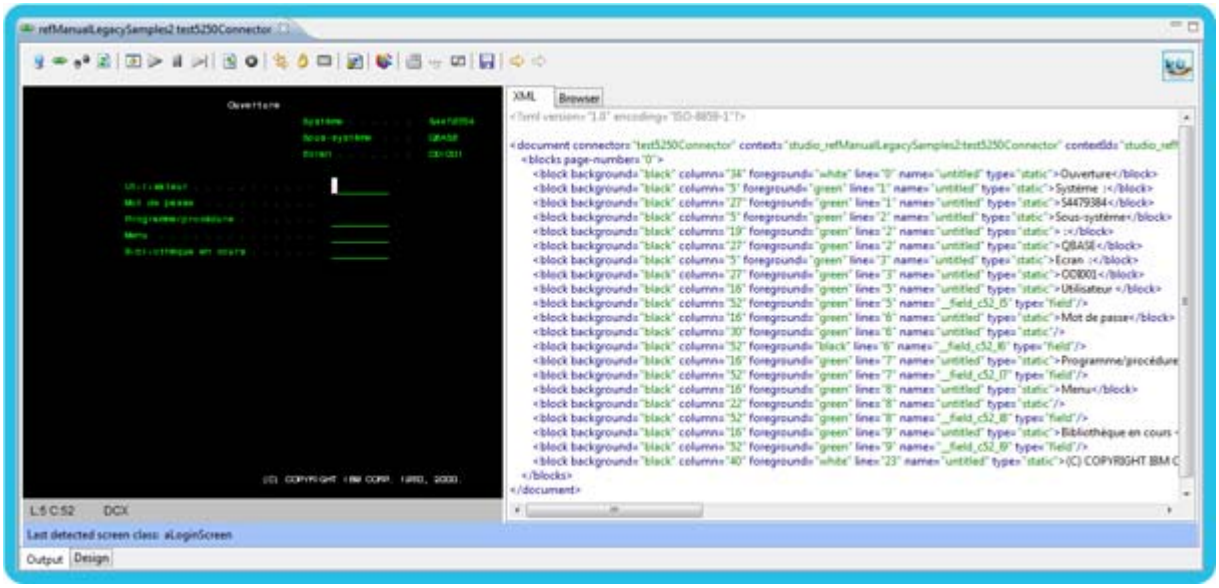


Figure 2 - 409: Javelin connector - Connector editor in Studio

JAVELIN TRANSACTION 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

JAVELIN SCREEN CLASS 

OBJECT DESCRIPTION

Defines a group of screens with common features, in a Legacy connector.

By the term "screen" is meant a set of identifiable data which may be rendered to the user or not. It is generally used regardless of the resource accessed by Convertigo (web page, Legacy screen, HTTP stream, etc.).

Thus, in the case of *Legacy connector* (also called *Javelin connector*) projects (Legacy Integrator and Legacy Publisher), a screen may be defined by the data sent back by a Legacy host, for a green screen display.

A *Javelin screen class* is identified by a set of criteria which are dedicated to screen's data detection. When accessing a screen (i.e. a legacy screen) thanks to a *Javelin connector*, Convertigo looks for detection criteria defined for screen classes in current connector.

Convertigo considers that the accessed screen belongs to the *Javelin screen class* which all criteria match and which have the greatest number of criteria matching. For screen classes that would have the same number of matching criteria, Convertigo considers that the screen belongs to the screen class that has the greatest depth. And if screen classes also have the same depth, Convertigo considers that the screen belongs to the first screen class in alphabetical order.

For Legacy Integrator and Legacy Publisher projects (screens in a *Javelin connector*), detection criteria are *Empty screen*, *Emulator technology*, *Find string* and *Regular expression*. You can see these objects definitions and properties for more information.

A *Javelin screen class* can also be associated with extraction rules executed on its detection by Convertigo. Extraction rules define which data are to be extracted from a screen and turned into a proper XML document.

Javelin screen classes are pivotal in the execution of transactions, since their detection triggers the execution of screen class handlers (including actions to be performed on detected screens) and extraction rules (extracting data to be turned into XML).

Note: A *Javelin screen class* do not define one screen only, but all screens matching the specified criteria. It is up to the Convertigo programmer to set detection criteria.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

EXAMPLES

Example 1

Let's consider the following legacy screen:

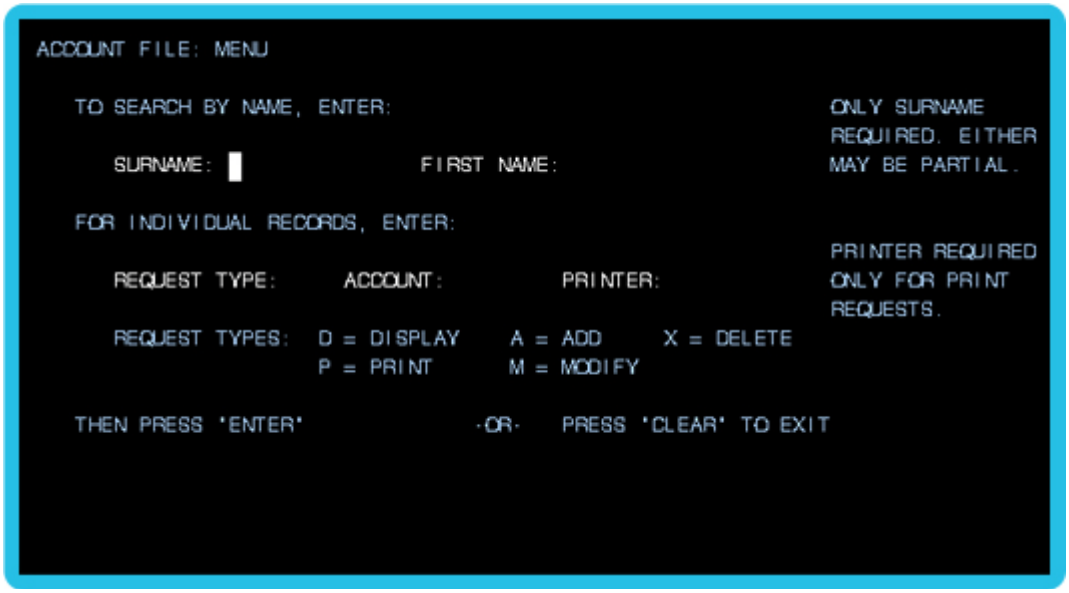


Figure 2 - 410: Javelin Screen class - Legacy screen

In this example, we want to define a *Screen class* matching this legacy screen. A *Screen class* object has no properties to configure, it is defined by its criteria.

```
Screen class [
]
```

The *Screen class* object is created in the **Screen classes** folder of the connector, inherited from the *DefaultScreenClass* screen class. It is created together with its first criterion and appears as follows in the **Projects** view:

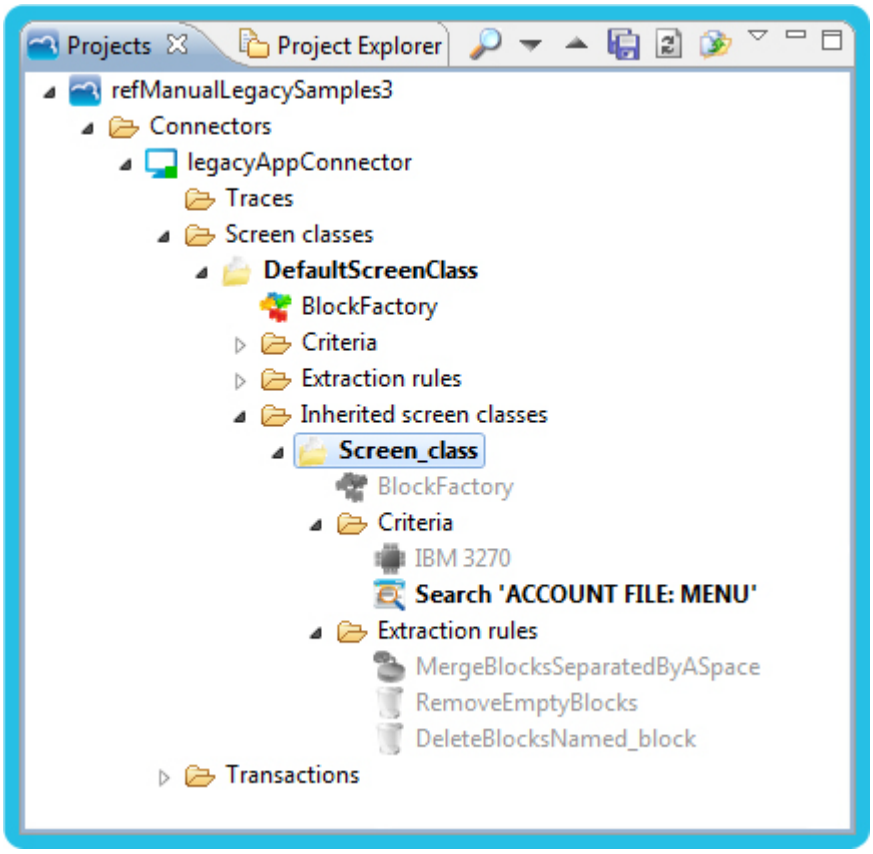


Figure 2 - 411: Javelin Screen class - Screen class and first criterion in Projects view

Thanks to its criterion defining the remarquable characteristics of the page, this *Screen class* matches the previous legacy screen. For more information about this criterion, see "*Find string*" criterion documentation and examples.

Example 2

The `sample_documentation_CLI` project set in the context of the "*Starting with Convertigo Legacy Integrator*" Quick Guide contains a number of *Javelin* screen classes defined to meet the project requirements.

The defined *Javelin* screen classes are named either in accordance with their function (`LoginScreen`, `ArticlesManagementScreen`) or with their screen identifier (`MNS10`, `MMS01B1`, `MMS001E`).

For example, the login screen including the "Entrez votre nom" (*Enter your name*) string is detected as belonging to the `LoginScreen` screen class:

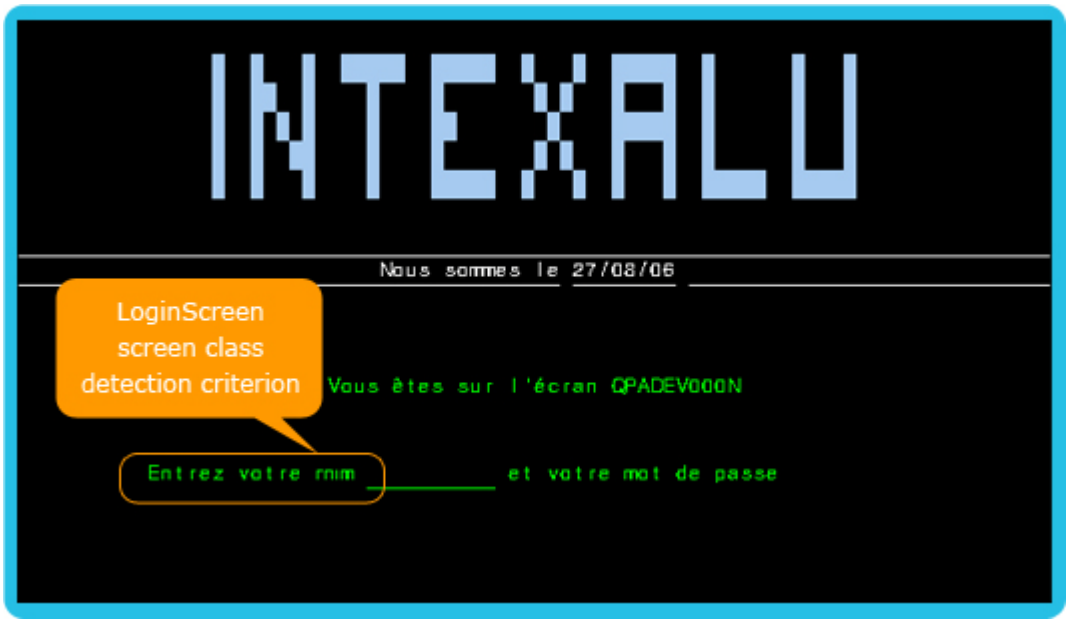


Figure 2 - 412: Javelin Screen class - LoginScreen screen class

Screens including an identifier are usually named after their identifier (here, `MNS10`):

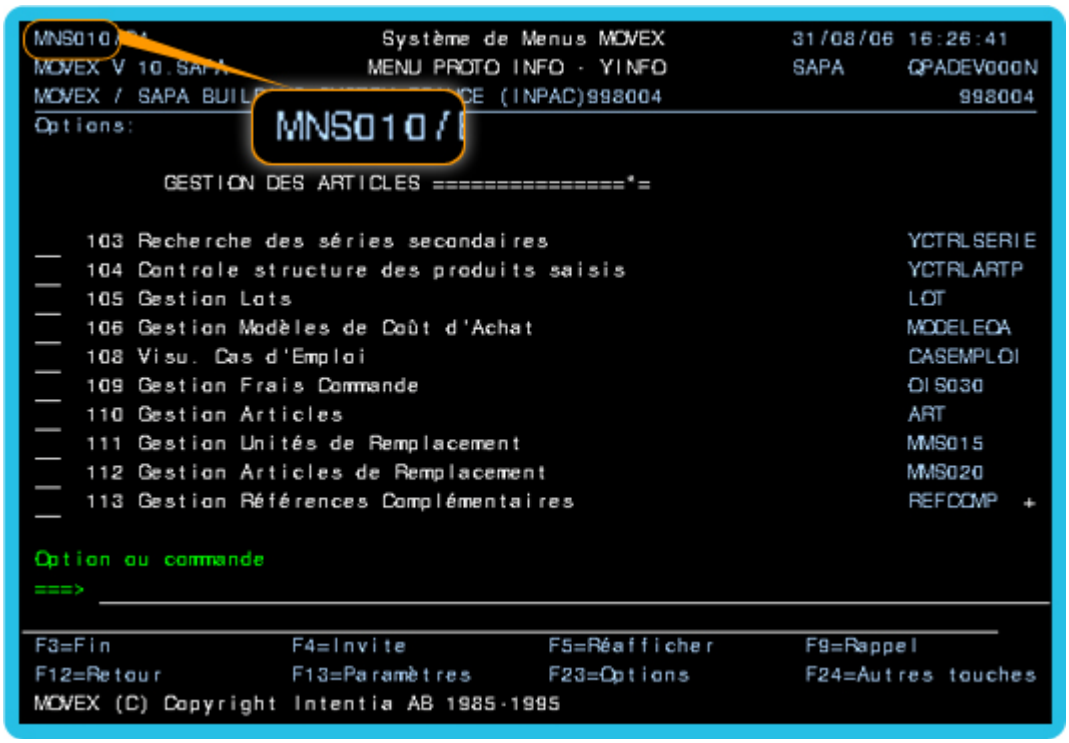


Figure 2 - 413: Javelin Screen class - MNS010 screen class

In the context of the `GetArticleData` transaction, when the `LoginScreen` screen class is detected, its associated screen class handler (entering the user name and password) is executed, then the subsequent screen class is detected.

Once created, *Javelin* screen classes appear together with their detection criteria and possible extraction rules in the **Projects** view of the Convertigo Studio:

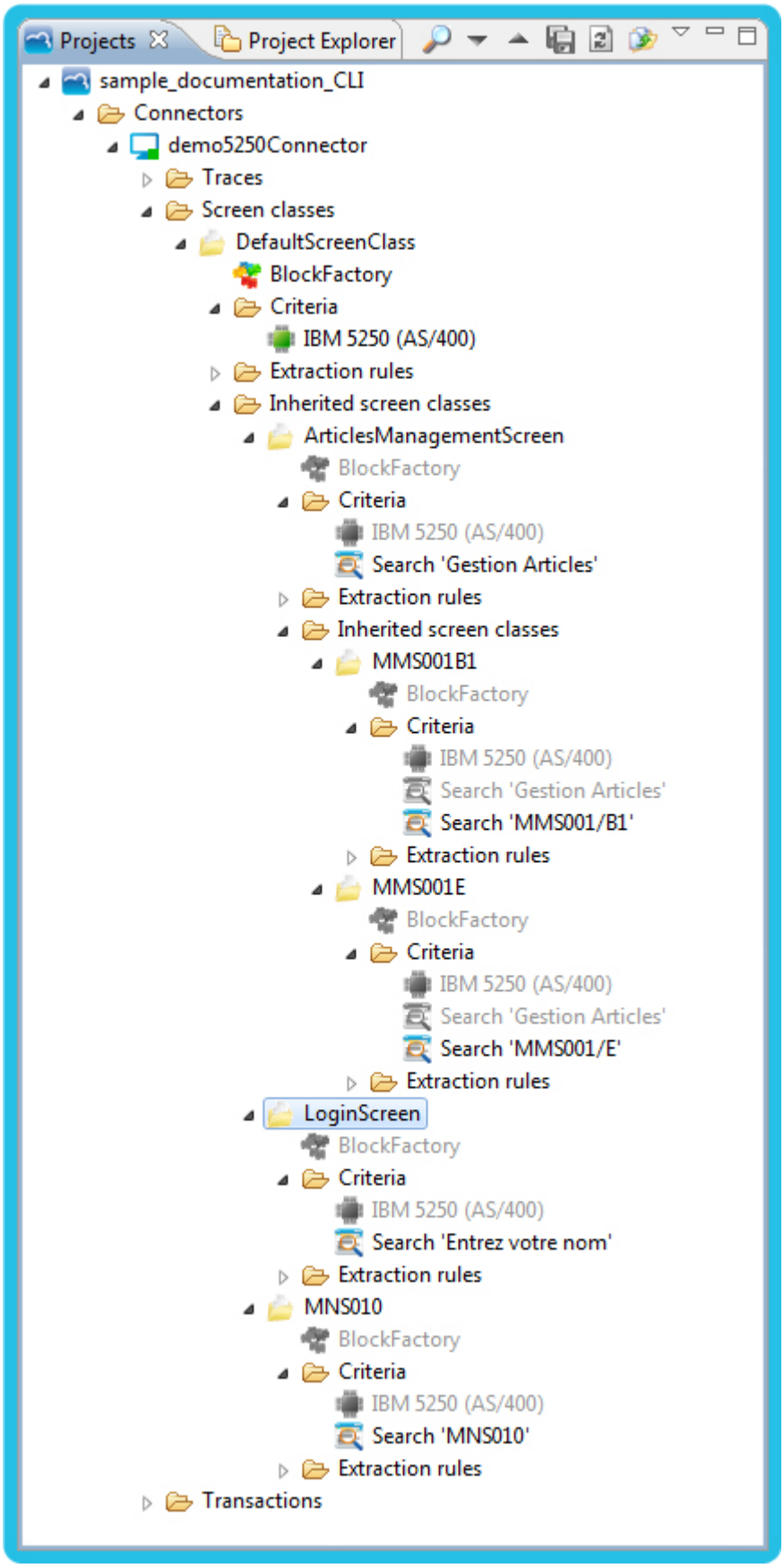


Figure 2 - 414: Javelin Screen class - Project's screen classes and respective criteria

Every screen class is always child to the `DefaultScreenClass` screen class. It therefore

inherits its default criteria and extraction rules:

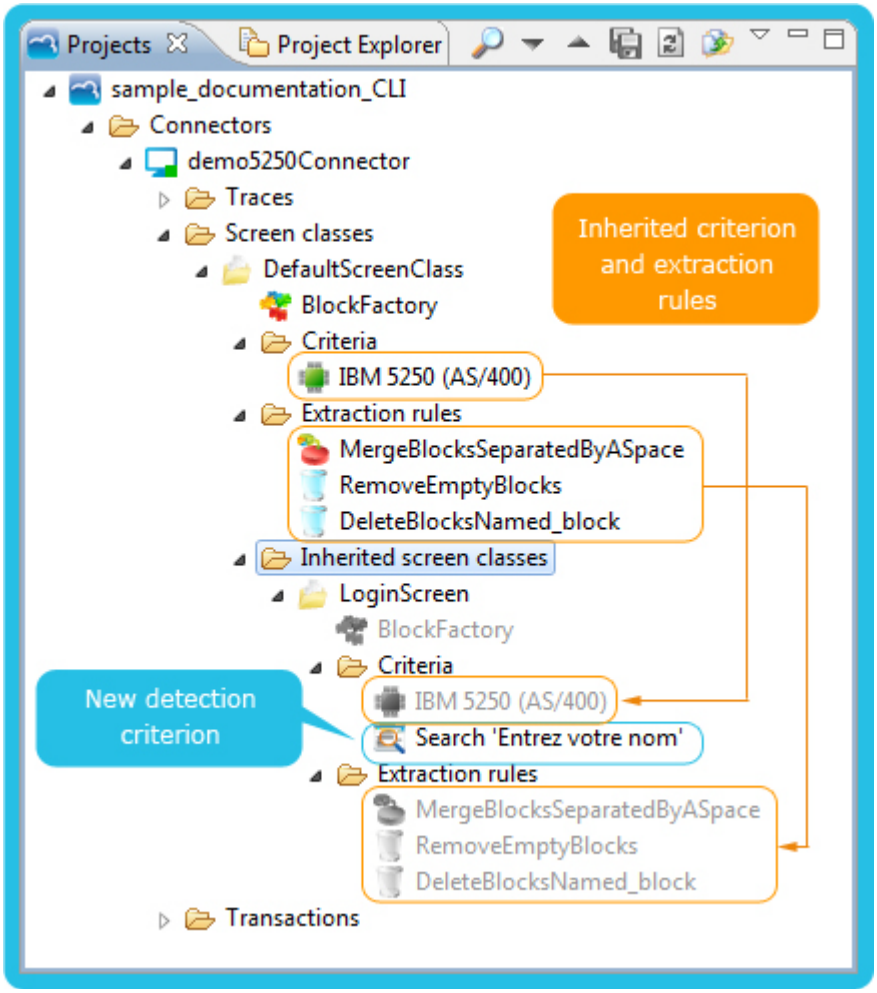


Figure 2 - 415: Javelin Screen class - Inherited criteria and extraction rules

Screen classes appear as follows in the **Properties** view (here, the `LoginScreen` screen class):

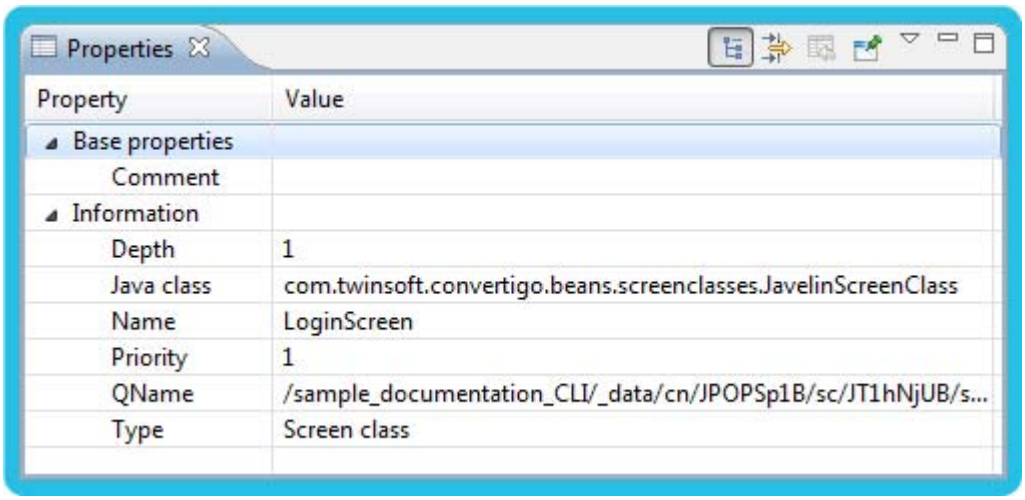


Figure 2 - 416: Javelin Screen class - LoginScreen screen class properties

For more information about the different criteria defined to detect these screen classes, see "*Find string criterion*" and "*Emulator technology criterion*" documentations and examples.

DEFAULT BLOCK FACTORY

OBJECT DESCRIPTION

The block factory is the process that splits legacy screens into elementary blocks.

For extraction purpose, legacy application screens are broken into elementary entities called `blocks`. The process in charge of generating blocks is called *Default block factory*.

The block generation process can be divided into three major steps:

- The block factory analyses screen lines.
- For each line, the block factory extracts character strings having same attributes (text color, background color, bold, blinking...).
- For each character string having same attributes, the block factory breaks sentences into single words by performing a separation on blank spaces. For example, the sentence `the_nice_hat` (where "_" represents a blank space) is divided into five blocks: `the`, _, `nice`, _ and `hat`. Consecutive blank spaces are grouped into one single block.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

EXAMPLES

The following example illustrates how the *Default block factory* processes a legacy screen and breaks it into blocks.

Before the block factory applies, the screen is a sequence of characters.

After it applies, the blocks are generated as described previously:

Figure 2 - 417: Block generation on legacy application screen (partial view of screen)

2.9.2 Criteria

EMULATOR TECHNOLOGY 

OBJECT DESCRIPTION

Defines the emulator technology used as a criterion for detecting screen classes.

A legacy project declares a unique *Emulator technology* criterion. It is always associated with the project's root screen class and cannot be changed or deleted. It is inherited and should match for all screen classes of a given project.

Available legacy technologies are :

- Bull DKU 7107
- IBM 3270
- IBM 5250
- Minitel
- Unix VT220

Matching condition: This criterion always matches as the emulator technology is defined by the legacy connector which connects to the legacy application.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EMPTY SCREEN 

OBJECT DESCRIPTION

Defines a legacy screen criterion searching for an empty screen.

The *Empty screen* criterion applies to legacy applications, which are usually launched from empty screens. It can be useful to define a screen class in order to detect when Convertigo doesn't manage to connect to the application.

In this case, it is recommended to use this criterion instead of a *Regular expression* criterion for optimizing response time.

Matching condition: This criterion matches the screen class when all screen map lines are empty.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EXAMPLES

If we consider the following legacy screen:

Figure 2 - 418: Empty screen criterion - Legacy screen

We can notice that this screen is empty wherever the block factory has splitted it into blocks representing screen lines.

The *Empty screen* criterion will match on this legacy screen.

FIND STRING 

OBJECT DESCRIPTION

Defines a legacy screen criterion searching for a string into the screen using possibly given presentation attributes.

Matching conditions: The *Find string* criterion matches on a screen when:

- the text starting at a given position (line, column) in the screen corresponds to the searched string,
- this text has defined presentation attributes.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Column	int	standard	Defines the column from which the criterion should match.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Line	int	standard	Defines the line from which the criterion should match.
Presentation attributes	int	standard	Defines the presentation attributes to match. This property allows to configure the criterion so that it matches only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or not to take into account. • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.
String	String	standard	Defines the string to search.

EXAMPLES

Let's consider the following legacy screen:

Figure 2 - 419: Find string criterion - Legacy screen

A *Find string* criterion is defined as follows:

```
Find string [
   line=0
   column=34
   attributes=[foreground="white", background="black"]
   string="Ouverture"
   reverse=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

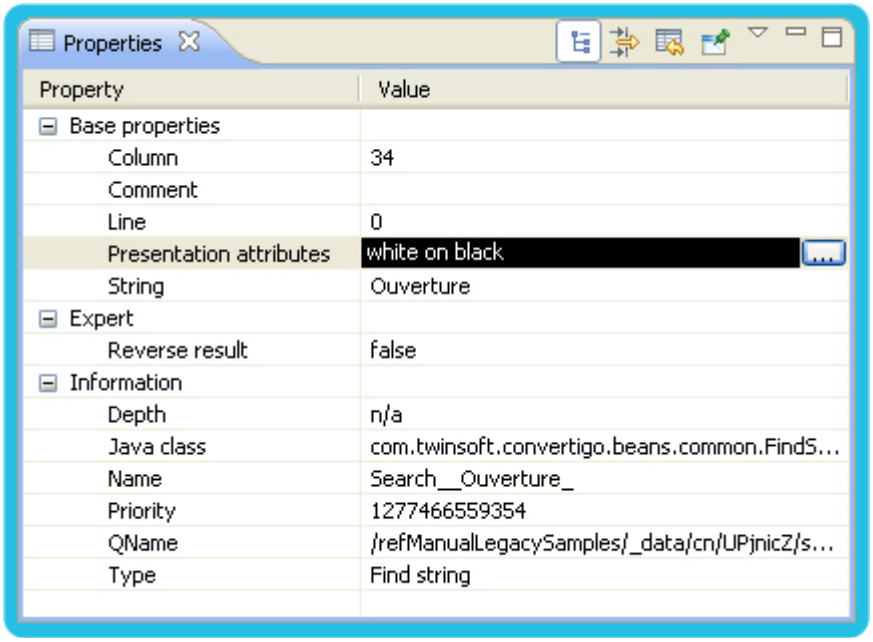


Figure 2 - 420: Find string criterion - Configuration example

This criterion matches on the previous legacy screen.

Regular expression (Legacy) 

Object Description

Defines a legacy screen criterion searching for a regular expression into the screen using possibly given presentation attributes.

Matching conditions: The *Regular expression* criterion matches on a screen when:

- the text starting at a given position (line, column) in screen corresponds to the defined regular expression,
- this text is defined with parameterized presentation attributes.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Column	int	standard	Defines the column from which the criterion should match.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Line	int	standard	Defines the line from which the criterion should match.
Presentation attributes	int	standard	Defines the presentation attributes to match. This property allows to configure the criterion so that it matches only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or not to take into account. • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

Property	Type	Category	Description
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EXAMPLES

Let's consider the following legacy screen:

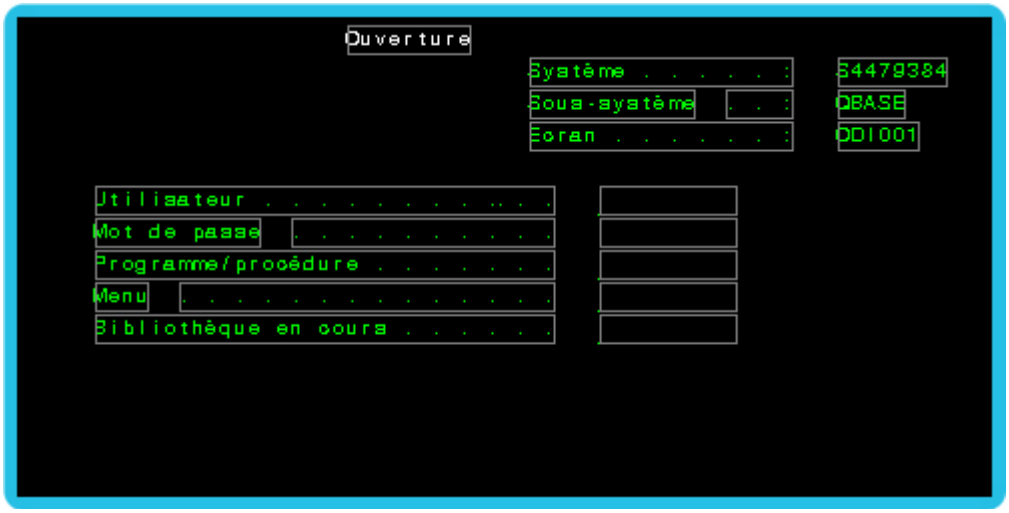


Figure 2 - 421: Regular expression criterion - Legacy screen

A *Regular expression* criterion is defined as follows:

```
Regular expression [
   line=0
   column=34
   attributes=[foreground="white", background="black"]
   Regular expression="O(u.+){2}"
   reverse=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

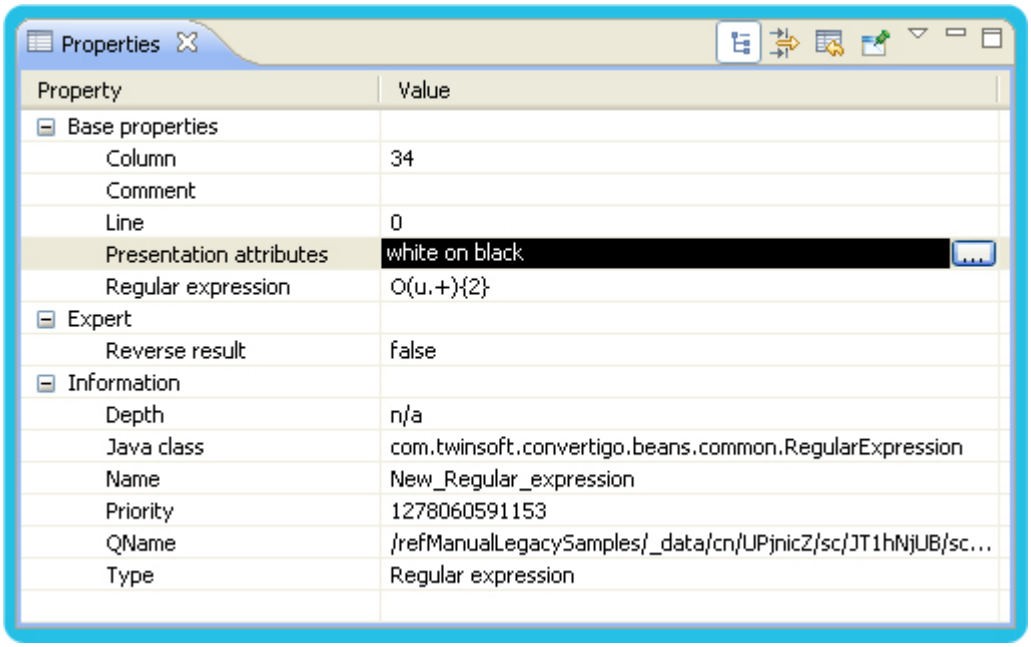


Figure 2 - 422: Regular expression criterion - Configuration example

This criterion matches on the previous legacy screen.

2.9.3 Extraction rules

PRESENTATION

STYLE OF BLOCKS 

OBJECT DESCRIPTION

Defines a `style` attribute on blocks displayed in HTML presentation.

The *Style of blocks* extraction rule is used to change the presentation style of webized screen elements. It adds a `style` attribute to blocks matching its selection parameters. The value of this attribute is a string in CSS format either:

- built by Convertigo using the different properties of the rule,
- or directly defined in the **Free style** property by Convertigo programmer.

With appropriate XSL templates, this attribute is used to change the style of the displayed blocks.

Notes:

- The *Style of blocks* rule is used on `static` (i.e. non editable) blocks.
- Only the `style` attribute is added to extracted XML elements.
- Since this rule does not create a new block type, it does not involve any specific XSL stylesheet. However, the `static` XSL template does use the `style` attribute.
- There is no need for the Convertigo developer to know CSS syntax to use this extraction rule (unless he/she wants to use the **Free Style** parameter).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Background color	int	configuration	Defines the background color of the block. The color is chosen into a list of colors.
Bold	boolean	configuration	Defines whether the font is `bold` or not.
Border	boolean	configuration	Defines whether the element has a border or not.
Border color	int	configuration	Defines the element border color. The color is chosen into a list of colors.
Border width	String	configuration	Defines the border width (in pixels).

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Font color	int	configuration	Defines the font color. The color is chosen into a list of colors.
Font family	int	configuration	Defines the font family. The font family is chosen into a list of fonts.
Font size	String	configuration	Defines the font size (`pt` for point, `px` for pixel).
Free style	String	configuration	CSS formatted style, takes place of other styles defined here. If used, this parameter overrides all other parameters. The Convertigo developer can input a string to be used as value for the `style` XML attribute. In order to be compatible with Convertigo default XSL templates, this string must be in proper CSS syntax.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Italic	boolean	configuration	Defines whether the font is `italic` or not.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Underlined	boolean	configuration	Defines whether the font is `underlined` or not.

EXAMPLES

If we consider the following legacy screen:

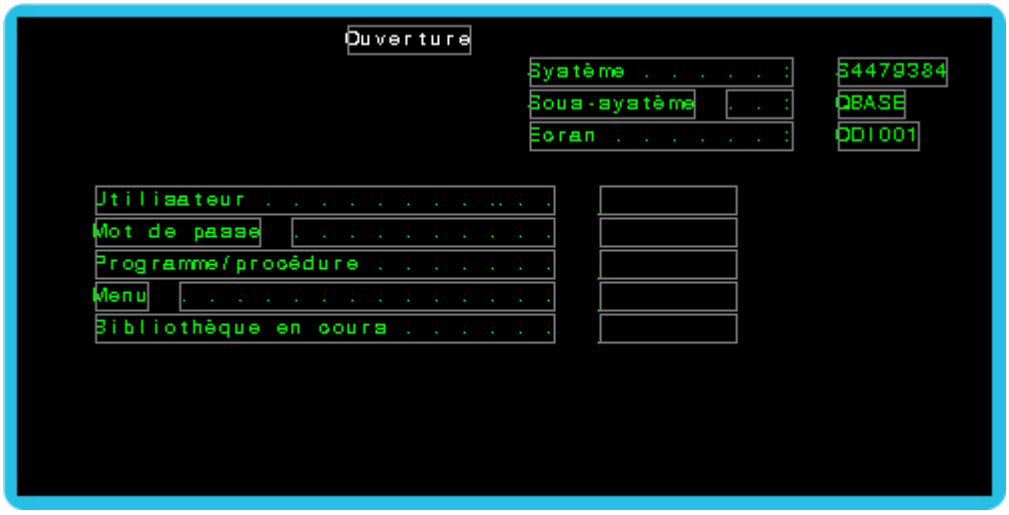


Figure 2 - 423: Style extraction rule - Legacy screen

Without the rule, the screen title is extracted into the following XML:

```
<blocks page-number="0">
  <block background="black" column="34" foreground="white" line="0" name="untitled" reverse="false" type="static">Ouverture</block>
  <block background="black" column="47" foreground="green" line="1" name="untitled" reverse="false" type="static">Système . :</block>
```

Figure 2 - 424: Style extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

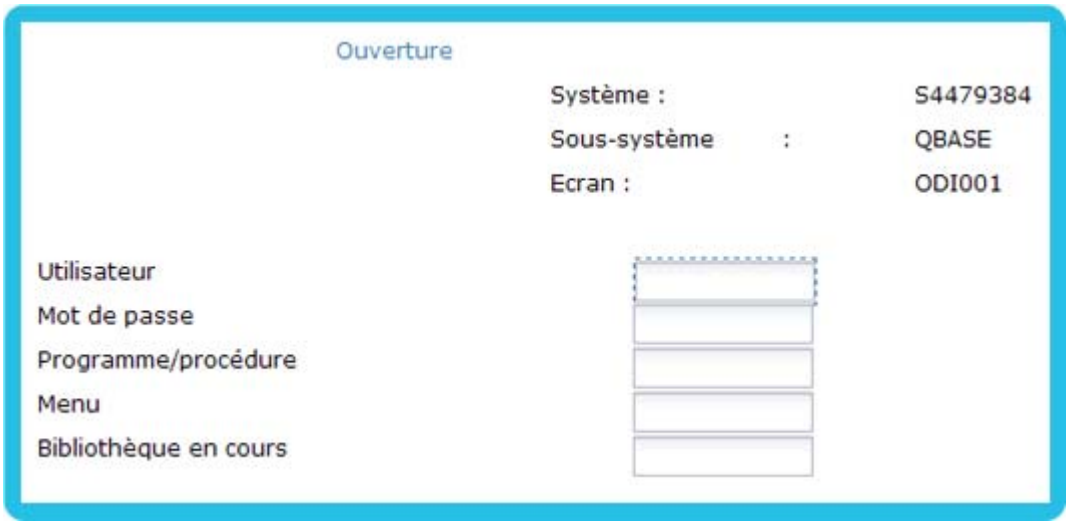


Figure 2 - 425: Style extraction rule - Webized page without rule

In this example, we want to increase the title visibility. A *Style of blocks* extraction rule is created with the following parameters:

```
Style of blocks [
   screen zone=[x=0, y=0, width=80, height=4]
   attributes=[foreground="white" background="black"]
   bold=true
   font color="coral"
   font family="Verdana"
   font size="14pt"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

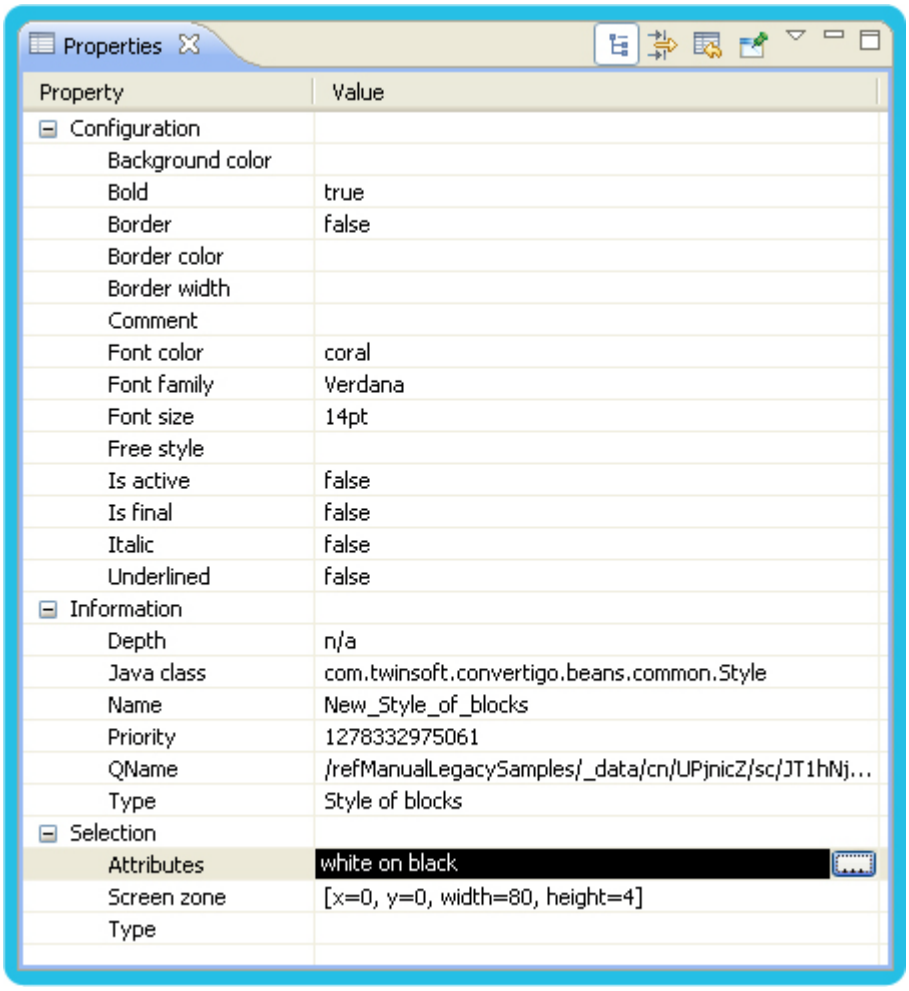


Figure 2 - 426: Style extraction rule - Configuration example

When the rule is executed, a `style` attribute is added to each matching block and the resulting XML is as follows:

```
itled" style="font-family:Verdana; font-size:14pt; font-weight:bold; color: coral;" type="static">Ouverture</block>
titled" type="static">Système  :</block>
```

Figure 2 - 427: Style extraction rule - Resulting XML with rule

After XSL transformation, thanks to XSL templates, the screen appears webized:

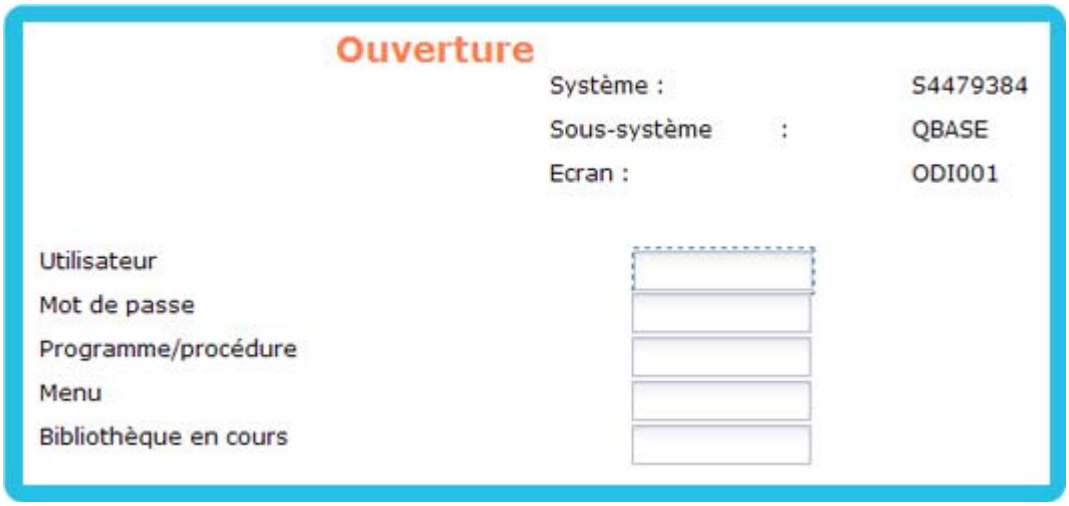


Figure 2 - 428: Style extraction rule - Webized page with rule

CONTAINER (LEGACY) 

OBJECT DESCRIPTION

Defines a container block grouping blocks together.

The *Container* extraction rule groups XML elements with same characteristics under a unique container XML element. This container element can be processed by a specific user-defined XSL style sheet.

For example, buttons webized via the *Command* extraction rule can be grouped as a menu and moved where required, or labels and fields composing a form can be grouped together and displayed in a frame.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Container layout	XMLRectangle	configuration	Defines the screen zone in which the container has to be displayed. This property allows to move the created container element to a specific area of the screen. The created block will be created with the specified screen zone values as positioning attributes. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value. These positioning attributes have to be handled by the XSL template rule that displays the container.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
XML tag	String	configuration	Defines the container element XML tag name (default: `container`). In order to be processed by XSL template rule, this element should have a specific tag name. This property allows to configure the XML tag name of the created element.

EXAMPLES

If we consider the following legacy screen:

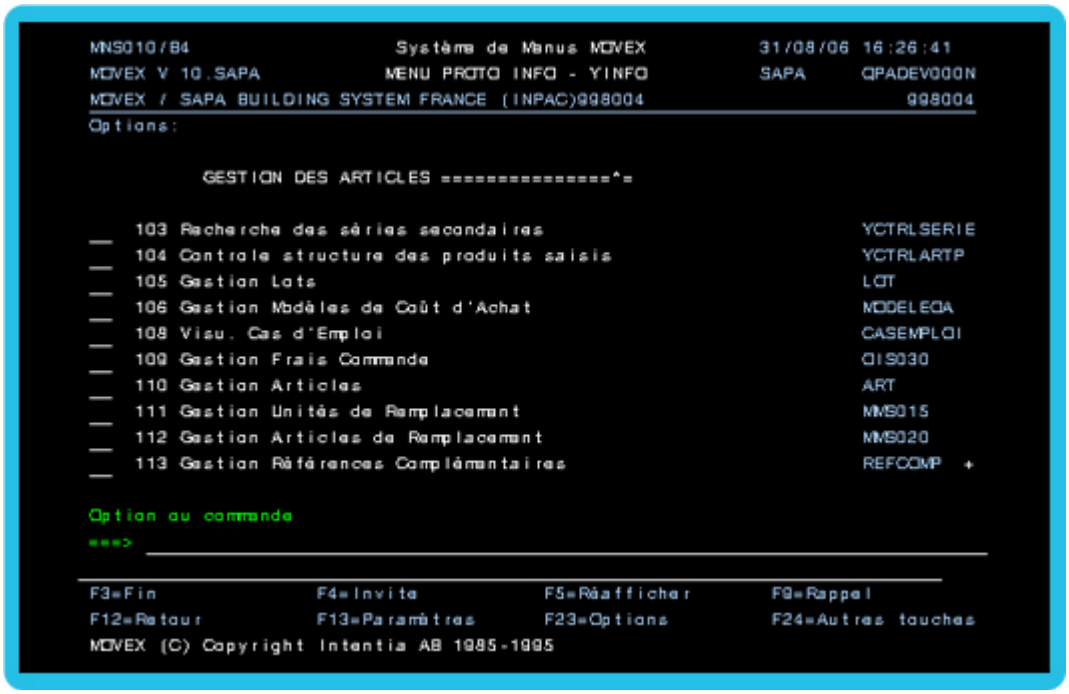


Figure 2 - 429: Container extraction rule - Legacy screen

We can notice that command keywords are present on the bottom of the screen. They are handled by a specific extraction rule and displayed as buttons in the webized screen.

Figure 2 - 430: Container extraction rule - Webized page without rule

The XML resulting from this screen is as follows:

Figure 2 - 431: Container extraction rule - Resulting XML without rule

In this example, we want to move the commands buttons to a left menu instead of the bottom of the page. To do so, a *Container* extraction rule is created with the following parameters:

```
Container [
   type="keyword"
   xml tag="commands"
   container layout=[x=0, y=0, width=-1, height=-1]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

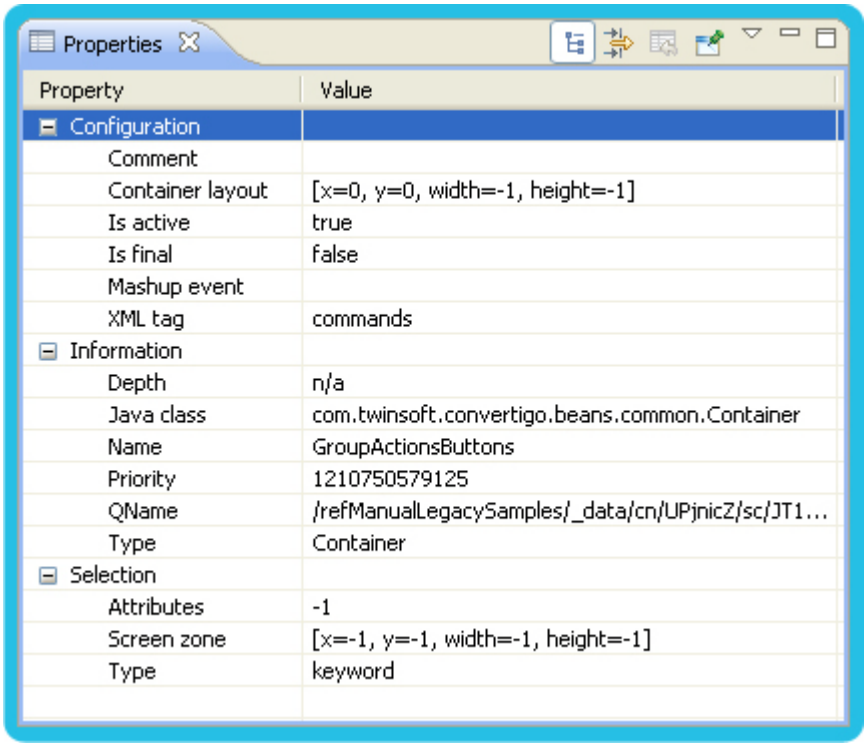


Figure 2 - 432: Container extraction rule - Configuration example

Type property is set to `keyword` so as command blocks are matching.

Container layout property is edited in the **Screen zone** editor:

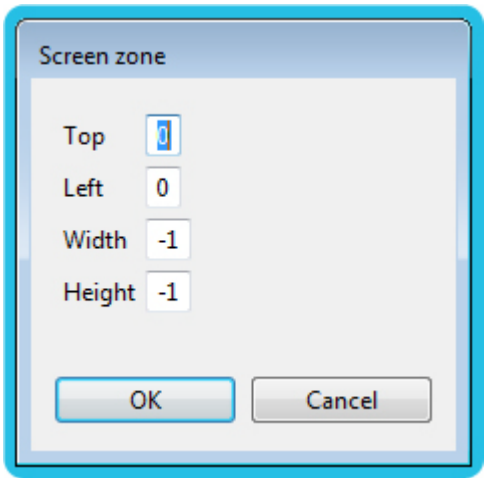


Figure 2 - 433: Container extraction rule - Container layout property edition

Top and **Left** values are set to `0` in order to position the container block to the top left corner of the page.

When the rule is executed, the resulting XML includes the matching keyword blocks into a `container` type element:

Figure 2 - 434: Container extraction rule - Resulting XML with rule

After XSL transformation, thanks to `commands` XSL template, it appears webized:

Figure 2 - 435: Container extraction rule - Webized page with rule

COMMON GUI COMPONENTS

CHOICE 

OBJECT DESCRIPTION

Lists entries expected in a field.

The *Choice* extraction rule turns a one-character field whose expected content belongs to a fixed list (for example Y=Yes, N=No) into an XML element of `choice` type (combo box or radio buttons) with a predefined list of actions.

Note: XML elements of the `choice` type are handled by the `choice` XSL template described in the `choice.xsl` file. To change the way choices are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action label separators	String	configuration	When extracting actions from the screen, defines the character(s) separating a label from the corresponding action value (if applicable). For example, if the list of choices is (Y=Yes, N=No), the action label separator is "=". This property is used only if actions are extracted from screen (**Options from screen** property set to `true`).
Actions	XMLVector	configuration	Defines the actions table listing possible actions. The **Actions** table is a two-column table: • **Label**: defines the action label to be displayed in the combo box (for example "Yes"), • **Command**: defines the action value to be sent in the field (for example "Y"). Notes: • A new action can be added to the list using the blue keyboard icon. The HTTP headers defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • This property is used only if actions are not extracted from screen (**Options from screen** property set to `false`).
Actions separators	String	configuration	When extracting actions from the screen, defines the character(s) separating actions from each others (if applicable). For example, if the list of choices is (Y=Yes, N=No), the actions separator is ",". This property is used only if actions are extracted from screen (**Options from screen** property set to `true`).

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
End pattern	String	configuration	When extracting actions from the screen, defines the actions block end pattern (if applicable). For example, if actions are grouped between brackets, the end pattern is the closing bracket. The rule is applied only if the block ends with this pattern. This property is used only if actions are extracted from screen (**Options from screen** property set to `true`).
Extraction policy	int	configuration	Defines the policy for extracting the action value (one character to send in the field) related to each action of the action list. This property defines how to find the action value to send in the field corresponding to each action label from the action list. It can take three values: • `index`: the action value is its rank in the action list, • `character separator`: the action value is separated from its label thanks to a character separator, which is defined in **Action label separators** property, • `first upper letter`: the action value is the first upper letter the action label. This property is used only if actions are extracted from screen (**Options from screen** property set to `true`).
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Options from screen	boolean	configuration	Defines whether options should be retrieved from the screen. If set to `true`, possible actions are extracted from the screen. Otherwise, actions are as set in the **Actions** table.

Property	Type	Category	Description
Radio buttons	boolean	configuration	Defines whether options should be displayed as radio buttons or combo box. If set to `true`, the extraction rule adds a `radio` attribute with the value `true` to the choice XML element. Otherwise, it adds a a `radio` attribute with the value `false`. This attribute is processed by the `choice` XSL template in the `choice.xsl` file. When attribute value is `true`, possible actions are displayed as radio buttons, otherwise, possible actions are displayed as a combo box.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Start pattern	String	configuration	When extracting actions from the screen, defines the actions block start pattern (if applicable). For example, if actions are grouped between brackets, the start pattern is the opening bracket. The rule is applied only if the block starts with this pattern. This property is used only if actions are extracted from screen (**Options from screen** property set to `true`).
Tag name	String	configuration	Defines the tag name of the element generated in output XML after extraction (by default: `choice`). By default, the generated element is of `choice` type. It is processed by the `choice` XSL template in the `choice.xsl` file.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

In this example, we are dealing with a choice which actions are extracted from screen, with no label (labels are actions values).

Let's consider the following legacy screen:

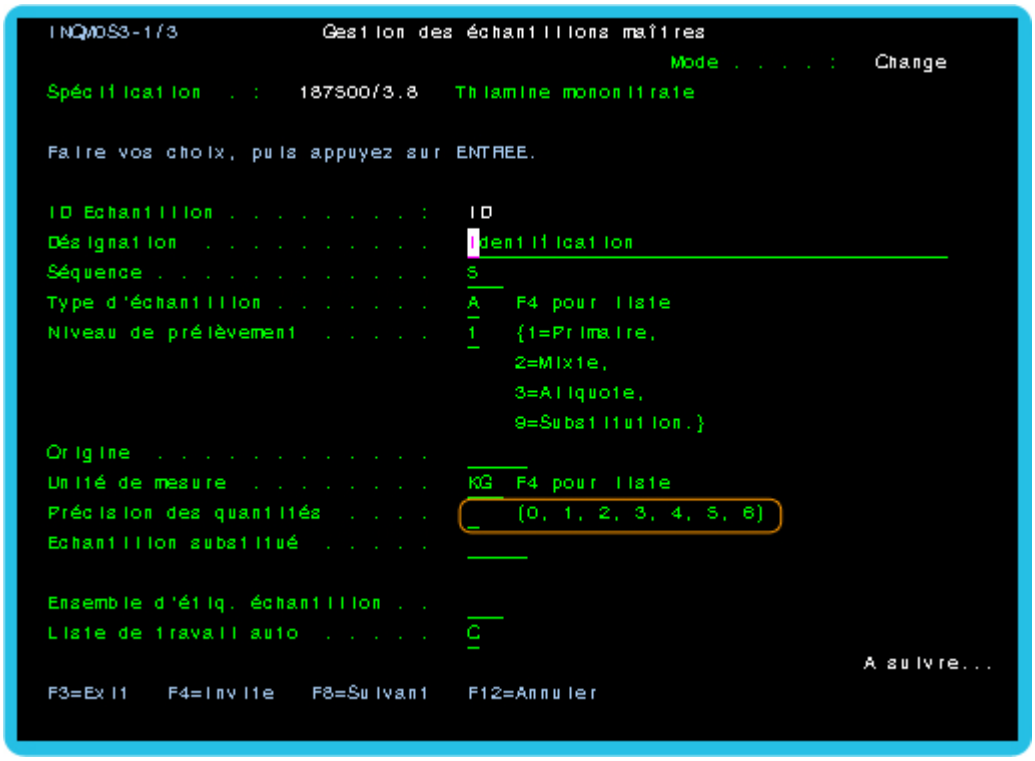


Figure 2 - 436: Choice extraction rule - Legacy screen

We can notice that the *Précision des quantités* field is a choice: the list of values to insert in the field is precised between brackets on the right of the field.

Without the rule, the resulting XML is as follows:

```
<block background="black" column="1" foreground="green" line="16" name="untitled" type="static">Précision des quantités</block>
<block background="black" column="26" foreground="green" line="16" name="untitled" type="static">. . .</block>
<block background="black" column="32" foreground="green" line="16" name="untitled" type="static">.</block>
<block background="black" column="36" foreground="green" line="16" name="__field_c36_l16" size="1" type="field"/>
<block background="black" column="40" foreground="green" line="16" name="untitled" type="static">(0, 1, 2, 3, 4, 5, 6)</block>
```

Figure 2 - 437: Choice extraction rule - Resulting XML without rule

In this example, a *Choice* extraction rule is created with the following parameters:

```
Choice [
   screen zone=[x=36, y=16, width=25, height=1]
   tag name="choice"
   options from screen=true
   start pattern="("
   end pattern=")"
   actions separators=","
   extraction policy="Character separator"
   actions label separators=""
   radio=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

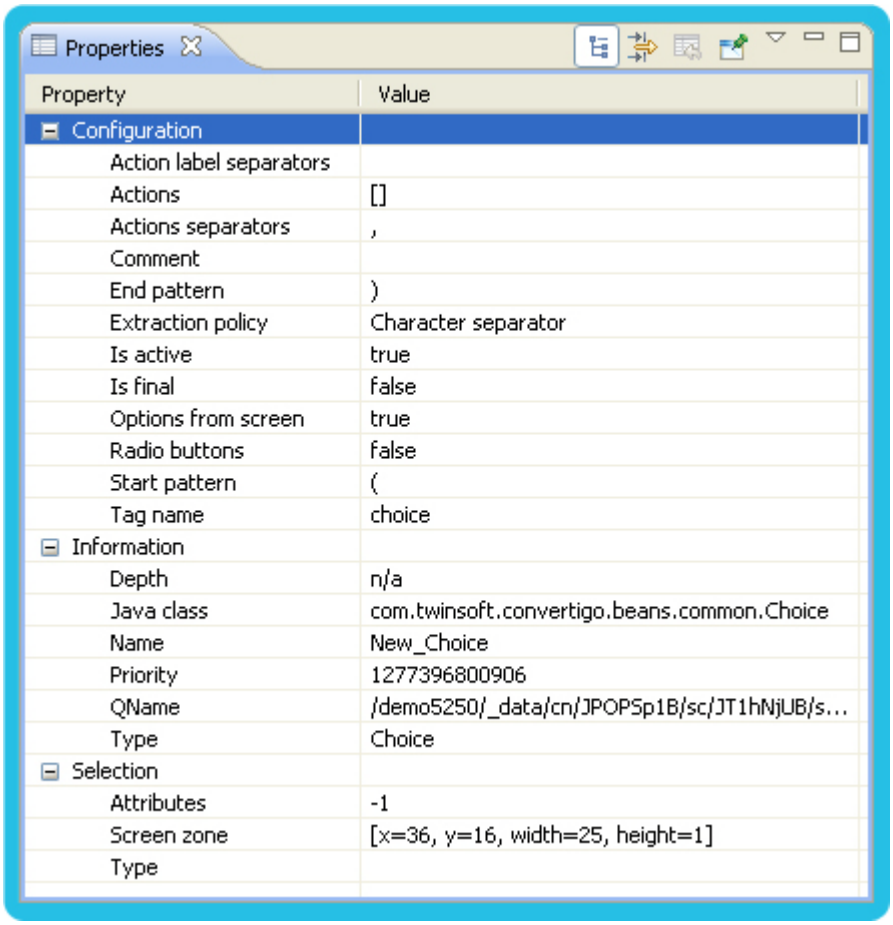


Figure 2 - 438: Choice extraction rule - Configuration example

When the rule is executed, the resulting XML includes the defined choice on matching field block:

```
<choice background="black" column="36" foreground="green" line="16"
name="__field_c36_l16" radio="false" size="1" type="choice">
  <item action="0" type="choiceItem" value="0"/>
  <item action="1" type="choiceItem" value="1"/>
  <item action="2" type="choiceItem" value="2"/>
  <item action="3" type="choiceItem" value="3"/>
  <item action="4" type="choiceItem" value="4"/>
  <item action="5" type="choiceItem" value="5"/>
  <item action="6" type="choiceItem" value="6"/>
  <item action="" selected="true" type="choiceItem" value=""/>
</choice>
```

Figure 2 - 439: Choice extraction rule - Resulting XML with rule set to combo box

After XSL transformation, thanks to `choice` XSL template, it appears webized:

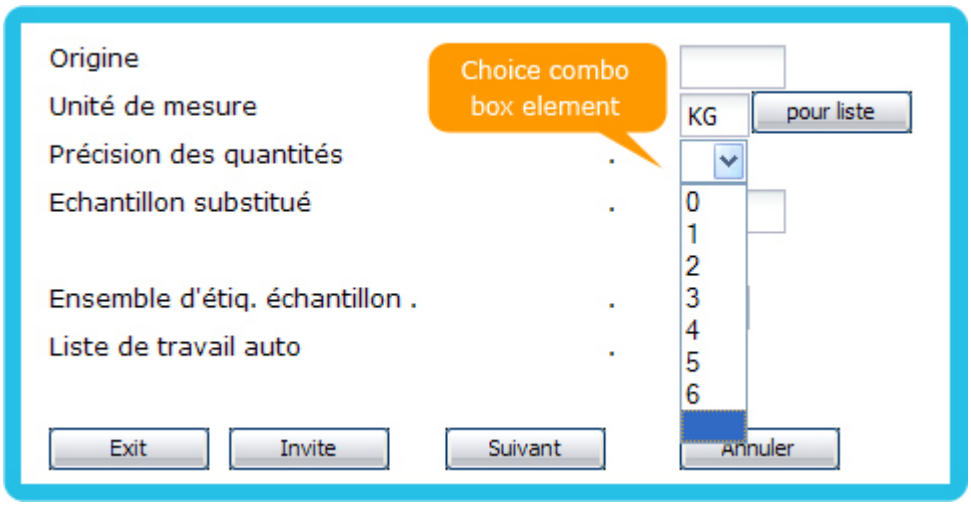


Figure 2 - 440: Choice extraction rule - Webized page with rule set to combo box

When changing the value of the **radio** property to `true`, the resulting XML is as follows:

```
<choice background="black" column="36" foreground="green" line="16"
name="__field_c36_l16" radio="true" size="1" type="choice">
  <item action="0" type="choiceItem" value="0"/>
  <item action="1" type="choiceItem" value="1"/>
  <item action="2" type="choiceItem" value="2"/>
  <item action="3" type="choiceItem" value="3"/>
  <item action="4" type="choiceItem" value="4"/>
  <item action="5" type="choiceItem" value="5"/>
  <item action="6" type="choiceItem" value="6"/>
  <item action="" selected="true" type="choiceItem" value=""/>
</choice>
```

Figure 2 - 441: Choice extraction rule - Resulting XML with rule set to radio buttons

After XSL transformation, thanks to `choice` XSL template, it appears webized:

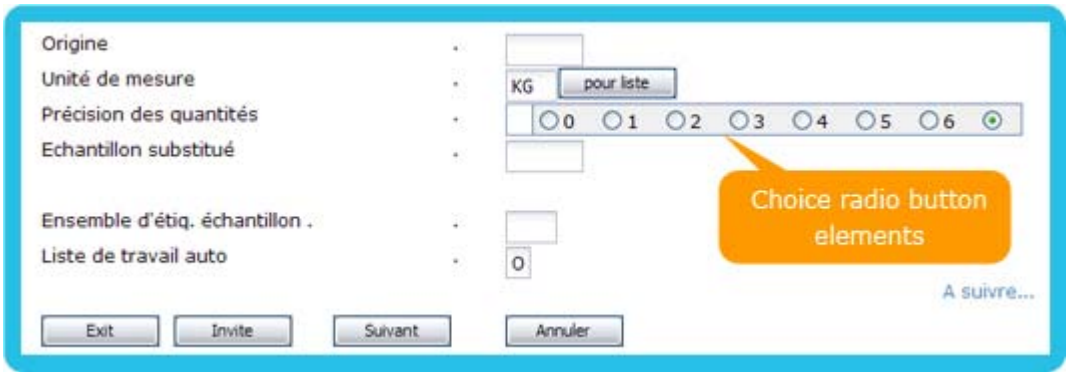


Figure 2 - 442: Choice extraction rule - Webized page with rule set to radio buttons

COMMANDS 

OBJECT DESCRIPTION

Defines and handles keywords and commands found in legacy screens.

The *Commands* extraction rule detects keywords in the screen and transforms them into `keyword` type blocks.

Each keyword must be found in a list provided in the **Keywords table** parameter of the rule.

The rule adds following XML attributes to matching blocks:

- `action`: the javelin action to be executed on the mainframe,
- `data`: optional additional data to be sent with the action.

Note: XML elements of the `keyword` type are handled by the `keyword` XSL template described in the `keyword.xsl` file. To change the way keywords are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Case dependency	boolean	configuration	Defines whether letter case should be respected in keyword detection. If set to `false`, keywords match even if the case is not similar. For example, `pf13` and `PF13` match the `PF13` keyword.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Keywords table	XMLVector	configuration	Defines a list of keywords that can be detected, with replacement texts, optional data to send and associated action. This property is a list of *Keywords*. For each *Keyword* you can define: • **Keyword**: Keyword string to handle when found in the screen, • **Replace Text**: Replacement text for the keyword (hotspot label, optional), • **Sent data**: Data to be sent before performing action (optionnal), • **Action**: Action key to be pressed when the user clicks on the button (action key corresponding to found keyword). Notes: • A new keyword can be added to the list using the blue keyboard icon. The keywords defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • The order of the keywords defined in this table is very important, it is used for detection priority. That means if two keywords can match on a block, only the first keyword from this table will be handled for this block.

Property	Type	Category	Description
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

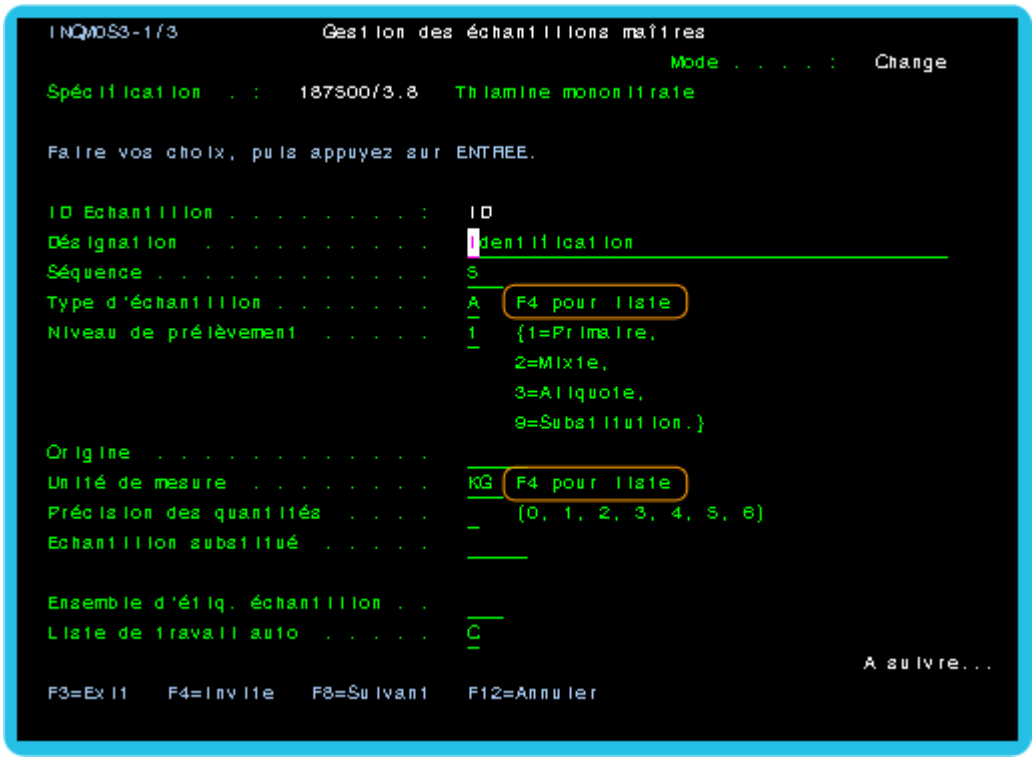


Figure 2 - 443: Commands extraction rule - Legacy screen

We can notice that two command keywords are present on the right of the screen, next to *Type d'échantillon* input and *Unité de mesure* input.

Without the rule, the resulting XML is as follows:

Figure 2 - 444: Commands extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

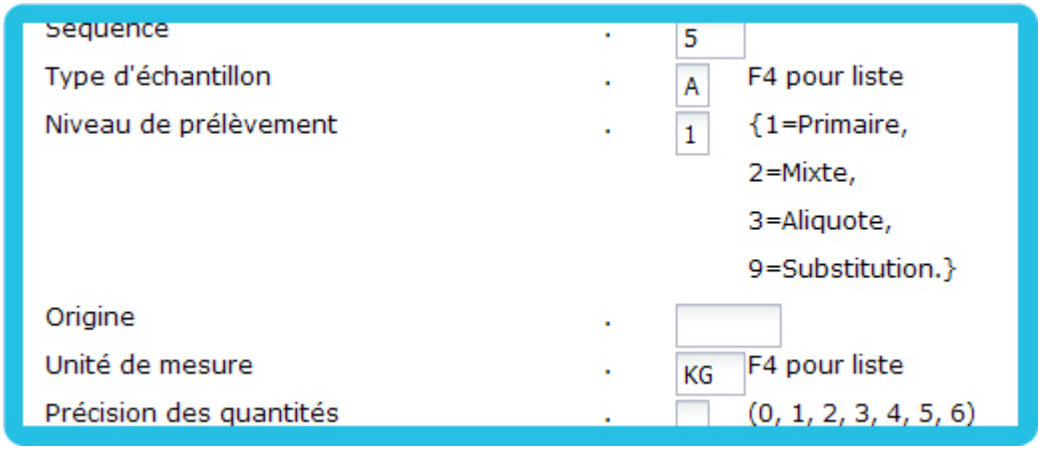


Figure 2 - 445: Commands extraction rule - Webized page without rule

In this example, a *Commands* extraction rule is created with the following parameters:

```
Commands [
   screen zone=[x=38, y=9, width=21, height=7]
   case dependency=false
   keywords table={
      Keyword [keyword="F4 pour liste" sent data=""
      replace text="Liste" action="KEY_PF4"]
   }
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

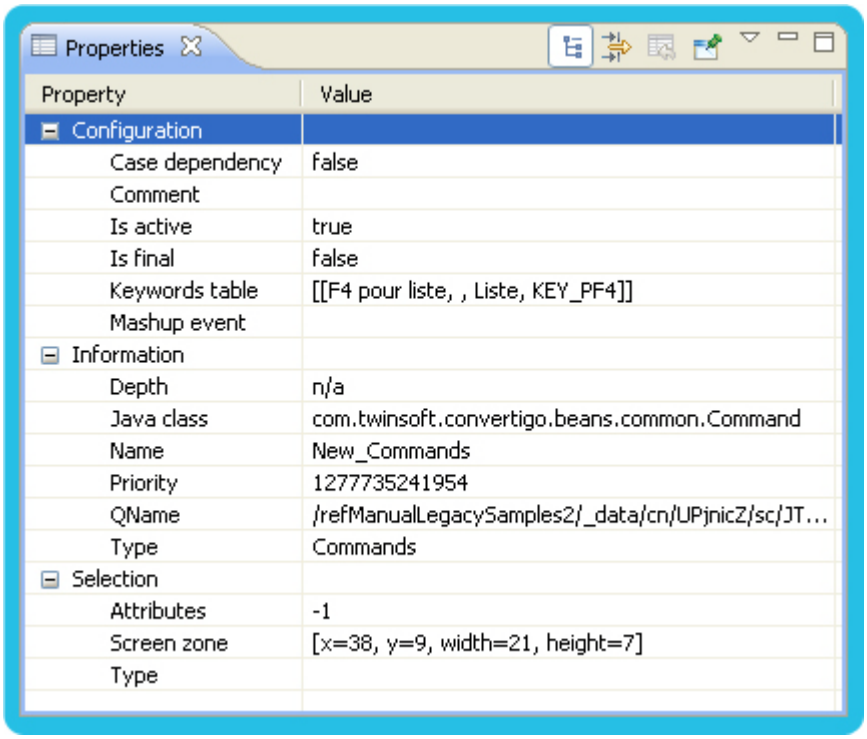


Figure 2 - 446: Commands extraction rule - Configuration example

Keywords table property is edited in the **Keywords** editor:

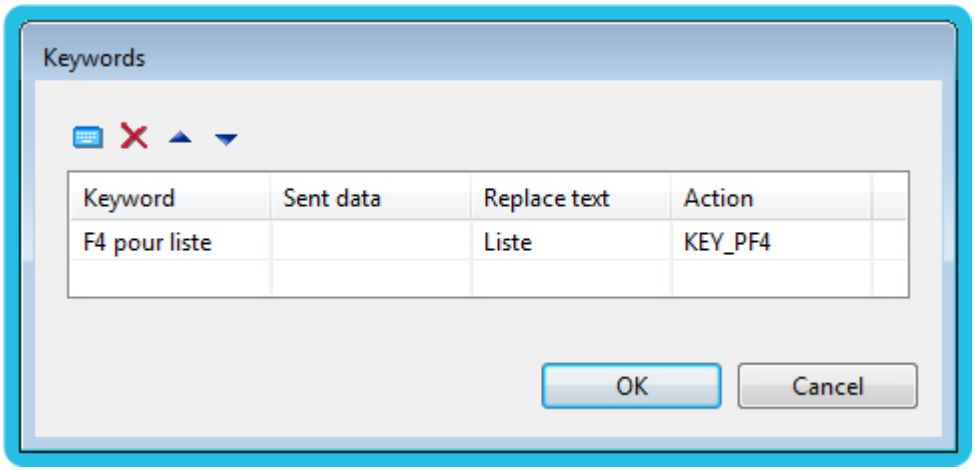


Figure 2 - 447: Commands extraction rule - Keywords table property edition

When the rule is executed, the resulting XML includes the defined `keyword` type elements on commands matching blocks:

Figure 2 - 448: Commands extraction rule - Resulting XML with rule

After XSL transformation, thanks to `keyword` XSL template, it appears webized:

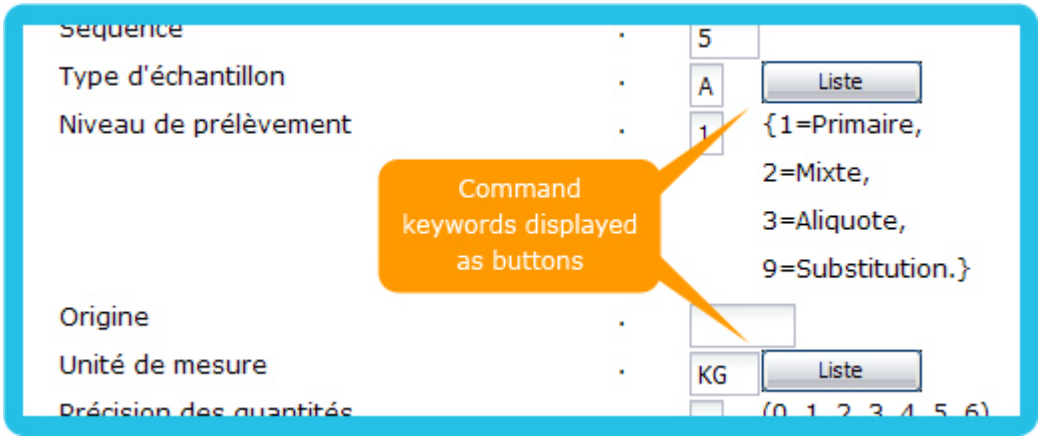


Figure 2 - 449: Commands extraction rule - Webized page with rule

FIELD/TEXT 

OBJECT DESCRIPTION

Adds a field or text on the webized screen.

Unlike other extraction rules, the *Field/Text* extraction rule does not extract data from green screen, but adds text elements on screen.

This rule adds `field` or `static` type XML element to the XML document. These elements are then processed by XSL transformation to display a field or text label in the HTML page.

Notes:

- XML elements of the `field` type are handled by the `field` XSL template described in the `field.xsl` file. To change the way fields are displayed in the HTML page, edit this file.
- XML elements of the `static` type are handled by the `static` XSL template described in the `static.xsl` file. To change the way texts are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Field attributes	int	configuration	Defines the generated field/text presentation attributes. This property allows to configure the presentation attributes to set to the created text/field, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".

Property	Type	Category	Description
Field layout	XMLRectangle	configuration	Defines the screen zone where the field/text is to be displayed. This property allows to position the created text or field element to a specific area of the screen. The created block will be created with the specified screen zone values as positioning attributes. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value. These positioning attributes have to be handled by the XSL template rule that displays the element.
Field name	String	configuration	Defines the field name. When data is submitted by the user through this field, it is sent to Convertigo as a variable named after this property. It can then be used in a transaction. Use the `__field_c<column>_l<line>` syntax to have Convertigo add data on screen at defined line and column.
Field type	String	configuration	Defines the field/text type. The element can be either of `field` or of `static` type.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Value	String	configuration	Defines the field or text value, depending on the element type. If the added element is of: • `field` type: the created field is filled with the value. • `static` type: the value is added as static text on screen.

EXAMPLES

Since the *Field/Text* rule does not extract any data from the screen, it is not related to any relevant green screen sample.

In this example, a first *Field/Text* extraction rule is created with the following parameters:

```
Field/Text [
   field layout=[x=16, y=10, width=16, height=1]
   field attributes=[foreground="green", background="black"]
   field type="static"
   value="New label"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

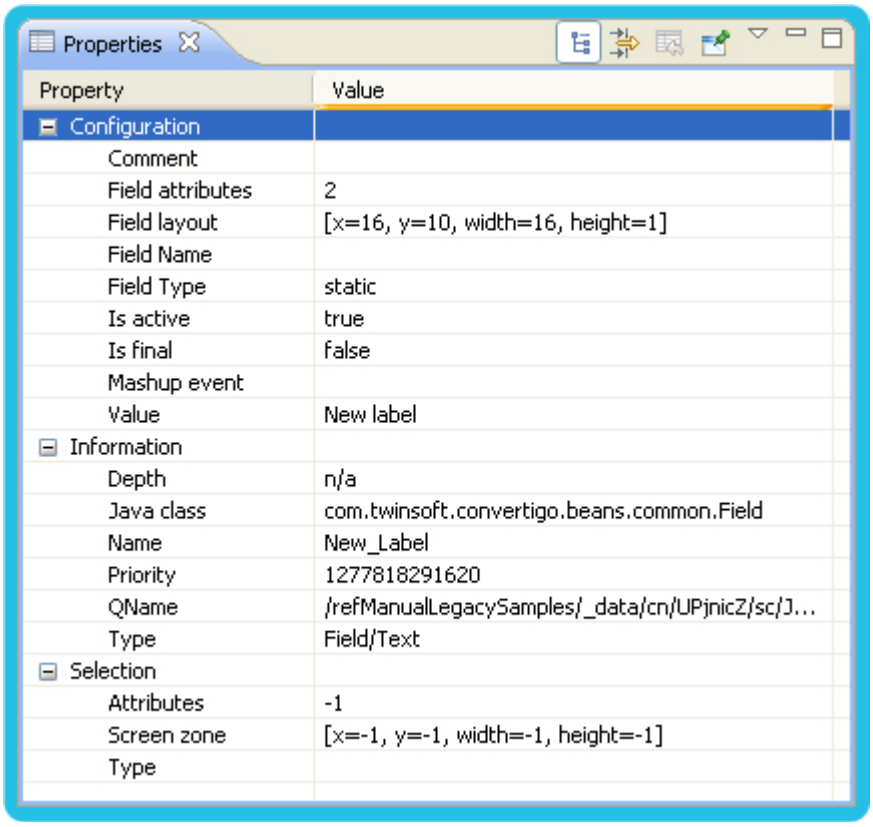


Figure 2 - 450: Field/Text extraction rule - First configuration example

Field layout property is edited in the **Screen zone** editor:

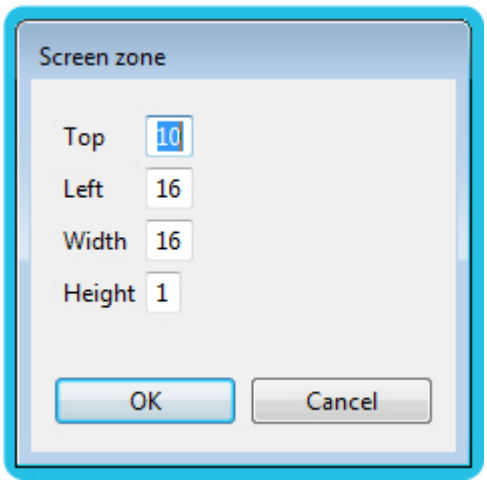


Figure 2 - 451: Field/Text extraction rule - Field layout property edition

Field attributes property is edited in the **Attributes** editor:

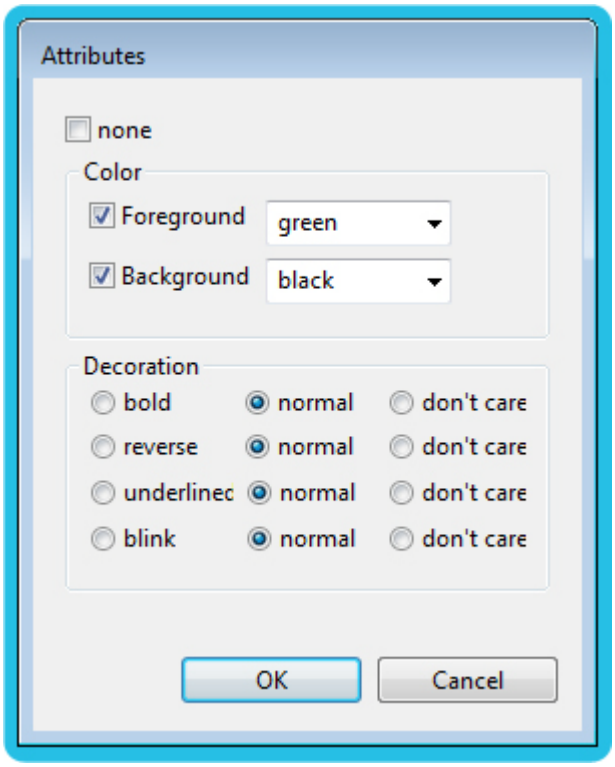


Figure 2 - 452: Field/Text extraction rule - Field attributes property edition

A second *Field/Text* extraction rule is created with the following parameters:

```
Field/Text [
  field layout=[x=52, y=10, width=10, height=1]
  field attributes=[foreground="green", background="black",
  underline=true]
  field type="field"
  field name="__field_c52_l10"
  value="New field content"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

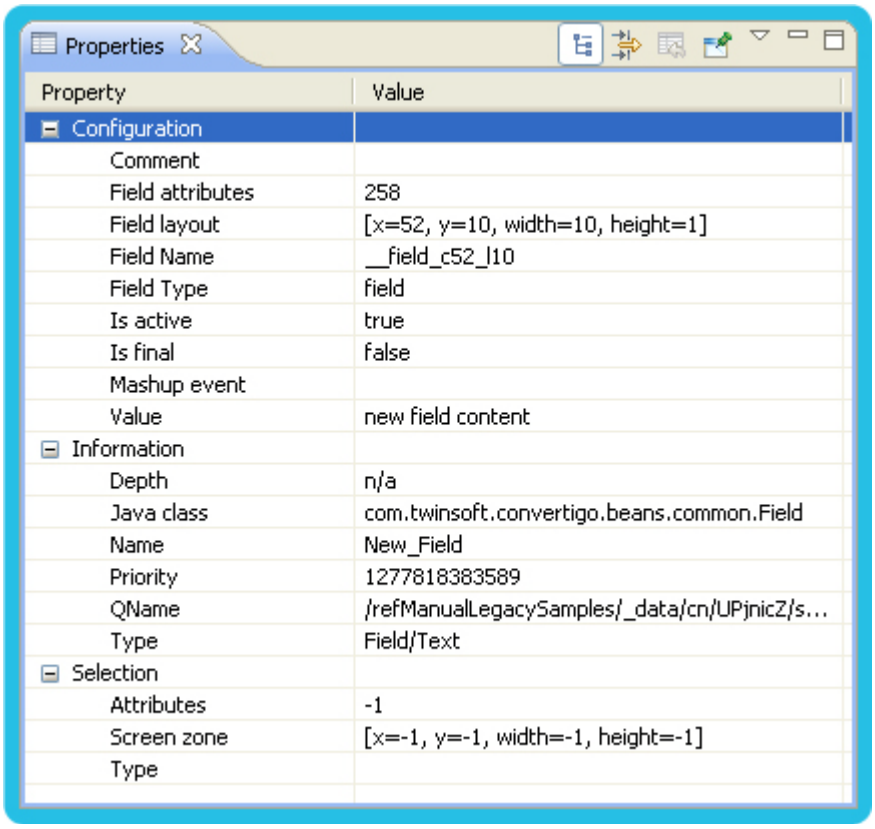


Figure 2 - 453: Field/Text extraction rule - Second configuration example

Field layout property is edited in the **Screen zone** editor:

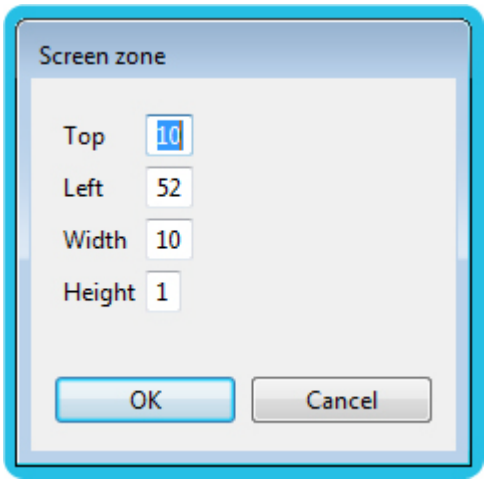


Figure 2 - 454: Field/Text extraction rule - Field layout property edition

Field attributes property is edited in the **Attributes** editor:

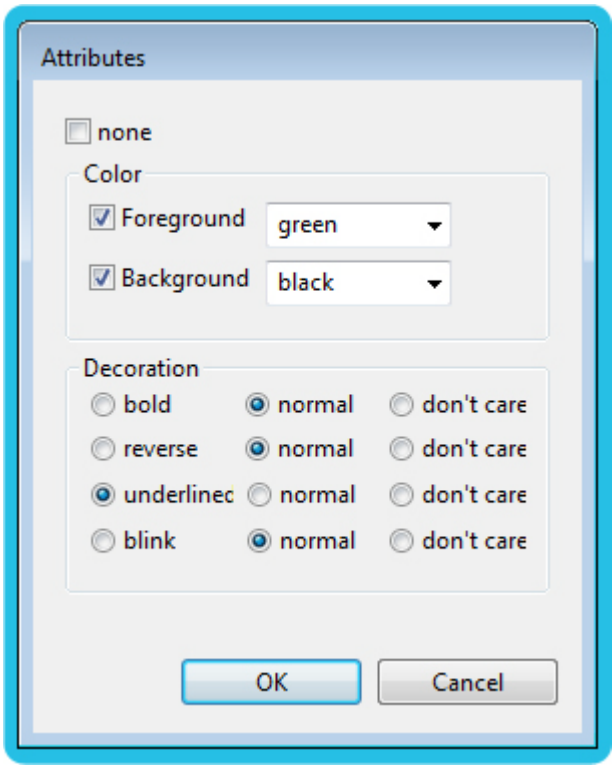


Figure 2 - 455: Field/Text extraction rule - Field attributes property edition

When applied, the two *Field/Text* rules create two XML elements: one of `static` type and one of `field` type:

```
<block background="black" column="16" foreground="green" hasFocus="false" line="10"
name="" size="16" type="static">New label</block>
<block background="black" column="52" foreground="green" hasFocus="false" line="10"
name="__field_c52_l10" size="10" type="field">new field content</block>
```

Figure 2 - 456: Field/Text extraction rule - Resulting XML

These elements appears in the green screen blocks:

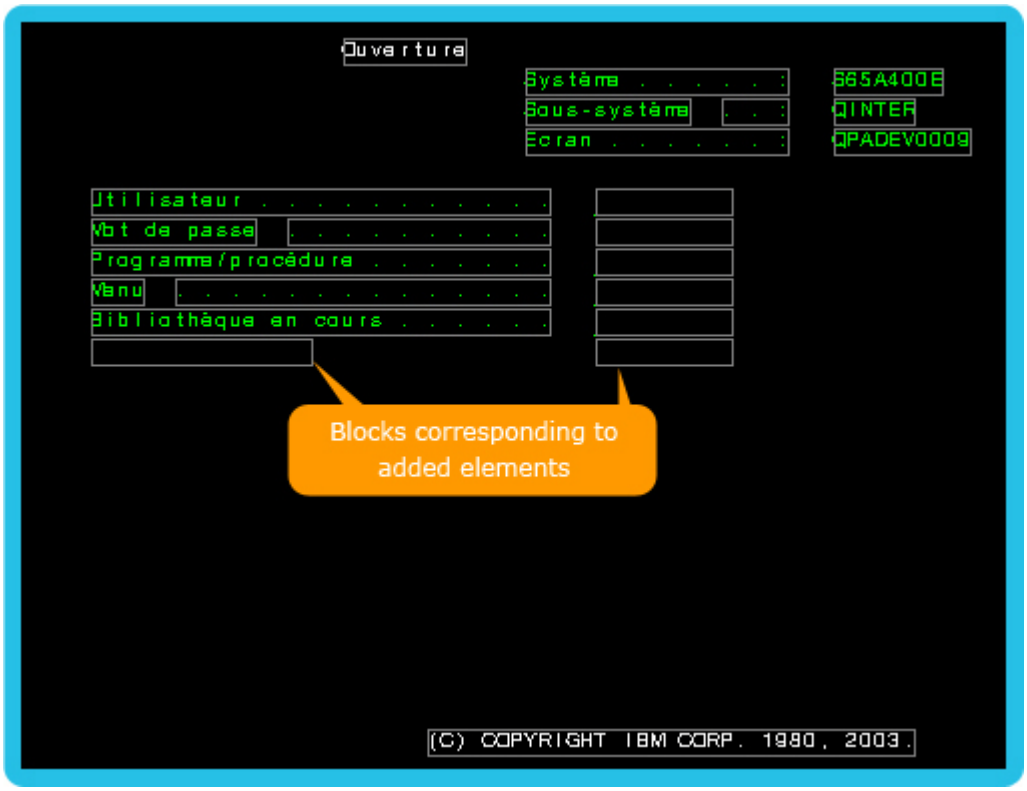


Figure 2 - 457: Fiedl/text extraction rule - Legacy screen with rules

After XSL transformation, thanks to XSL templates, they appear webized:

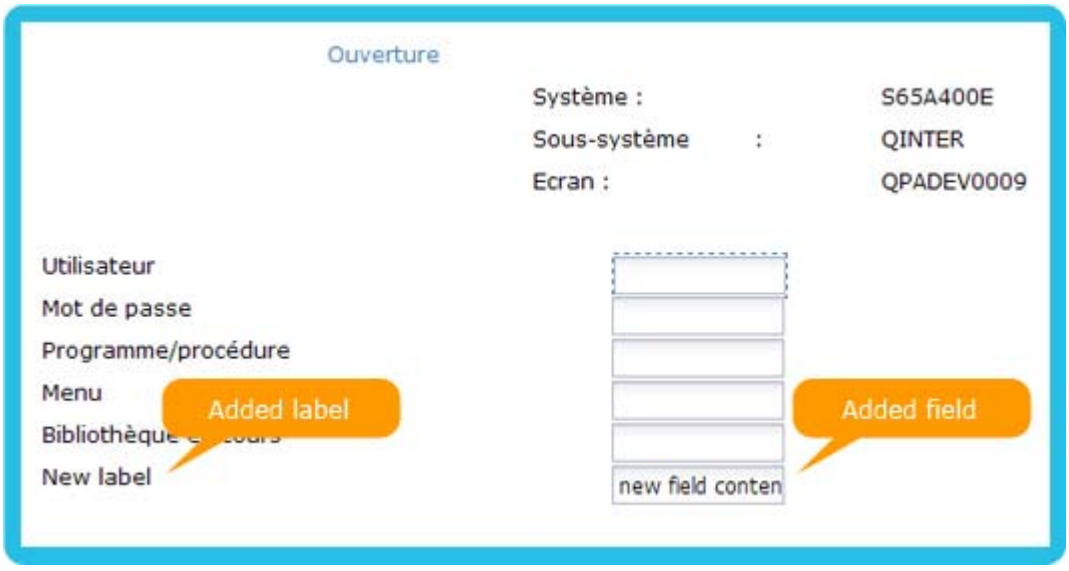


Figure 2 - 458: Fiedl/text extraction rule - Webized page with rules

Fields for VT emulators 

Object Description

Finds and marks blocks as being of `field` type.

The *Fields for VT emulators* extraction rule applies to VT220 integration only. It is used to recognize fields in VT emulator.

In fact, unlike in IBM mainframes, VT screens including fields are not typed as fields. Field detection is based on display attributes (background color, foreground color, etc.). When matching blocks are found, they are added to the XML document as `field` type XML elements.

Note:XML elements of the `field` type are handled by the `field` XSL template described in the `field.xsl` file. To change the way fields are displayed in the HTML page, edit this file.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

DATE 

OBJECT DESCRIPTION

Handles fields of `date` type.

The *Date* extraction rule applies to Legacy Publishing only. It is intended to detect a field as a date and to manage it as such. As a consequence, blocks containing a date are extracted as `date` blocks with relevant attributes, not as `field` blocks containing a date as a string.

Note: This rule adds a `date` type XML element to the XML document. It is associated with the `date` XSL template, described in the `date.xsl` file. On the webized page, a calendar pops-up when clicking on the date, to select the appropriate value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Am/pm marker	boolean	configuration	Includes the `am_pm_marker` attribute.
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Day in month	boolean	configuration	Includes the `day` attribute. The day is included on two digits, from `01` to `31`.
Day in week, long name	boolean	configuration	Includes the `day_in_week_long` attribute. The day is included in full spelling.
Day in week, short name	boolean	configuration	Includes the `day_in_week_short` attribute. The day is included in short (`Mon`, `Tue`, `Wed`, etc.).
Format	String	configuration	Defines the date format. This property defines the pattern to be complied with for the field to be recognized as a date (for example, `dd/mm/yyyy`). For more information on usable symbols, see Appendix "*Date format - Usable symbols*".
Hour in am/pm (0-11)	boolean	configuration	Includes the `hour_in_day_0_to_11` attribute.
Hour in am/pm (1-12)	boolean	configuration	Includes the `hour_in_day_1_to_12` attribute.

Property	Type	Category	Description
Hour in day (0-23)	boolean	configuration	Includes the `hour_in_day_0_to_23` attribute.
Hour in day (1-24)	boolean	configuration	Includes the `hour_in_day_1_to_24` attribute.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Language	String	configuration	Defines the language of the date (`ISO 639` compliant).
Millisecond	boolean	configuration	Includes the `milliseconds` attribute.
Minute in hour	boolean	configuration	Includes the `minutes` attribute.
Month in year	boolean	configuration	Includes the `month` attribute. The month is included on two digits, from `01` to `12`.
Month in year, long name	boolean	configuration	Includes the `month_name_long` attribute. The month is included in full spelling.
Month in year, short name	boolean	configuration	Includes the `month_name_short` attribute. The month is included in short (`Jan`, `Feb`, `Mar`, `Apr`, etc.).
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Second in minute	boolean	configuration	Includes the `seconds` attribute.
Time zone (General time zone)	boolean	configuration	Includes the `time_zone_text` attribute.
Time zone (RFC 822 time zone)	boolean	configuration	Includes the `time_zone_number` attribute. The time zone is included in accordance with the `RFC-822` standard.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Week in month	boolean	configuration	Includes the `week_in_month` attribute. The week of the month is included on two digits.
Week in year	boolean	configuration	Includes the `week_in_year` attribute. The week of the year is included on two digits, from `01` to `52`.
Year	boolean	configuration	Includes the `year` attribute. The year is included on four digits.

EXAMPLES

If we consider the following legacy screen:

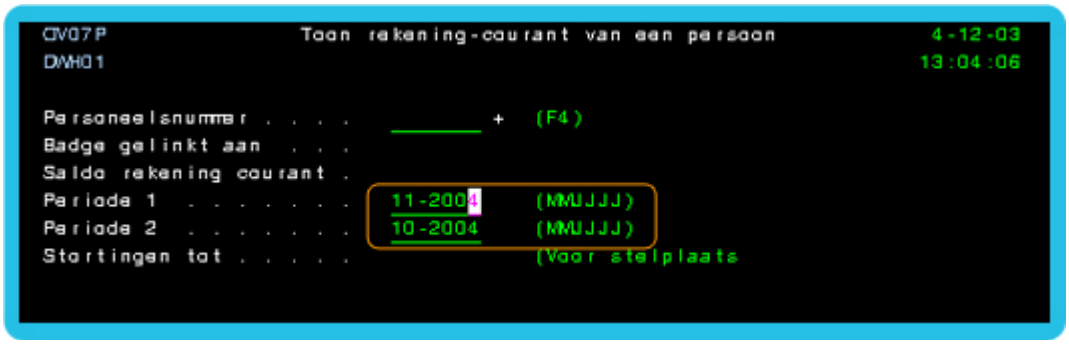


Figure 2 - 459: Date extraction rule - Legacy screen

We can notice that two fields are dates in the screen: *Periode1* and *Periode2*. They are symbolised by the `(MMJJJJ)` label on the fields right.

Without a *Date* extraction rule, the XML resulting from this screen is as follows:

```
<block background="white" line="6" name="untitled" type="static">Periode 1</block>
<block background="white" line="6" name="untitled" type="static"/>
<block background="green" line="6" name="__field_c28_l6" numeric="true" size="7" type="field">11-2004</block>
<block background="green" line="6" name="untitled" type="static">(MMJJJJ)</block>
<block background="white" line="7" name="untitled" type="static">Periode 2</block>
<block background="white" line="7" name="untitled" type="static"/>
<block background="green" line="7" name="__field_c28_l7" numeric="true" size="7" type="field">10-2004</block>
<block background="green" line="7" name="untitled" type="static">(MMJJJJ)</block>
```

Figure 2 - 460: Date extraction rule - Resulting XML without rule

When webized, these elements appear as simple fields:

Figure 2 - 461: Date extraction rule - Webized page without rule

In this example, a *Date* extraction rule is created with the following parameters:

```
Date [
   type="field"
   format="MM-yyyy"
   month in year=true
   year=true
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

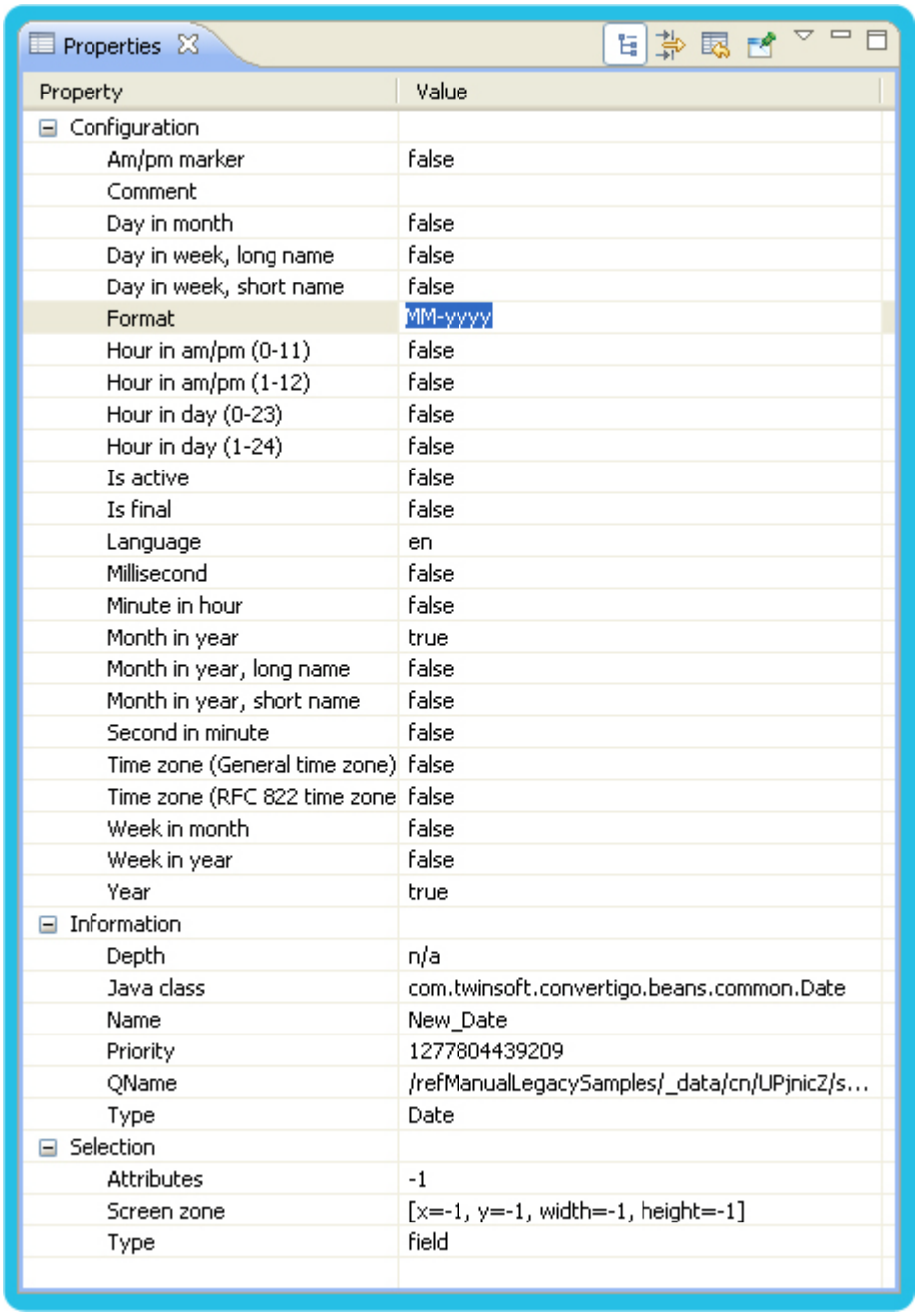


Figure 2 - 462: Date extraction rule - Configuration example

When the rule is executed, the resulting XML includes the `date` type XML elements, with `pattern` attribute and others attributes corresponding to date parts selected by the rule properties:

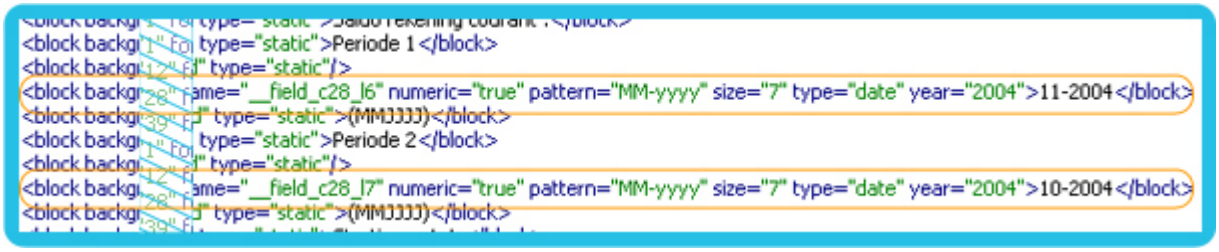


Figure 2 - 463: Date extraction rule - Resulting XML with rule

After XSL transformation, thanks to `date` XSL template, it appears webized:

Figure 2 - 464: Date extraction rule - Webized page with rule

PANEL 

OBJECT DESCRIPTION

Handles screen panels.

The *Panel* extraction rule allows the grouping of a set of blocks within a panel. It creates a `panel` type element surrounding these blocks.

The extraction rule is based on remarkable character patterns in order to determine:

- the panel position,
- blocks being a part of it.

To determine the panel position, the rule can also use screen attributes. This is mostly the case for 5250 applications, where windows can be displayed with no border characters, but with colored borders.

Note: XML elements of the `panel` type are handled by the `panel` XSL template described in the `panel.xsl` file. To change the way panels are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Minimum number of sides	int	configuration	Defines the minimum number of sides for a panel to be detected. By default, the rule needs 4 sides for a panel to be detected. Sometimes, applications display text in borders (Window description for example). Borders might therefore not match exactly the rule parameters, so you can reduce the number of sides and still have the panel detected.
Panel bottom	String	configuration	Defines the bottom character (zone 7, see *"Panel zone description"* table below).
Panel left	String	configuration	Defines the left character (zone 4, see *"Panel zone description"* table below).
Panel lower left	String	configuration	Defines the lower left character (zone 6, see *"Panel zone description"* table below).
Panel lower right	String	configuration	Defines the lower right character (zone 8, see *"Panel zone description"* table below).
Panel right	String	configuration	Defines the right character (zone 5, see *"Panel zone description"* table below).
Panel top	String	configuration	Defines the top character (zone 2, see *"Panel zone description"* table below).
Panel upper left	String	configuration	Defines the upper left character (zone 1, see *"Panel zone description"* table below).
Panel upper right	String	configuration	Defines the upper right character (zone 3, see *"Panel zone description"* table below).
Remove blocks in borders	boolean	configuration	If checked, all text blocks found in the panel border are deleted. In some applications, windows are detected through their colored border, but there can be also text in borders. As a result, the webized screen is of poor quality. Set this parameter to `true` to remove text in borders.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.

Property	Type	Category	Description
Title attribute	int	selection	Defines the attributes of the title of the panel. In some applications, a panel can contain a text in the border that corresponds to the window title. This property allows to configure the presentation attributes of this title to differentiate it from the rest of the panel border. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

Usually, the *Panel* rule works by detecting the panel on the screen. You can also specify where the panel must be searched, by using screen zone parameters.

A panel is defined by the following zones

Table 2 - 2: Panel zone description

Zone	Description
1	Character defining the panel upper left corner.
2	Character sequence defining the panel upper part.
3	Character defining the panel upper right corner.
4	Character sequence defining the panel left part.
5	Character sequence defining the panel right part.
6	Character defining the panel bottom left corner.
7	Character sequence defining the panel bottom part.
8	Character defining the panel bottom right corner.
9	Blocks that will be grouped within the panel.

Each zone of the panel is described by a rule parameter. These parameters correspond to the

characters defining the zone.

For example, if we consider the following panel type:

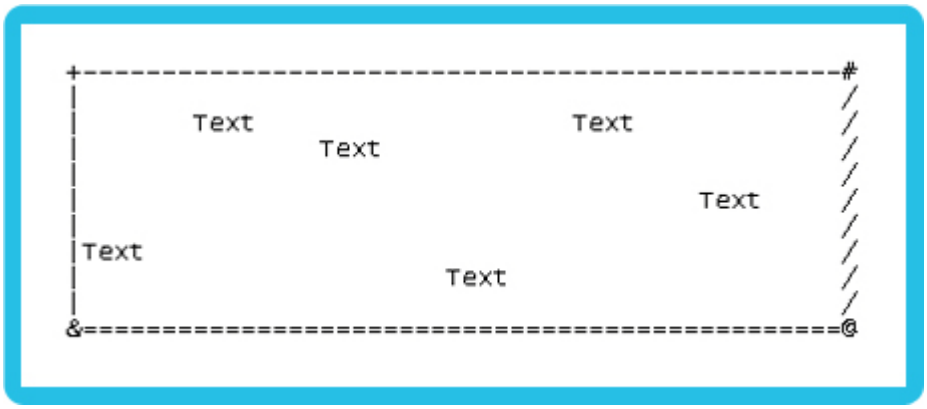


Figure 2 - 465: Panel type

The panel zones are defined as follows:

Table 2 - 3: Example panel - Zone parameter value

Zone	Value
1	+
2	-
3	#
4	|
5	/
6	&
7	=
8	@

Let’s configure this rule on a sample. If we consider the following legacy screen:

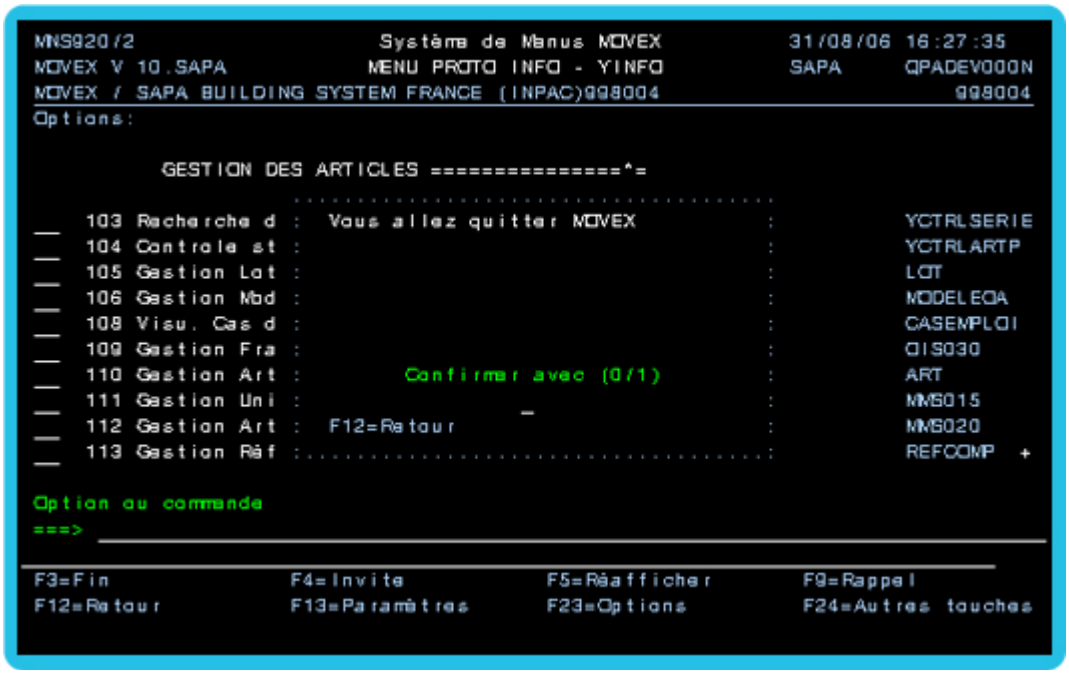


Figure 2 - 466: Panel extraction rule - Legacy screen

We can notice the panel that is displayed in the middle of the screen, over the screen content.

Without the rule, the resulting XML is as follows:

```
<block background="black" column="5" foreground="black" line="6" name="untitled" type="static">102</block>
<block background="black" column="21" foreground="blue" line="6" name="untitled" type="static">.....</blc
<block background="black" column="5" foreground="white" line="7" name="untitled" type="static">103</block>
<block background="black" column="9" foreground="white" line="7" name="untitled" type="static">Recherche d</block>
<block background="black" column="21" foreground="blue" line="7" name="untitled" type="static">:</block>
<block background="black" column="24" foreground="white" line="7" name="untitled" type="static">Vous allez quitter MOVEX</block>
<block background="black" column="58" foreground="blue" line="7" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="7" name="untitled" type="static">YCTRLSERIE</block>
<block background="black" column="5" foreground="white" line="8" name="untitled" type="static">104</block>
<block background="black" column="9" foreground="white" line="8" name="untitled" type="static">Controle st</block>
<block background="black" column="21" foreground="blue" line="8" name="untitled" type="static">:</block>
<block background="black" column="58" foreground="blue" line="8" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="8" name="untitled" type="static">YCTRLARTP </block>
<block background="black" column="5" foreground="white" line="9" name="untitled" type="static">105</block>
<block background="black" column="9" foreground="white" line="9" name="untitled" type="static">Gestion Lot</block>
<block background="black" column="21" foreground="blue" line="9" name="untitled" type="static">:</block>
<block background="black" column="58" foreground="blue" line="9" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="9" name="untitled" type="static">LOT</block>
<block background="black" column="5" foreground="white" line="10" name="untitled" type="static">106</block>
<block background="black" column="9" foreground="white" line="10" name="untitled" type="static">Gestion Mod</block>
<block background="black" column="21" foreground="blue" line="10" name="untitled" type="static">:</block>
<block background="black" column="58" foreground="blue" line="10" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="10" name="untitled" type="static">MODELEOA</block>
<block background="black" column="5" foreground="white" line="11" name="untitled" type="static">108</block>
<block background="black" column="9" foreground="white" line="11" name="untitled" type="static">Visu. Cas d</block>
<block background="black" column="21" foreground="blue" line="11" name="untitled" type="static">:</block>
<block background="black" column="58" foreground="blue" line="11" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="11" name="untitled" type="static">CASEMPLOI </block>
<block background="black" column="5" foreground="white" line="12" name="untitled" type="static">109</block>
<block background="black" column="9" foreground="white" line="12" name="untitled" type="static">Gestion Fra</block>
<block background="black" column="21" foreground="blue" line="12" name="untitled" type="static">:</block>
<block background="black" column="58" foreground="blue" line="12" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="12" name="untitled" type="static">OIS030</block>
<block background="black" column="5" foreground="white" line="13" name="untitled" type="static">110</block>
<block background="black" column="9" foreground="white" line="13" name="untitled" type="static">Gestion Art</block>
<block background="black" column="21" foreground="blue" line="13" name="untitled" type="static">:</block>
<block background="black" column="30" foreground="green" line="13" name="untitled" type="static">Confirmer avec (0/1)</block>
<block background="black" column="58" foreground="blue" line="13" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="blue" line="13" name="untitled" type="static">ART</block>
<block background="black" column="5" foreground="white" line="14" name="untitled" type="static">111</block>
<block background="black" column="9" foreground="white" line="14" name="untitled" type="static">Gestion Uni</block>
<block background="black" column="21" foreground="blue" line="14" name="untitled" type="static">:</block>
```

Figure 2 - 467: Panel extraction rule - Resulting XML without rule

After XSL transformation, it appears webized:

Figure 2 - 468: Panel extraction rule - Webized page without rule

In this example, a *Panel* extraction rule is created with the following parameters:

```
Panel [
   minimum number of sides=4
   panel bottom="."
   panel left=":"
   panel lower left=":"
   panel lower right=":"
   panel right=":"
   panel top="."
   panel upper left="."
   panel upper right="."
   remove blocks in border=true
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

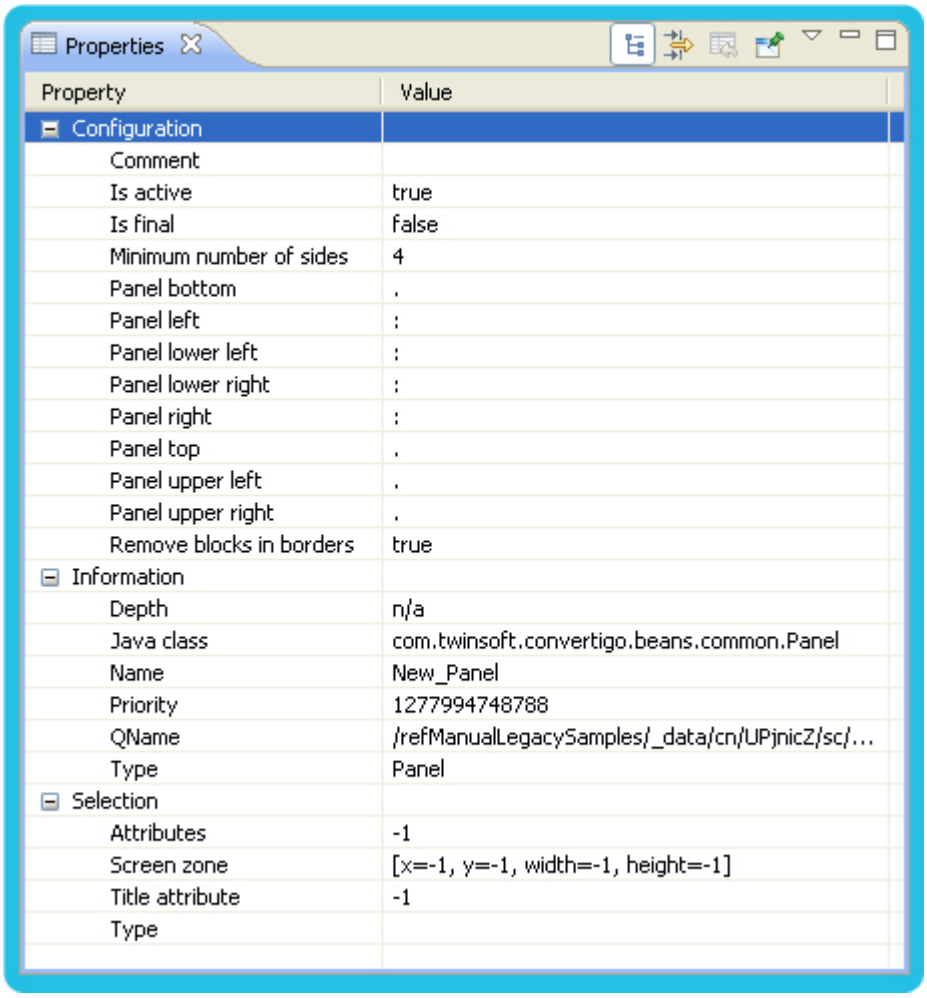


Figure 2 - 469: Panel extraction rule - Configuration example

When the rule is executed, the resulting XML includes the detected `panel` type element surrounding matching blocks:

```
<block background="black" column="5" foreground="black" line="6" name="untitled" type="static">102</block>
<block background="black" column="5" foreground="white" line="7" name="untitled" type="static">103</block>
<block background="black" column="9" foreground="white" line="7" name="untitled" type="static">Recherche d</block>
<block background="black" bottomTitle="" column="21" foreground="black" height="10" line="6" name="New_Panel" topTitle="" type="panel" width="37">
  <block background="black" column="24" foreground="white" line="7" name="untitled" type="static">Vous allez quitter MOVEX</block>
  <block background="black" column="30" foreground="green" line="13" name="untitled" type="static">Confirmer avec (0/1)</block>
  <block background="black" column="39" foreground="white" hasFocus="true" line="14" name="__field_c39_l14" numeric="true" size="1" type="field"/>
  <block action="KEY_PF1" background="black" column="24" data="" foreground="blue" line="15" name="untitled" type="keyword">F12=Retour</block>
</block>
<block background="black" column="69" foreground="blue" line="7" name="untitled" type="static">YCTRLSERIE</block>
<block background="black" column="5" foreground="white" line="8" name="untitled" type="static">104</block>
```

Figure 2 - 470: Panel extraction rule - Resulting XML with rule

After XSL transformation, thanks to `keyword` XSL template, it appears webized:

Figure 2 - 471: Panel extraction rule - Webized page with rule

SEPARATOR 

OBJECT DESCRIPTION

Defines how separators are materialized in legacy screens.

The *Separator* extraction rule defines character strings to be considered as separators. It searches the screen for strings made of the same character, like series of "minus" signs ("----"), or of underscores ("____").

Characters considered as separators are listed in the **Separators** extraction rule property. The rule matches strings only if they contain at least the number of occurrences (set as value of the **Minimum number of occurrences** property) in one of the specified delimiters.

Notes:

- The separator element has only one new attribute, `width`, which represents the length of the separator, that is to say the number of separator characters in the separator string.
- XML elements of the `separator` type are handled by the `separator` XSL template described in the `separator.xsl` file. To change the way separators are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Minimum number of occurrences	int	configuration	Defines the minimum number of occurrences of the character in a string to consider it a separator.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Separators	String	configuration	Defines the list of separator characters. List of all characters which, once chained, are to be considered as a separator.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
XML tag name	String	configuration	Defines the tag name of the separator block

EXAMPLES

If we consider the following legacy screen:

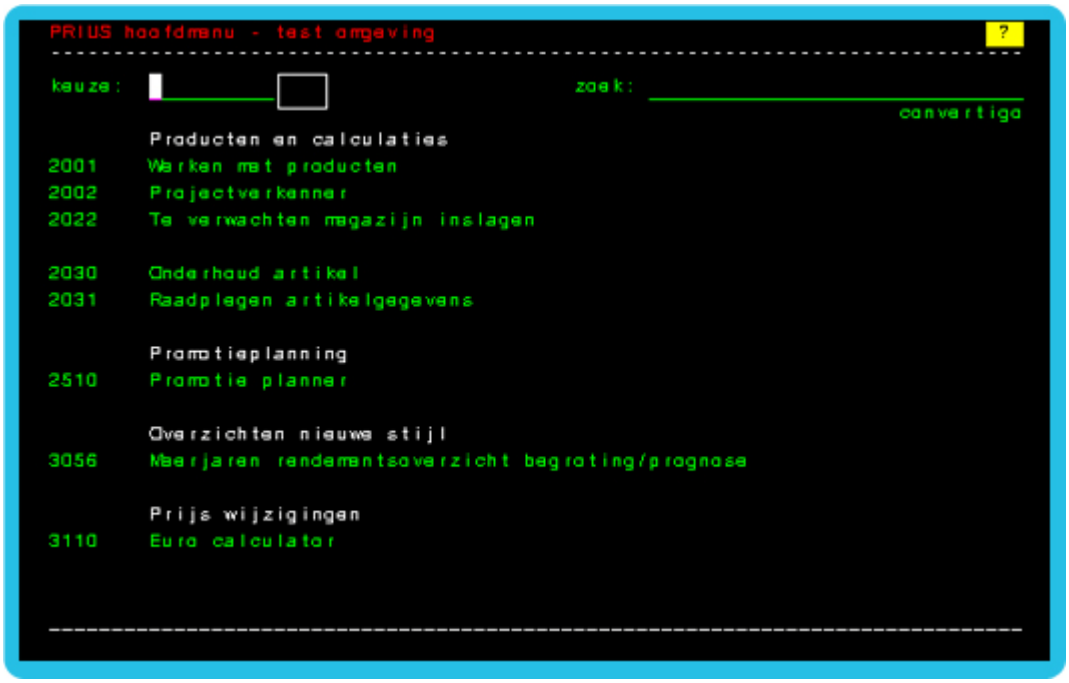


Figure 2 - 472: Separator extraction rule - Legacy screen

Without the rule, the resulting XML is as follows:

```
<blocks page-number="0">
  <block background="black" column="1" foreground="red" line="0" name="untitled" type="static">PRIUS hoofdmenu -</block>
  <block background="black" column="19" foreground="red" line="0" name="untitled" type="static">test omgeving</block>
  <block background="yellow" column="77" foreground="black" line="0" name="untitled" type="static">? </block>
  <block background="black" column="1" foreground="white" line="1" name="untitled" type="static">
  <block background="black" column="1" foreground="green" line="2" name="untitled" type="static">keuze: </block>
  <block background="black" column="9" foreground="green" hasFocus="true" line="2" name="__field_c9_l2" size="10" type="field"/>
  <block background="black" column="43" foreground="green" line="2" name="untitled" type="static">zoek: </block>
  <block background="black" column="49" foreground="green" line="2" name="__field_c49_l2" size="30" type="field"/>
  <block background="black" column="69" foreground="green" line="3" name="untitled" type="static">convertigo</block>
  <block background="black" column="1" foreground="black" line="4" name="untitled" type="static">2000</block>
  <block background="black" column="9" foreground="white" line="4" name="untitled" type="static">Producten en calculaties</block>
  <block background="black" column="1" foreground="green" line="5" name="untitled" type="static">2001 </block>
  <block background="black" column="9" foreground="green" line="5" name="untitled" type="static">Werken met producten</block>
  <block background="black" column="1" foreground="green" line="6" name="untitled" type="static">2002</block>
  <block background="black" column="9" foreground="green" line="6" name="untitled" type="static">Projectverkenner</block>
  <block background="black" column="1" foreground="green" line="7" name="untitled" type="static">2022</block>
  <block background="black" column="9" foreground="green" line="7" name="untitled" type="static">Te verwachten magazijn inslagen</block>
  <block background="black" column="1" foreground="green" line="9" name="untitled" type="static">2030</block>
  <block background="black" column="9" foreground="green" line="9" name="untitled" type="static">Onderhoud artikel</block>
  <block background="black" column="1" foreground="green" line="10" name="untitled" type="static">2031 </block>
  <block background="black" column="9" foreground="green" line="10" name="untitled" type="static">Raadplegen artikelgegevens</block>
  <block background="black" column="1" foreground="black" line="12" name="untitled" type="static">2500</block>
  <block background="black" column="9" foreground="white" line="12" name="untitled" type="static">Promotieplanning</block>
  <block background="black" column="1" foreground="green" line="13" name="untitled" type="static">2510</block>
  <block background="black" column="9" foreground="green" line="13" name="untitled" type="static">Promotie planner</block>
  <block background="black" column="1" foreground="black" line="15" name="untitled" type="static">3000</block>
  <block background="black" column="9" foreground="white" line="15" name="untitled" type="static">Overzichten nieuwe stijl</block>
  <block background="black" column="1" foreground="green" line="16" name="untitled" type="static">3056</block>
  <block background="black" column="9" foreground="green" line="16" name="untitled" type="static">Meerjaren rendementsoverzicht begroting/p
  <block background="black" column="1" foreground="black" line="18" name="untitled" type="static">3100</block>
  <block background="black" column="9" foreground="white" line="18" name="untitled" type="static">Prijs wijzigingen</block>
  <block background="black" column="1" foreground="green" line="19" name="untitled" type="static">3110</block>
  <block background="black" column="9" foreground="green" line="19" name="untitled" type="static">Euro calculator</block>
  <block background="black" column="1" foreground="white" line="22" name="untitled" type="static">
  <block background="blue" column="2" foreground="black" line="23" name="untitled" type="static">Afsluiten </block>
  <block background="blue" column="17" foreground="black" line="23" name="untitled" type="static">Actie</block>
  <block background="black" column="72" foreground="blue" line="23" name="untitled" type="static">ããã</block>
</blocks>
</document>
```

Figure 2 - 473: Separator extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

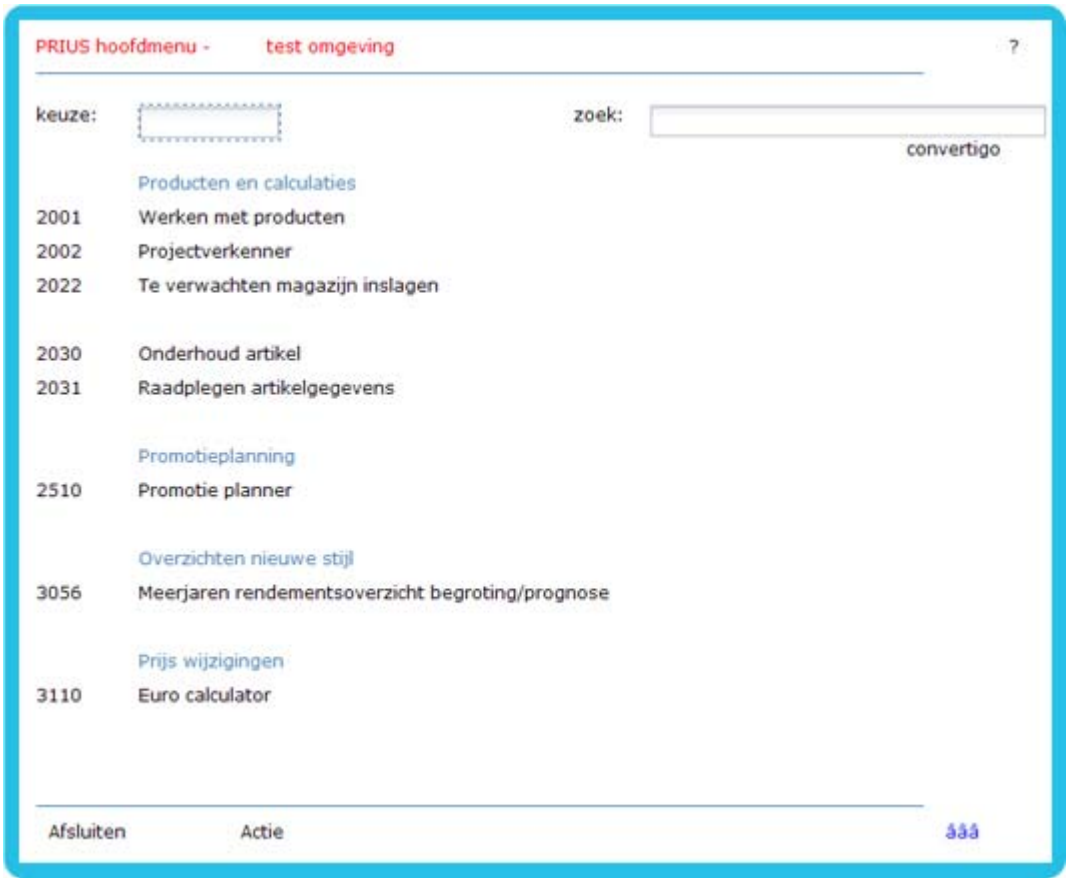


Figure 2 - 474: Separator extraction rule - Webized page without rule

In this example, we want to transform the two dashes lines on top and bottom of the screen into separator. A *Separator* extraction rule is created with the following parameters:

```
Separator [
   attributes=[foreground="white" background="black"]
   separators="¯_"
   minimum number of occurrences=10
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

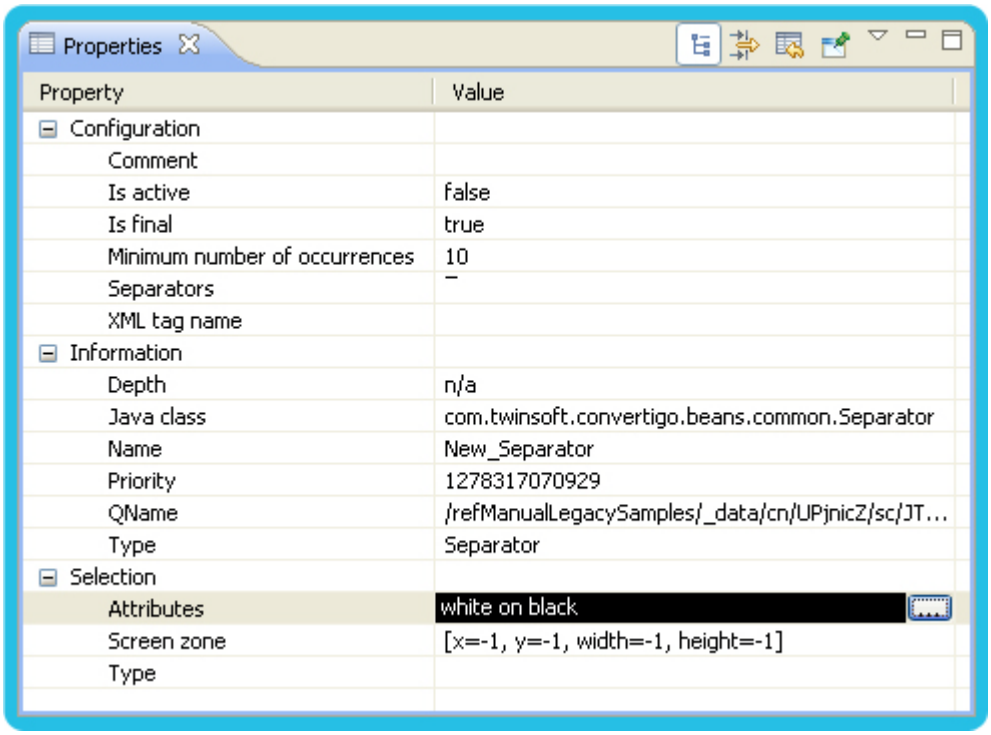


Figure 2 - 475: Separator extraction rule - Configuration example

When the rule is executed, the matching blocks are removed from the resulting XML which includes the two `separator` type blocks instead:

```
<blocks page-number="0">
  <block background="black" column="1" foreground="red" line="0" name="untitled" type="static">PRIUS hoofdmenu -</block>
  <block background="black" column="19" foreground="red" line="0" name="untitled" type="static">test omgeving</block>
  <block background="yellow" column="77" foreground="black" line="0" name="untitled" type="static">? </block>
  <block background="black" column="1" foreground="white" line="1" name="untitled" type="separator" width="78"/>
  <block background="black" column="1" foreground="green" line="2" name="untitled" type="static">keuze:</block>
  <block background="black" column="9" foreground="green" hasFocus="true" line="2" name="__field_c9_l2" size="10" type="field"/>
  <block background="black" column="43" foreground="green" line="2" name="untitled" type="static">zoek:</block>
  <block background="black" column="49" foreground="green" line="2" name="__field_c49_l2" size="30" type="field"/>
  <block background="black" column="69" foreground="green" line="3" name="untitled" type="static">convertigo</block>
  <block background="black" column="1" foreground="black" line="4" name="untitled" type="static">2000</block>
  <block background="black" column="9" foreground="white" line="4" name="untitled" type="static">Producten en calculaties</block>
  <block background="black" column="1" foreground="green" line="5" name="untitled" type="static">2001</block>
  <block background="black" column="9" foreground="green" line="5" name="untitled" type="static">Werken met producten</block>
  <block background="black" column="1" foreground="green" line="6" name="untitled" type="static">2002</block>
  <block background="black" column="9" foreground="green" line="6" name="untitled" type="static">Projectverkenner</block>
  <block background="black" column="1" foreground="green" line="7" name="untitled" type="static">2022</block>
  <block background="black" column="9" foreground="green" line="7" name="untitled" type="static">Te verwachten magazijn inslagen</block>
  <block background="black" column="1" foreground="green" line="9" name="untitled" type="static">2030</block>
  <block background="black" column="9" foreground="green" line="9" name="untitled" type="static">Onderhoud artikel</block>
  <block background="black" column="1" foreground="green" line="10" name="untitled" type="static">2031</block>
  <block background="black" column="9" foreground="green" line="10" name="untitled" type="static">Raadplegen artikelgegevens</block>
  <block background="black" column="1" foreground="black" line="12" name="untitled" type="static">2500</block>
  <block background="black" column="9" foreground="white" line="12" name="untitled" type="static">Promotieplanning</block>
  <block background="black" column="1" foreground="green" line="13" name="untitled" type="static">2510</block>
  <block background="black" column="9" foreground="green" line="13" name="untitled" type="static">Promotie planner</block>
  <block background="black" column="1" foreground="black" line="15" name="untitled" type="static">3000</block>
  <block background="black" column="9" foreground="white" line="15" name="untitled" type="static">Overzichten nieuwe stijl</block>
  <block background="black" column="1" foreground="green" line="16" name="untitled" type="static">3056</block>
  <block background="black" column="9" foreground="green" line="16" name="untitled" type="static">Meerjaren rendementsoverzicht begroting/pr
  <block background="black" column="1" foreground="black" line="18" name="untitled" type="static">3100</block>
  <block background="black" column="9" foreground="white" line="18" name="untitled" type="static">Prijs wijzigingen</block>
  <block background="black" column="1" foreground="green" line="19" name="untitled" type="static">3110</block>
  <block background="black" column="9" foreground="green" line="19" name="untitled" type="static">Euro calculator</block>
  <block background="black" column="1" foreground="white" line="22" name="untitled" type="separator" width="78"/>
  <block background="blue" column="2" foreground="black" line="23" name="untitled" type="static">Afsluiten </block>
  <block background="blue" column="17" foreground="black" line="23" name="untitled" type="static">Actie</block>
  <block background="black" column="72" foreground="blue" line="23" name="untitled" type="static">ãâã</block>
</blocks>
```

Figure 2 - 476: Separator extraction rule - Resulting XML with rule

After XSL transformation, thanks to `separator` XSL template, the screen appears webized:

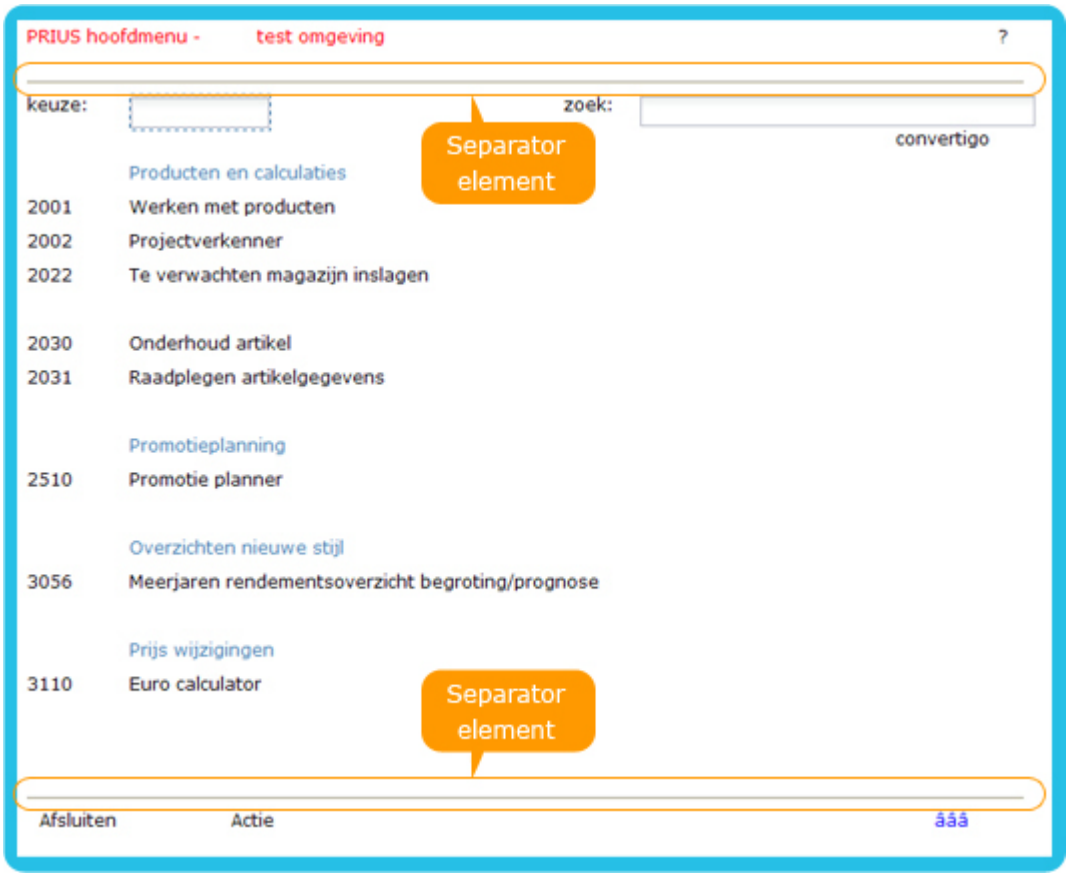


Figure 2 - 477: Separator extraction rule - Webized page with rule

Record (Legacy) 

Object Description

Extract data from screen as a record (set of structured data).

The *Record* extraction rule extracts legacy data as a structured set of complex XML elements sharing similarities. For example, a record list can be a list of names and addresses.

A record is defined through a starting and an ending block, each one characterized by a block type and an XML tag name. In short, a *Record* rule can be defined as the definition of a pair of start and end tags indicating where the record starts and ends.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
End separator XML tag name	String	configuration	Defines the XML tag name of the separator block ending the record.
End separator type	String	configuration	Defines the type of the separator block ending the record. The end separator default type is `separator`. This parameter can take either default (`field`, `static`, etc.) or user-defined types.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Multiple records	boolean	configuration	Defines whether the record must be defined on a "per page" basis. If set to `true`, one record is generated for each screen.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Start separator XML tag name	String	configuration	Defines the XML tag name of the separator block starting the record.
Start separator type	String	configuration	Defines type of the separator block starting the record. The start separator default type is `separator`. This parameter can take either default (`field`, `static`, etc.) or user-defined types.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
XML tag name	String	configuration	Defines the tag name of the XML record generated in output XML after extraction.

EXAMPLES

If we consider the following legacy screen:

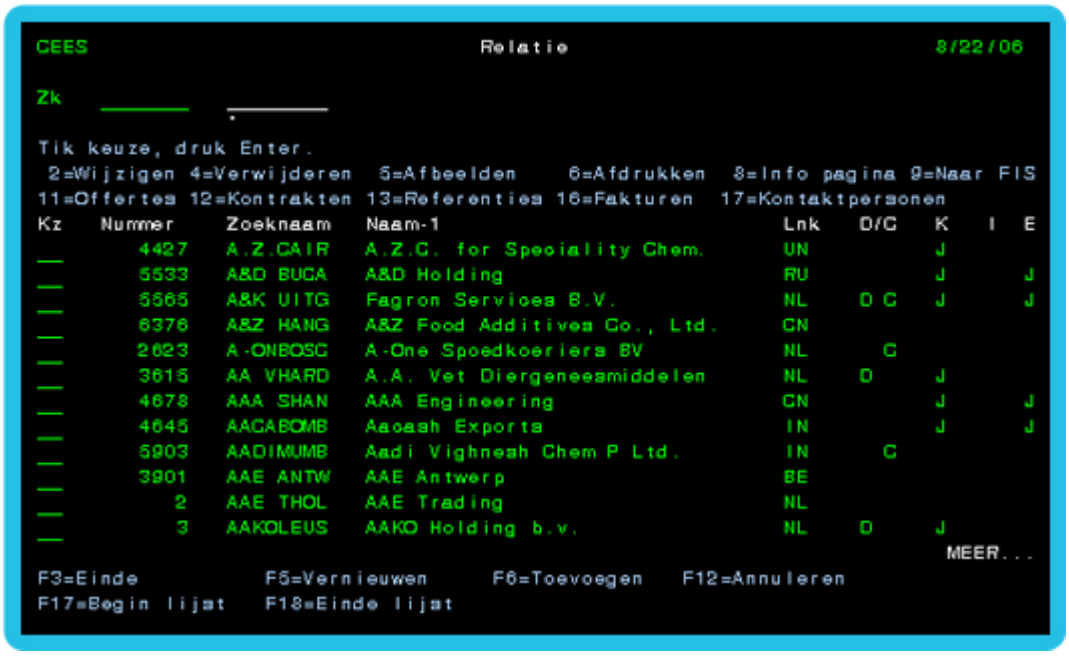


Figure 2 - 478: Record extraction rule - Legacy Screen

Several *Tag name* extraction rules and a *Removing of blocks* extraction rule have been set upt in order to name extracted data and remove the blocks that are not named. The XML resulting from this screen is as follows:

Figure 2 - 479: Record extraction rule - Resulting XML without rule

In this example, a *Record* rule is created with the following parameters:

```
Record [
   screen zone=[x=5, y=7, width=62, height=14]
   Start separator type="static"
   Start separator XML tag name="nummer"
   End separator type="static"
   End separator XML tag name="lnk"
```

```
  XML tag="record"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

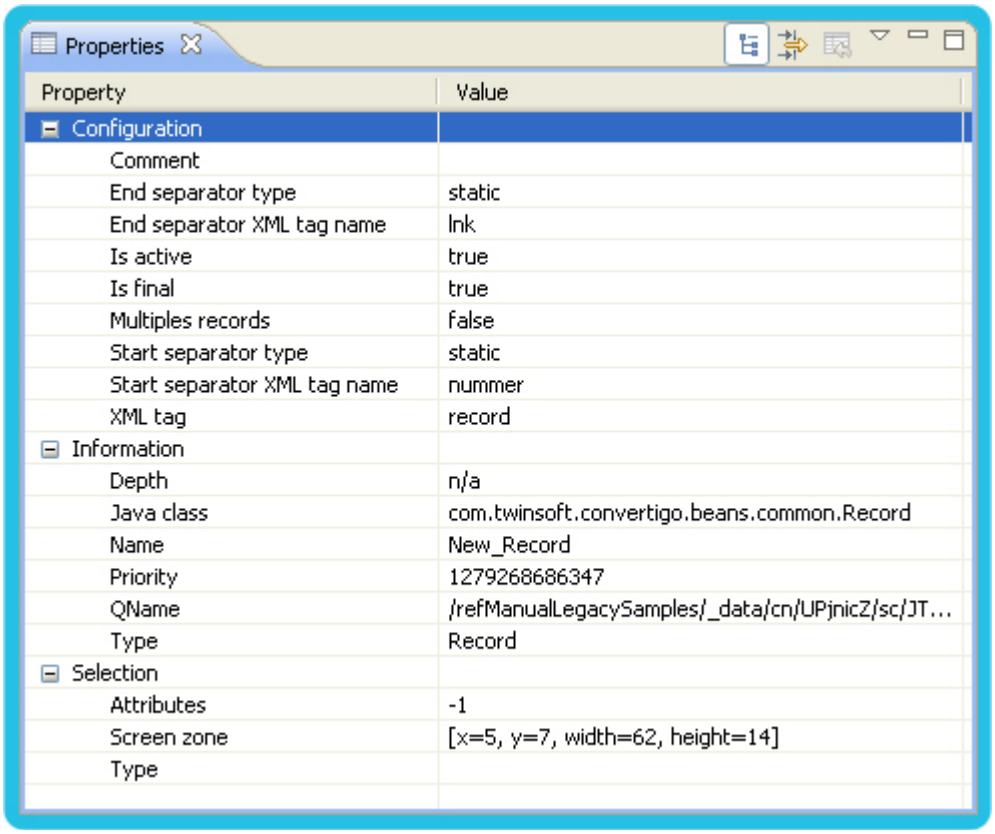


Figure 2 - 480: Record extraction rule - Configuration example

When the rule is executed, the resulting XML includes a `record` tag container for each set of data:

```
  <zoeknaam background="black" column="16" foreground="green" history="false" line="10" name="untitled" type="static">A&amp;amp;K UITG</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="10" name="untitled" type="static">Fagron Services B.V.</naam1>
</record>
<record background="black" column="9" foreground="black" line="11" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="11" name="untitled" type="static">6376</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="11" name="untitled" type="static">A&amp;amp;Z HANG</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="11" name="untitled" type="static">A&amp;amp;Z Food Additives Co., Ltd.</naam1>
</record>
<record background="black" column="9" foreground="black" line="12" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="12" name="untitled" type="static">2623</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="12" name="untitled" type="static">A-ONBOSC</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="12" name="untitled" type="static">A-One Spoedkoeriers BV</naam1>
</record>
<record background="black" column="9" foreground="black" line="13" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="13" name="untitled" type="static">3615</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="13" name="untitled" type="static">AA VHARD</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="13" name="untitled" type="static">A.A. Vet Diergeneesmiddelen</naam1>
</record>
<record background="black" column="9" foreground="black" line="14" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="14" name="untitled" type="static">4678</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="14" name="untitled" type="static">AAA SHAN</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="14" name="untitled" type="static">AAA Engineering</naam1>
</record>
<record background="black" column="9" foreground="black" line="15" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="15" name="untitled" type="static">4645</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="15" name="untitled" type="static">AACABOMB</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="15" name="untitled" type="static">Aacash Exports</naam1>
</record>
<record background="black" column="9" foreground="black" line="16" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="16" name="untitled" type="static">5903</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="16" name="untitled" type="static">AADIMUMB</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="16" name="untitled" type="static">Aadi Vighnesh Chem P Ltd.</naam1>
</record>
<record background="black" column="9" foreground="black" line="17" name="" type="record">
  <nummer background="black" column="9" foreground="green" history="false" line="17" name="untitled" type="static">3901</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="17" name="untitled" type="static">AAE ANTW</zoeknaam>
  <naam1 background="black" column="27" foreground="green" history="false" line="17" name="untitled" type="static">AAE Antwerp</naam1>
</record>
<record background="black" column="12" foreground="black" line="18" name="" type="record">
  <nummer background="black" column="12" foreground="green" history="false" line="18" name="untitled" type="static">2</nummer>
  <zoeknaam background="black" column="16" foreground="green" history="false" line="18" name="untitled" type="static">AAE THOL</zoeknaam>
```

Figure 2 - 481: Record extraction rule - Resulting XML with rule

This rule organizes extracted data, it is not meant to be displayed in webized screen.

TABLE (LEGACY) 

OBJECT DESCRIPTION

Extract data from screen as a table (set of structured data containing rows and columns).

The *Table* extraction rule structures screen areas including text into a table XML structure. The generated structure can be used:

- to represent arrays as graphic tables for webization projects, or
- as a structured table for data integration projects.

Prior to running a *Table* extraction rule, blocks must be properly merged or split so that blocks can be dispatched in the defined cells.

The *Table* extraction rule works following two modes:

- **Automatic mode** (set by default if no column description is provided): the rule automatically determines the columns of the array and dispatches data in columns depending on their position to the first line of the array.
- **Explicit mode**: the rule is based on the column description made by the Convertigo application programmer to distribute data as required in each column. Each block of the first line represents a column defined as follows:
- starting position = block starting column
- ending position = (starting position of next block) - 1.

If complex arrays are involved, it is recommended to explicitly describe array columns by editing the **Columns** parameter. This is the second mode.

The *Table* extraction rule is similar to the *Subfile* extraction rule but must be configured manually. It supports actions lines, and scrolling. With 5250 environments, it is preferable to use the *Subfile* rule if the application complies with CUA (IBM guide line for designing applications) because the *Subfile* rule is automatic and does not require a per-screen configuration.

For all other environments (3270, DKU, VT, etc.), use preferably the *Table* extraction rule. This rule allows to output structured arrayed data whatever the input data, provided that you configure it for a specific screen class.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Actions end pattern	String	configuration	Pattern that ends the block including actions. Used when actions are extracted from screen. Some actions lines start and end with patterns: for example `(A=Action B=Bound C=Create)`. In this case, the end pattern is the `")"` character.

Property	Type	Category	Description
Actions label separators	String	configuration	Defines the list of separator characters used to separate each action from its action label in the actions line. Used when actions are extracted from screen. In our example, this separator is the = character.
Actions separators	String	configuration	Defines the list of separator characters used in the actions line to separate two actions (usually blank space). Used when actions are extracted from screen.
Actions start pattern	String	configuration	Pattern that starts the block including actions. Used when actions are extracted from screen. Some actions lines start and end with patterns: for example `(A=Action B=Bound C=Create)`. In this case, the start pattern is the `"("` character.
Actions table	XMLVector	configuration	Defines actions associated with the selection column. Each action is described using the following items: • **Value**: value to be typed in by a user in the selection field (in the selection column) to perform a specific action. In our example, would be `A`. • **Key**: key used to validate the action. Usually, `KEY_ENTER` for 3270 and 5250 or `KEY_XMIT` for BULL DKU. • **Label**: label to be displayed to the user when clicking on the contextual menu. In our example, `Action`. Notes: • A new action can be added to the list using the blue keyboard icon. The actions defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • This property is used when no actions line is displayed on screen (mostly 3270 application). In other cases, actions can be automatically extracted from the screen by using the **Relative index of the actions line** parameter.
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".

Property	Type	Category	Description
Auto validate	boolean	configuration	Auto validates action triggered after an option has been chosen in the selection column. Set this to `true` if you want the action to be executed immediately after a user has clicked on an option of the contextual action menu. Usually set to `false` to enable users to click options on several lines before validating by `ENTER`.
Columns	XMLVector	configuration	Defines the list of columns of the table, with their header, position and width. This parameter explicitly describes the columns of the array. For each column, you have to describe the following elements: • **Label**: text of the title displayed (Webization) and XML tag name of the data (Data integration). The **Label** property supports the `"\"` character specifying a line break within the column title. • **Initial column**: starting position of the column (0 based). • **Final column**: ending position of the column (0 based). • **Line index**: for "folded" tables where a logical line of data is represented physically as several lines. Gives the index of the physical line starting from 0. Notes: • A new column can be added to the list using the blue keyboard icon. The columns defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • If no column is defined, the rule tries to automatically define the columns (see **Automatic mode** property description).
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Do not accumulate	boolean	configuration	Enables to limit to one page table data accumulation in a same tag (i.e. create one tag per page). Normally, when accumulating data as a table on several pages (**Return** property set to `accumulate` in a transaction's exit handler), data come in as new lines of the same table. In some specific cases, you will want to disable this feature, by setting this property to `true`.
Height resize	int	configuration	Defines a number of lines by which increase (positive value) or decrease (negative value) the table height. The webized height of a table is automatically computed from its number of lines. To enable the scroll feature, set this property to a number of lines smaller than the number of physical lines in the table by setting a negative value into this property.

Property	Type	Category	Description
Index of the selection column	int	configuration	Defines the selection (or action) column using its index in the column table. Equals to `0` in most of the cases. Set to `-1` if there is no selection column. Selection column definition is important as contextual menus will only appear if it is correctly defined.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Keep empty lines	boolean	configuration	Defines whether to keep empty lines from the table. Empty table lines are removed by default (property set to `false`. Set this property to `true` to keep them.
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.

Property	Type	Category	Description
Offset	int	configuration	Defines the table offset (in lines) used to shift the table position. After extraction, the table is displayed on the webized screen exactly where the data zone has been defined. In most cases, a title line appears above the table in addition to the table title line. You can then use the offset parameter to shift the table up and cover the redundant title line. Usually set to the number of lines of the title zone (2 in our example, to shift up the table by two lines). Please note that the **Offset** property supports negative values (moves the webized table down).
Relative index of the actions line	int	configuration	Defines where actions are in relation to the table. Set this parameter to relative line of the actions line from the top of the data zone. In our example, the index is `-4`. This enables the *Table* extraction rule automatically extract the actions table from the screen. Set this to `0` if no actions line is to be extracted.
Remove actions line	boolean	configuration	Defines whether to remove actions line or not from webized screen (leaving the actions as a popup when clicking a row in the table). If set to `true`, the actions line is removed from the webized screen and shown in selection fields contextual menu only.
Remove titles	boolean	configuration	Defines whether the title row of the table must be removed. When extracted, the XML table contains titles in the first row, which is useful for webization projects. But for data integration projects (web services), titles are usually unwanted. In this case, set the property to `true`.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
XML tag name	String	configuration	Defines the XML tag name for the generated block (of type "table"). The default XML Tag name (`table`) can be overridden using this property. Can be useful for data integration projects, for example to extract two tables from the same screen (by setting a different tag name for each table).

EXAMPLES

The `sample_documentation_CLI` project set in the context of the *"Starting with Convertigo Legacy Integrator"* Quick Guide contains a *Table* extraction rule called `ArticlesTable`.

The purpose of the `ArticlesTable` extraction rule is to extract a legacy screen table into a three-column XML table:

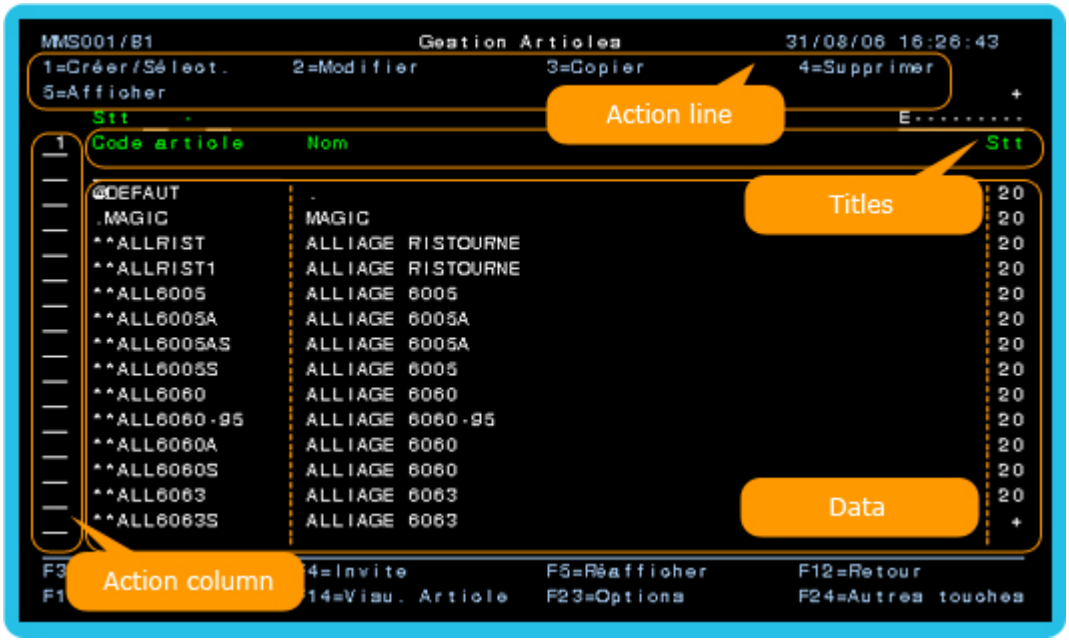


Figure 2 - 482: Table extraction rule - Legacy screen

A legacy screen table can be divided into several zones, described as follows:

Table 2 - 4: Table zone description

Table zone	Description
Actions line	Zone where actions can be launched from in a table. Displayed in the following format: `<StartPattern>` `<ActionCode><LabelSeparator><Label><Separator>` `<ActionCode><LabelSeparator><Label><Separator>...` `<EndPattern>`. Not always present.
Titles line	Zone where titles are displayed in a table.
Data	Zone where data is displayed in a table (corresponding to the **Screen zone** property).
Selection column	Zone where users input action code to be applied to associated line in the table.

Let's configure this rule on a sample. If we consider the previous legacy screen, we can notice the four table zones, which indicates that a *Table* extraction rule can be created.

Without the rule, the XML resulting from this screen is as follows:

Figure 2 - 483: Table extraction rule - Resulting XML without rule

After XSL transformation, it appears webized:

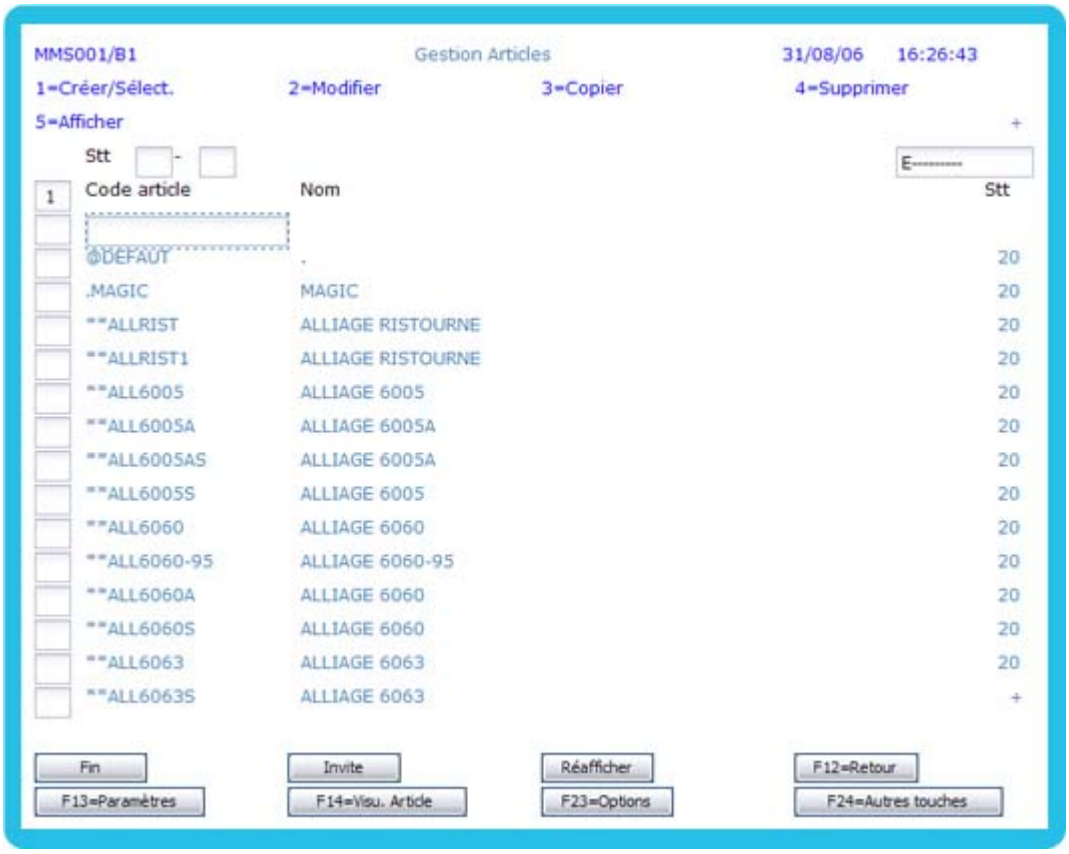


Figure 2 - 484: Table extraction rule - Webized page without rule

In this example, a *Table* extraction rule is created with the following parameters:

```
Table [
   screen zone=[x=1, y=5, width=78, height=15]
   columns={
      [Label="Selection" Initial column=1 Final Column=4 Line index=0],
      [Label="Code article" Initial column=5 Final Column=21
      Line index=0],
      [Label="Nom" Initial column=22 Final Column=75 Line index=0]
      [Label="Stt" Initial column=76 Final Column=78 Line index=0]
   }
   index of the selection column=0
   relative index of the actions line=-4
   actions label separator="="
   remove actions line=true
   remove titles=false
   auto validate=true
   height resize=-10
   offset=2
   xml tag name="articles"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

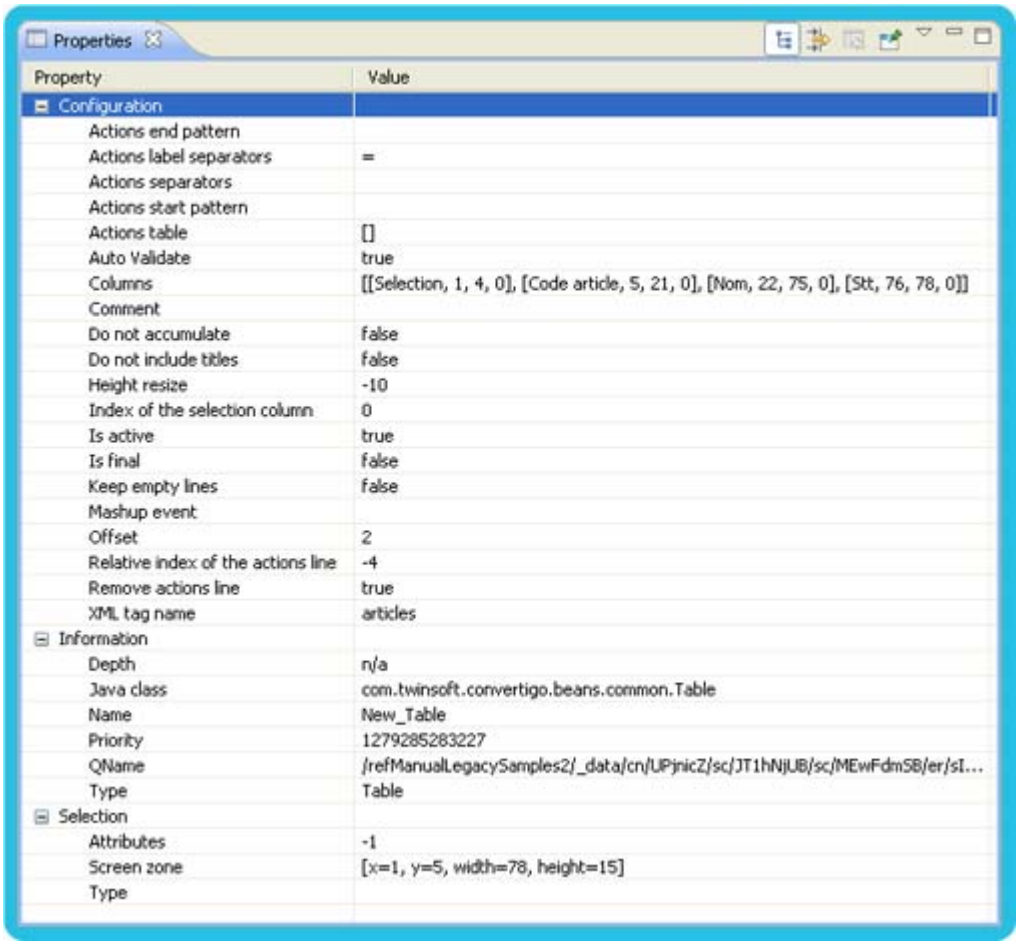


Figure 2 - 485: Table extraction rule - Configuration example

The **Columns** property describes the extracted columns of the table. It is edited in the **Columns definition of the table** editor:

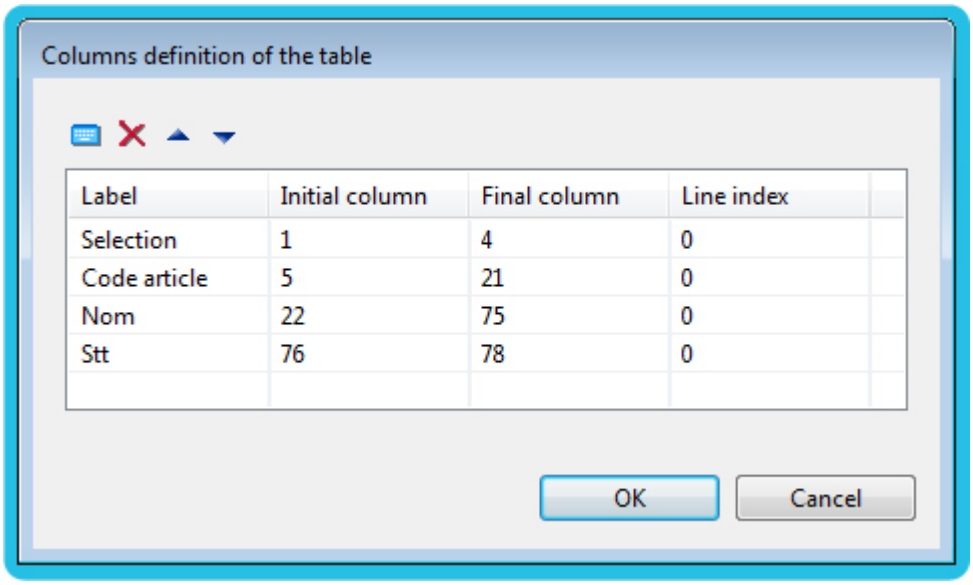


Figure 2 - 486: Table extraction rule - Columns property edition

In this sample case, the columns are named after the labels displayed on the screen `Code article`, `Nom` and `Stt` but they could also be changed in the **Label** property column of the editor.

The **XML tag name** sets the name of the table XML tag name in the resulting XML document (`articles`). The **Screen zone** property delineates the screen zone containing data to extract.

After data have been extracted, the corresponding XML is generated as follows:

Figure 2 - 487: Table extraction rule - Resulting XML

After XSL transformation, thanks to `table` XSL template, it appears webized:

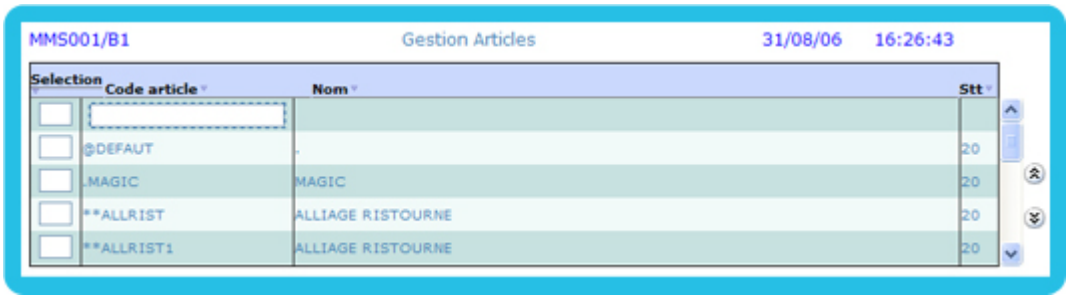


Figure 2 - 488: Table extraction rule - Webized page with rule

We can notice the scrollbar that is created by the XSL template thanks to the setting of **Resize**

property with a negative number (that decreases the table height).

BUTTON 

OBJECT DESCRIPTION

Adds a button on a legacy screen.

Unlike other rules, the *Button* extraction rule is not designed to extract data from the green screen, but to add a button on the detected screen.

This rule adds a `keyword` type XML element to the XML document. The `keyword` XML element is then processed by XSL transformation to display a new button in the HTML page. This element is similar in many ways to the XML elements created by the *SNA Commands* extraction rule.

Note: XML elements of the `keyword` type are handled by the `keyword` XSL template described in the `keyword.xsl` file. To change the way buttons are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	configuration	Defines the action triggered when pressing the button. Action can be: • a key action as `KEY_PF1`, • the name of a transaction if the **Transaction** property is set to `true`.
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".

Property	Type	Category	Description
Button layout	XMLRectangle	configuration	Defines the screen zone where the button is to be displayed. This property allows to position the created button element to a specific area of the screen. The created block will be created with the specified screen zone values as positioning attributes. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value. These positioning attributes have to be handled by the XSL template rule that displays the button.
Button name	String	configuration	Defines the text displayed on the button.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
End pattern	String	configuration	Defines the block ending pattern for the rule to be executed. Deprecated. It is recommended not to edit this property.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Start pattern	String	configuration	Defines the block starting pattern for the rule to be executed. Deprecated. It is recommended not to edit this property.

Property	Type	Category	Description
Transaction	boolean	configuration	Defines whether a transaction must be launched when clicking the button. If set to `true`, defines the launching of the transaction set in the **Action** property on button click, by adding a `dotransaction` attribute to the `keyword` XML element.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

Since the *Button* rule does not extract any data from the screen, it is not related to any relevant green screen sample.

In this example, a *Button* extraction rule is created with the following parameters:

```
Button [
  button layout=[x=9, y=17, width=12, height=1]
  button name="Click here"
  action="KEY_ENTER"
  transaction=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

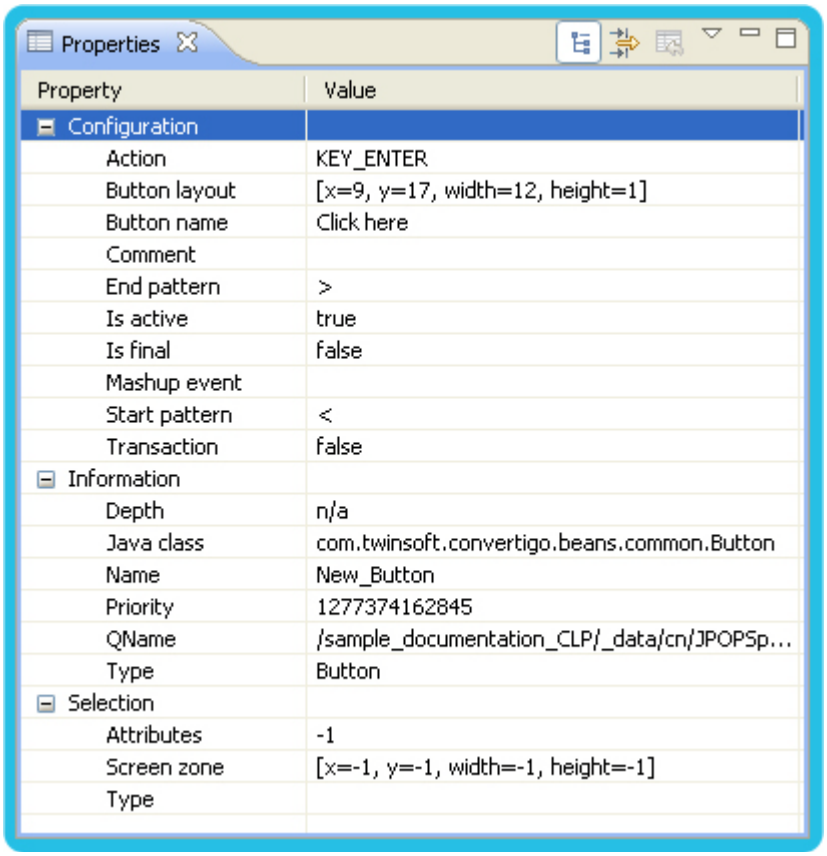


Figure 2 - 489: Button extraction rule - Configuration example

Button layout property is edited in the **Screen zone** editor:

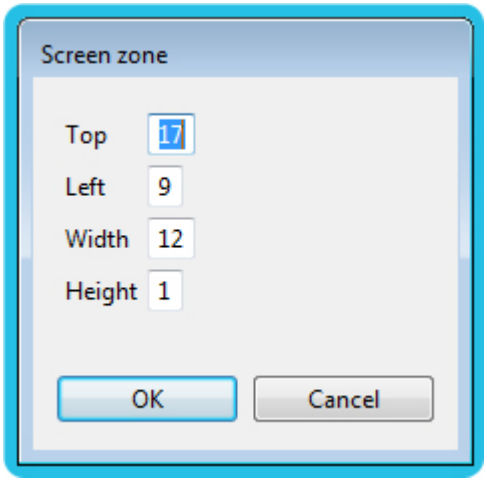


Figure 2 - 490: Button extraction rule - Button layout property edition

When applied, the *Button* rule creates an XML element of `keyword` type:

```
<block action="KEY_ENTER" column="9" height="1" line="17" name="untitled" type="keyword" width="12">Click here</block>
```

Figure 2 - 491: Button extraction rule - Resulting XML

After XSL transformation, thanks to `keyword` XSL template, it appears webized:

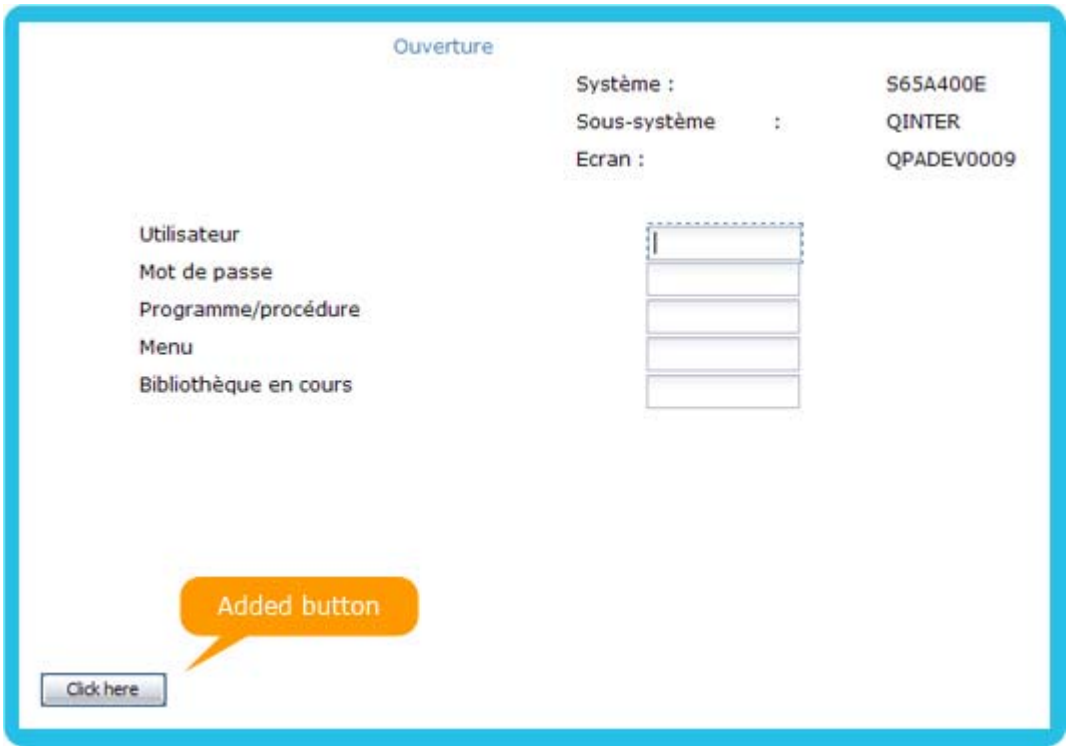


Figure 2 - 492: Button extraction rule - Webized page with rule

IMAGE 

OBJECT DESCRIPTION

Adds an image on a legacy screen.

Unlike other extraction rules, the *Image* extraction rule is not designed to extract data from the green screen, but to add an image on the detected screen.

This rule adds an `image` type XML element to the XML document. The `image` XML element is then processed by XSL transformation to display the image in the HTML page.

Note: XML elements of the `image` type are handled by the `image` XSL template described in the `image.xsl` file. To change the way images are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Action	String	configuration	Defines the action triggered when clicking on the image. Action can be: • a key action as `KEY_PF1`, • the name of a transaction if the **Transaction** property is set to `true`.
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Image label	String	configuration	Defines the alternative text displayed when the image is missing.

Property	Type	Category	Description
Image layout	XMLRectangle	configuration	Defines the screen zone where the image is to be displayed. This property allows to position the added image element to a specific area of the screen. The created block will be created with the specified screen zone values as positioning attributes. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value. These positioning attributes have to be handled by the XSL template rule that displays the image.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Keep image size	boolean	configuration	Defines whether the image must be kept to its original size or forced into the image layout. If set to `false`, the image is sized according to the image layout.

Property	Type	Category	Description
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Transaction	boolean	configuration	Defines whether a transaction must be launched when clicking on the image. If set to `true`, defines the launching of the transaction set in the **Action** property on image click, by adding a `dotransaction` attribute to the `image` XML element.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
URL	String	configuration	Defines the image URL. The URL can be defined either as an absolute or as a relative (to the project directory) URL.
Z-order	String	configuration	Defines the image `z-order`. Filling this property will define a `z-order` attribute to the added image. If left empty, no attribute is added to the `image` type XML element. Note: This attribute has to be handled in the webization framework to be taken into account. The `image` XSL template doesn't handle this property by default.

EXAMPLES

Since the *Image* rule does not extract any data from the screen, it is not related to any relevant green screen sample.

In this example, an *Image* extraction rule is created with the following parameters:

```
Image [
  image layout=[x=2, y=0, width=12, height=4]
  image label="Convertigo logo"
  keep image size=true
  transaction=false
  URL="images/convertigo-small.gif"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

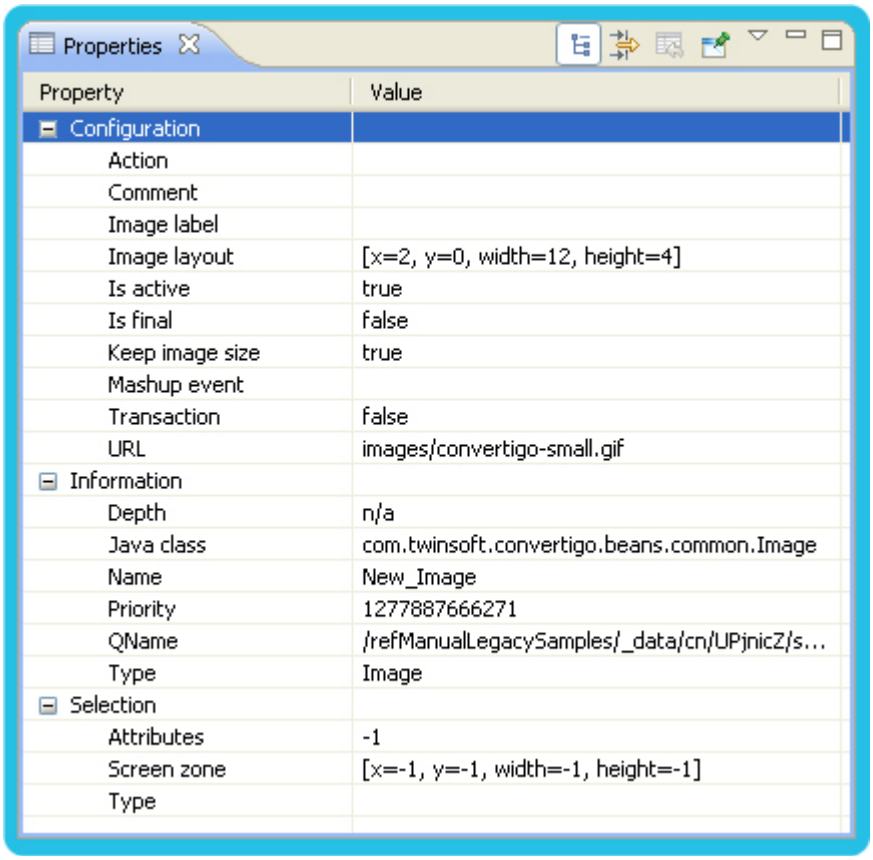


Figure 2 - 493: Image extraction rule - Configuration example

Image layout property is edited in the **Screen zone** editor:

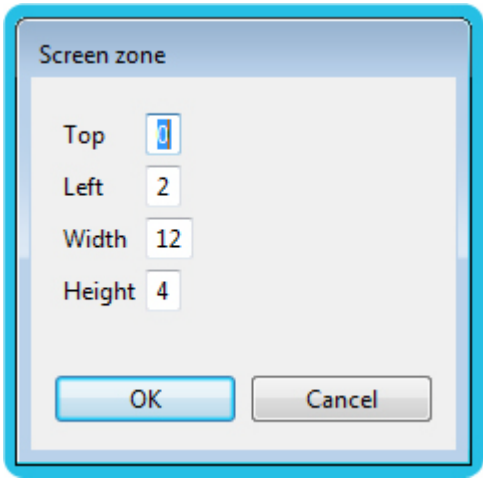


Figure 2 - 494: Image extraction rule - Image layout property edition

When applied, the *Image* rule creates an XML element of `image` type:

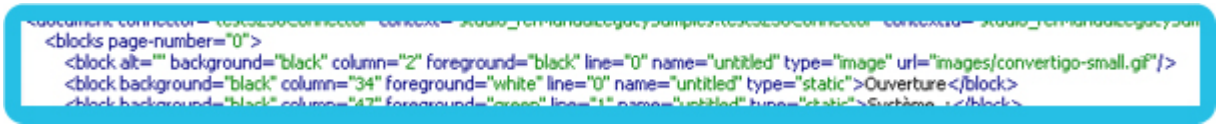


Figure 2 - 495: Image extraction rule - Resulting XML

After XSL transformation, thanks to `image` XSL template, it appears webized:

Figure 2 - 496: Image extraction rule - Webized page with rule

SNA GUI COMPONENTS

SNA COMMANDS 

OBJECT DESCRIPTION

Defines and handles keywords and commands found in legacy screens for SNA.

The *SNA commands* extraction rule detects patterns of the form `<KeyName><Separator><Action>`, where `KeyName` must be found in a list provided in the **Keywords table** property of the rule.

The rule transforms these blocks into `keyword` type blocks and adds following XML attributes to matching blocks:

- `action`: the javelin action to be executed on the mainframe,
- `data`: optional additional data to be sent with the action.

Note: XML elements of the `keyword` type are handled by the `keyword` XSL template described in the `keyword.xsl` file. To change the way keywords are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Case dependency	boolean	configuration	Defines whether letter case should be respected in keyword detection. If set to `false`, keywords match even if the case is not similar. For example, `pf13` and `PF13` match the `PF13` keyword.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Keyword separators	String	configuration	Defines a concatenated list of keyword separator(s) that can be used. This property lists separator characters used to separate each keyword name from its action. For example, "=" is a keyword separator that matches the `F3=EXIT` command.
Keywords table	XMLVector	configuration	Defines a list of keywords that can be detected, with replacement texts, optional data to send and associated action. This property is a list of *Keywords*. For each *Keyword* you can define: • **Keyword**: Keyword string to handle when found in the screen, • **Replace Text**: Replacement text for the keyword (hotspot label, optional), • **Sent data**: Data to be sent before performing action (optionnal), • **Action**: Action key to be pressed when the user clicks on the button (action key corresponding to found keyword). Notes: • A new keyword can be added to the list using the blue keyboard icon. The keywords defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. • The order of the keywords defined in this table is very important, it is used for detection priority. That means if two keywords can match on a block, only the first keyword from this table will be handled for this block.
Label location	int	configuration	Defines where to search for the label. The label location value can be one of the following: • `From right block`: the label is to the right of the keyword. For example: `F12=Open` • `From left block`: the label is to the left of the keyword. For example: `Open=F12` • `From block above`: the label is on top of the keyword. For example: `DO This F12` • `From block below`: the label is below the keyword. For example: `F12 Do This`

Property	Type	Category	Description
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Separator mandatory	boolean	configuration	Defines whether a separator character between keyword and label is mandatory. If set to `true`, a block containing a keyword (defined in **Keywords table** property) but which doesn't have one of the separator characters (defined in **Keyword separators** property) next to it will not match the rule.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

Figure 2 - 497: SNA commands extraction rule - Legacy screen

We can notice that six command keywords are present on the bottom of the screen, under the selection line.

Without the rule, the resulting XML is as follows:

```
    </block>
    <block background="black" column="1" foreground="green" line="18" name="untitled" type="static">Selection or command</block>
    <block background="black" column="1" foreground="green" line="19" name="untitled" type="static">===&gt;</block>
    <block background="black" column="6" foreground="green" hasFocus="true" line="19" name="__field_c6_l19" size="153" type="field"/>
    <block background="black" column="1" foreground="blue" line="21" name="untitled" type="static">F3=Exit</block>
    <block background="black" column="11" foreground="blue" line="21" name="untitled" type="static">F4=Prompt</block>
    <block background="black" column="23" foreground="blue" line="21" name="untitled" type="static">F9=Retrieve</block>
    <block background="black" column="37" foreground="blue" line="21" name="untitled" type="static">F12=Cancel</block>
    <block background="black" column="50" foreground="blue" line="21" name="untitled" type="static">F13=Information Assistant</block>
    <block background="black" column="1" foreground="blue" line="22" name="untitled" type="static">F23=Set initial menu</block>
    <block background="black" column="1" foreground="white" line="23" name="untitled" type="static">(C) COPYRIGHT IBM CORP. 1980, 2003.</block>
  </blocks>
</document>
```

Figure 2 - 498: SNA commands extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

Figure 2 - 499: SNA commands extraction rule - Webized page without rule

In this example, a *SNA commands* extraction rule is created with the following parameters:

```
SNA commands [
   screen zone=[x=1, y=21, width=78, height=2]
   case dependency=false
   keyword separators="="
   keywords table={
     Keyword [keyword="F10" sent data="" replace text=""
     action="KEY_PF10"],
     Keyword [keyword="F11" sent data="" replace text=""
     action="KEY_PF11"],
     Keyword [keyword="F12" sent data="" replace text=""
     action="KEY_PF12"],
```

```
      Keyword [keyword="F13" sent data="" replace text=""
      action="KEY_PF13"],
      Keyword [keyword="F14" sent data="" replace text=""
      action="KEY_PF14"],
      Keyword [keyword="F15" sent data="" replace text=""
      action="KEY_PF15"],
      Keyword [keyword="F16" sent data="" replace text=""
      action="KEY_PF16"],
      Keyword [keyword="F17" sent data="" replace text=""
      action="KEY_PF17"],
      Keyword [keyword="F18" sent data="" replace text=""
      action="KEY_PF18"],
      Keyword [keyword="F19" sent data="" replace text=""
      action="KEY_PF19"],
      Keyword [keyword="F20" sent data="" replace text=""
      action="KEY_PF20"],
      Keyword [keyword="F21" sent data="" replace text=""
      action="KEY_PF21"],
      Keyword [keyword="F22" sent data="" replace text=""
      action="KEY_PF22"],
      Keyword [keyword="F23" sent data="" replace text=""
      action="KEY_PF23"],
      Keyword [keyword="F24" sent data="" replace text=""
      action="KEY_PF24"],
      Keyword [keyword="F1" sent data="" replace text=""
      action="KEY_PF1"],
      Keyword [keyword="F2" sent data="" replace text=""
      action="KEY_PF2"],
      Keyword [keyword="F3" sent data="" replace text=""
      action="KEY_PF3"],
      Keyword [keyword="F4" sent data="" replace text=""
      action="KEY_PF4"],
      Keyword [keyword="F5" sent data="" replace text=""
      action="KEY_PF5"],
      Keyword [keyword="F6" sent data="" replace text=""
      action="KEY_PF6"],
      Keyword [keyword="F7" sent data="" replace text=""
      action="KEY_PF7"],
      Keyword [keyword="F8" sent data="" replace text=""
      action="KEY_PF8"],
      Keyword [keyword="F9" sent data="" replace text=""
      action="KEY_PF9"]
    }
    label=From right block
    separator mendatory=true
  ]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

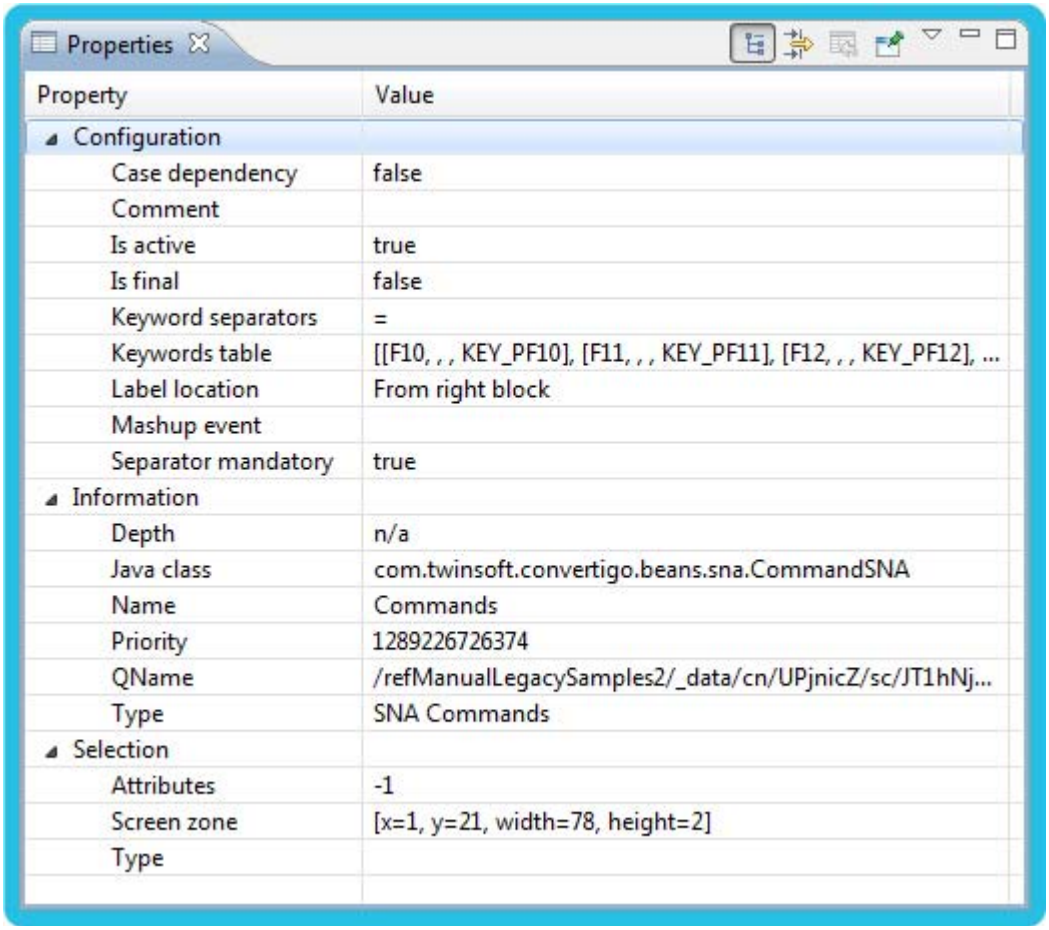


Figure 2 - 500: SNA commands extraction rule - Configuration example

Keywords table property is edited in the **Keywords** editor:

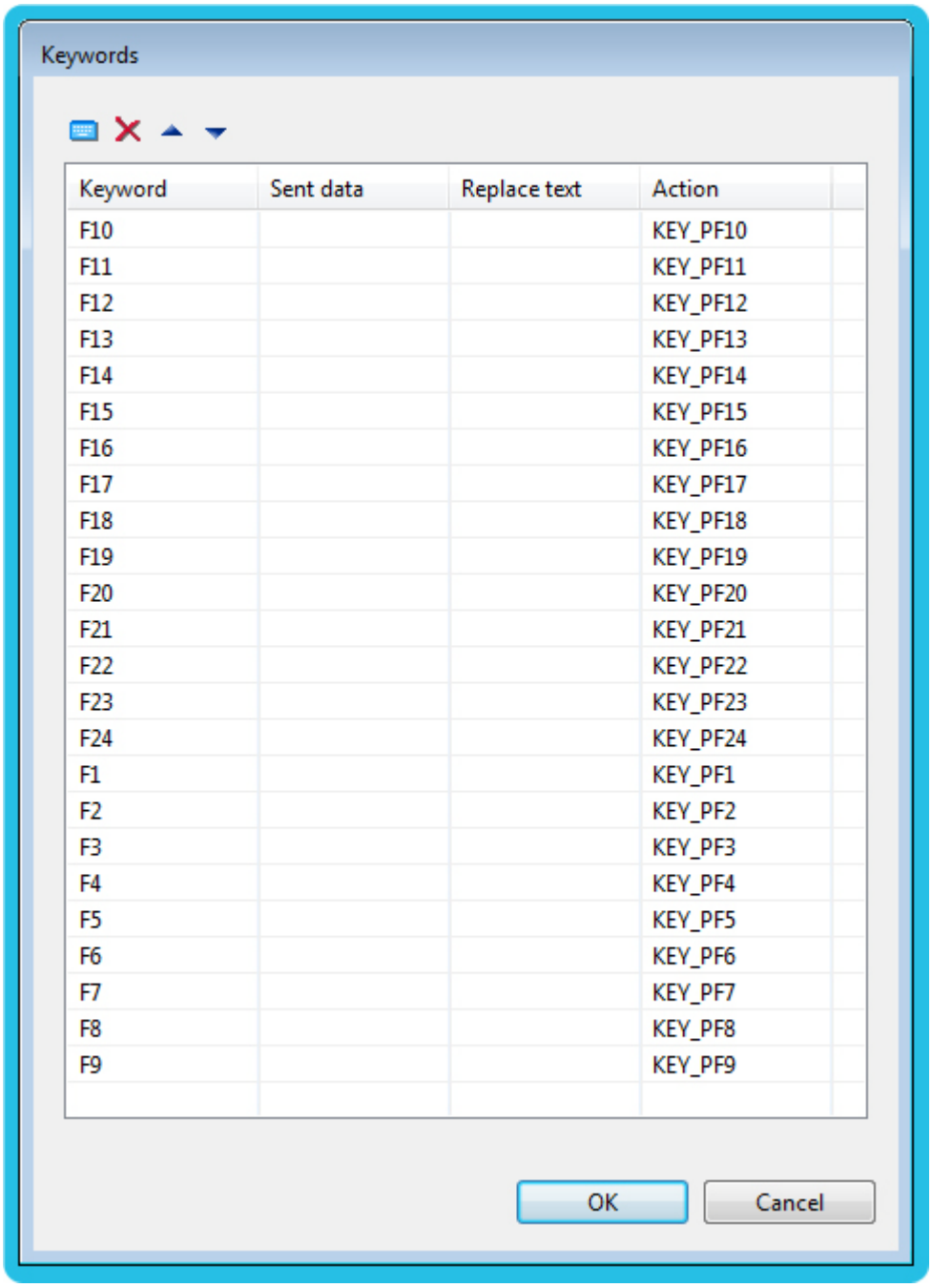


Figure 2 - 501: SNA commands extraction rule - Keywords table property edition

The keywords order in this table is very important because it defines the priority of detection. The `F12`, `F13` and `F24` keywords are detected first on corresponding blocks, instead of `F1` or `F2` that could also be detected on the same blocks, because they are placed before in the **Keywords table**.

When the rule is executed, it changes matching blocks `type` to `keyword` and adds `action` and `data` XML attributes:

```
    </block>
    <block background="black" column="1" foreground="green" line="18" name="untitled" type="static">Selection or command</block>
    <block background="black" column="1" foreground="green" line="19" name="untitled" type="static">===&amp;gt;</block>
    <block background="black" column="6" foreground="green" hasFocus="true" line="19" name="__field_c6_l19" size="153" type="field"/>
    <block action="KEY_PF3" background="black" column="1" data="" foreground="blue" line="21" name="untitled" type="keyword">Exit</block>
    <block action="KEY_PF4" background="black" column="11" data="" foreground="blue" line="21" name="untitled" type="keyword">Prompt</block>
    <block action="KEY_PF9" background="black" column="23" data="" foreground="blue" line="21" name="untitled" type="keyword">Retrieve</block>
    <block action="KEY_PF12" background="black" column="37" data="" foreground="blue" line="21" name="untitled" type="keyword">Cancel</block>
    <block action="KEY_PF13" background="black" column="50" data="" foreground="blue" line="21" name="untitled" type="keyword">Information Assistant</block>
    <block action="KEY_PF23" background="black" column="1" data="" foreground="blue" line="22" name="untitled" type="keyword">Set initial menu</block>
    <block background="black" column="1" foreground="white" line="23" name="untitled" type="static">(C) COPYRIGHT IBM CORP. 1980, 2003.</block>
  </blocks>
</document>
```

Figure 2 - 502: SNA commands extraction rule - Resulting XML with rule

After XSL transformation, thanks to `keyword` XSL template, it appears webized:

Figure 2 - 503: SNA commands extraction rule - Webized page with rule

AS400 MENU 

OBJECT DESCRIPTION

Defines an AS/400 like menu.

The *AS400 menu* extraction rule handles AS/400 and other IBM mainframe application menus, such as 5250 or 3270 screen menus. In such applications, menus have all a standard format which the *AS400 menu* extraction rule automatically manages.

The rule detects in the green screen patterns such as the following:

```
<number><separator> <menu item label>
```

The number can be any number. The separator is represented by one character only; it is set in the **Separators** property. The menu item label can be any string of characters.

The *AS400 menu* extraction rule creates an XML element of snamenu type. This element contains the menu itself, with each line of the menu being tagged as a menuitem element, of menuitem type.

Each menuitem element includes the following attributes added by the extraction rule:

- id: Number identified in the original text detected as pattern (i.e. number of the menu item),
- literal: Text content of the menu item (i.e. its label).

Note: XML elements of the snamenu type are handled by the SNA menu XSL template described in the snamenu.xsl file. To change the way SNA menus are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: Foreground color, Background color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: bold, reverse, underlined, blink, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Separators	String	configuration	Defines a concatenated list of separator characters that can be used between a menu item number and its label. This property lists separator characters used to separate each menu item number from its label. For example, "`.`" is a separator that matches the `1. User task` menu item.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

Figure 2 - 504: AS400 menu extraction rule - Legacy screen

We can notice that a menu is present in the middle of the page, listing twelve items.

Without the rule, the resulting XML is as follows:

```
<block background="black" column="1" foreground="blue" line="2" name="untitled" type="static">Select one of the following:</block>
<block background="black" column="6" foreground="green" line="4" name="untitled" type="static">1. User tasks</block>
<block background="black" column="6" foreground="green" line="5" name="untitled" type="static">2. Office tasks</block>
<block background="black" column="6" foreground="green" line="6" name="untitled" type="static">3. General system tasks</block>
<block background="black" column="6" foreground="green" line="7" name="untitled" type="static">4. Files, libraries, and folders</block>
<block background="black" column="6" foreground="green" line="8" name="untitled" type="static">5. Programming</block>
<block background="black" column="6" foreground="green" line="9" name="untitled" type="static">6. Communications</block>
<block background="black" column="6" foreground="green" line="10" name="untitled" type="static">7. Define or change the system</block>
<block background="black" column="6" foreground="green" line="11" name="untitled" type="static">8. Problem handling</block>
<block background="black" column="6" foreground="green" line="12" name="untitled" type="static">9. Display a menu</block>
<block background="black" column="5" foreground="green" line="13" name="untitled" type="static">10. Information Assistant options</block>
<block background="black" column="5" foreground="green" line="14" name="untitled" type="static">11. iSeries Access tasks</block>
<block background="black" column="5" foreground="green" line="16" name="untitled" type="static">90. Sign off</block>
```

Figure 2 - 505: AS400 menu extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

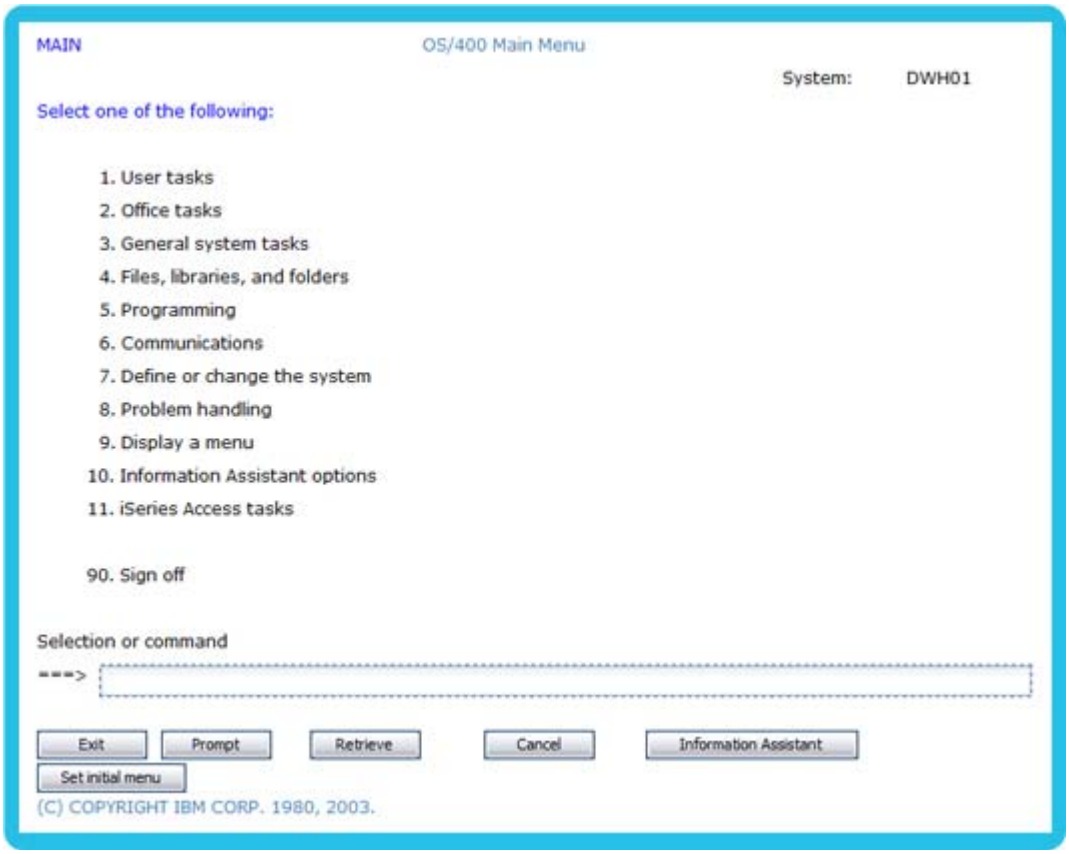


Figure 2 - 506: AS400 menu extraction rule - Webized page without rule

In this example, a *AS400 menu* extraction rule is created with the following parameters:

```
AS400 menu [
   screen zone=[x=0, y=3, width=79, height=14]
   separators="."
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

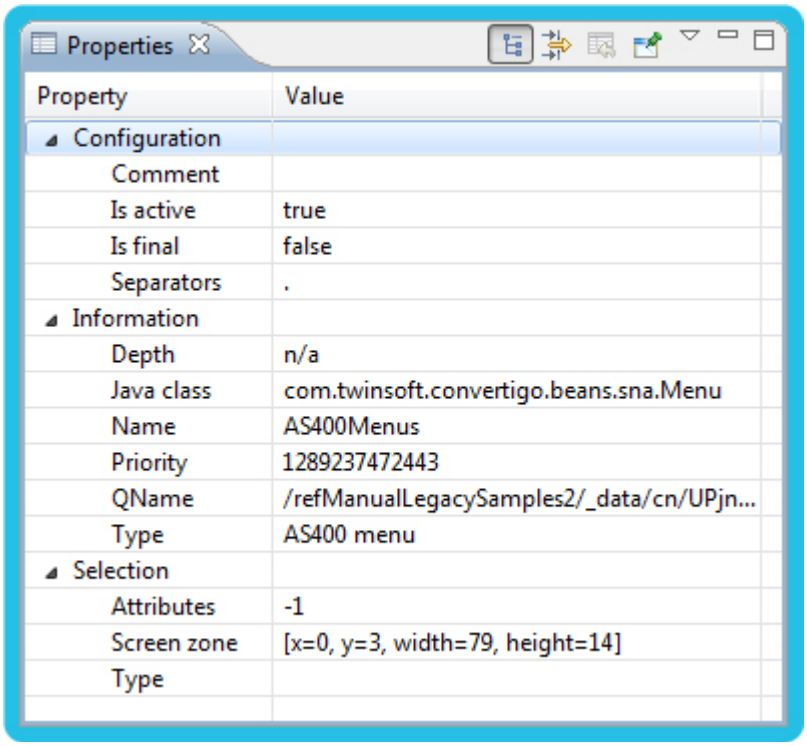


Figure 2 - 507: AS400 menu extraction rule - Configuration example

When the rule is executed, it detects in the green screen blocks that match patterns such as the following:

`<number><separator> <menu item label>`

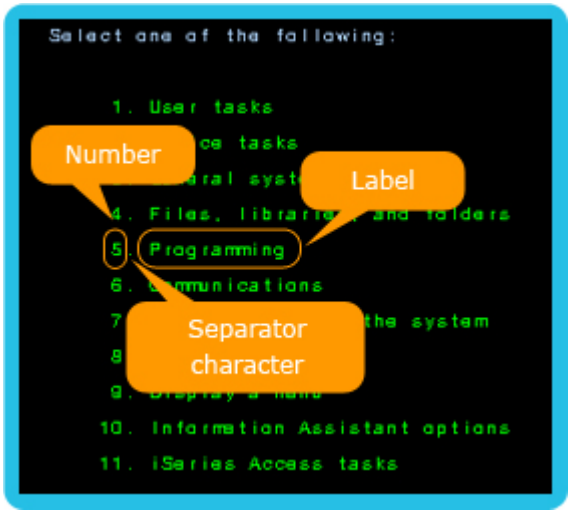


Figure 2 - 508: AS400 menu extraction rule - Principle

The rule creates a `snamenu` type block and inserts matching blocks under this element, tagging them `menuitem`:

```
<block background="black" column="6" foreground="green" line="4" name="untitled" type="snamenu">
  <menu>
    <menuitem background="black" column="6" foreground="green" id="1" line="4" literal="User tasks" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="2" line="5" literal="Office tasks" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="3" line="6" literal="General system tasks" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="4" line="7" literal="Files, libraries, and folders" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="5" line="8" literal="Programming" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="6" line="9" literal="Communications" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="7" line="10" literal="Define or change the system" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="8" line="11" literal="Problem handling" reverse="false" type="menuitem"/>
    <menuitem background="black" column="6" foreground="green" id="9" line="12" literal="Display a menu" reverse="false" type="menuitem"/>
    <menuitem background="black" column="5" foreground="green" id="10" line="13" literal="Information Assistant options" reverse="false" type="menuitem"/>
    <menuitem background="black" column="5" foreground="green" id="11" line="14" literal="iSeries Access tasks" reverse="false" type="menuitem"/>
    <menuitem background="black" column="5" foreground="green" id="90" line="16" literal="Sign off" reverse="false" type="menuitem"/>
  </menu>
</block>
```

Figure 2 - 509: AS400 menu extraction rule - Resulting XML with rule

The rule also changes `menuitem` elements `type` to `menuitem` and adds `id` and `literal` XML attributes.

After XSL transformation, thanks to `snamenu` XSL template, it appears webized:

Figure 2 - 510: AS400 menu extraction rule - Webized page with rule

SUBFILE 

OBJECT DESCRIPTION

Extracts structured data from an `AS/400` like subfile.

The *Subfile* extraction rule extracts table data from screens for AS/400 specific subfile format. It is similar to the *Table* extraction rule, but runs automatically with no need to manually configure columns.

Subfiles are detected only if required parameters are properly set. A number of conditions must also be met for the rule to apply:

- End of the subfile zone must be indicated by an end marker string (End, +, More..., etc.),
- Presentation attributes of the end marker must be different than those of the subfile content (usually white on black background),
- Presentation attributes of data must be different than those of the end marker or titles (data usually displayed in green on black background),
- Subfile must start with a title line,
- Presentation attributes of titles must be different than those of the subfile content (usually white on black background, sometimes underlined),
- Each title must be separated from the next by a minimum of 2 spaces (if the title line is made of one large field only) OR each title is in a distinct AS400 DSPV map field,
- Each title is left-aligned in its column,
- Actions (if any) are to be listed above the subfile,
- Presentation attributes of actions (if any) must be different than those of titles (usually blue on black background),
- Each action (if any) must respect the following pattern: `<action code><separator><action label>`. By default, `separator` is represented by the "=" sign. For example: `E=Edit`, `A=Add`, etc.
- For actions to be detected, the first column of each non-empty line in the subfile must contain an input field.

Note: Actions (last four points) are optional for correct subfile detection.

As a conclusion, the subfile structure always follows the same pattern:

`<Actions>` (optionnal)

`<Title line>`

`<Subfile content>`

`<End marker>`

This order cannot be changed, and no other data can be inserted in the 5250 screen zone of the subfile.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Actions line attributes	int	configuration	Defines the attributes of the actions of the subfile (if existing). Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
End marker string attributes	int	configuration	Defines the attributes of the end marker string. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Start detection from line	int	configuration	Detects subfile from specified line. The end marker string is detected by default anywhere on the screen. You can specify the line where this detection should start. Useful to ignore subfile markers that could be present in the subfile data.
Subfile end marker strings	String	configuration	Defines the list of strings that can be used to detect the end of the subfile. This string must be located according to the CUA spec (below last subfile row, to the far right) and be provided as a comma separated list of marker strings. Make sure that all of them are set, including in any language. For example: A `suivre...,Fin,+,Bottom,End,More...`
Title row attributes	int	configuration	Defines the attributes of the title row of the subfile. Make sure that attributes are described so that all title lines match. For example, the **underline** parameter of this property can be set to "`don't care`" to match the normal and underline titles. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

Let's consider the following legacy screen:

Figure 2 - 511: Subfile extraction rule - Legacy screen containging CUA subfile

It shows a standard CUA subfile. Subfile zones can be defined as follows:

Table 2 - 5: Subfile zone description

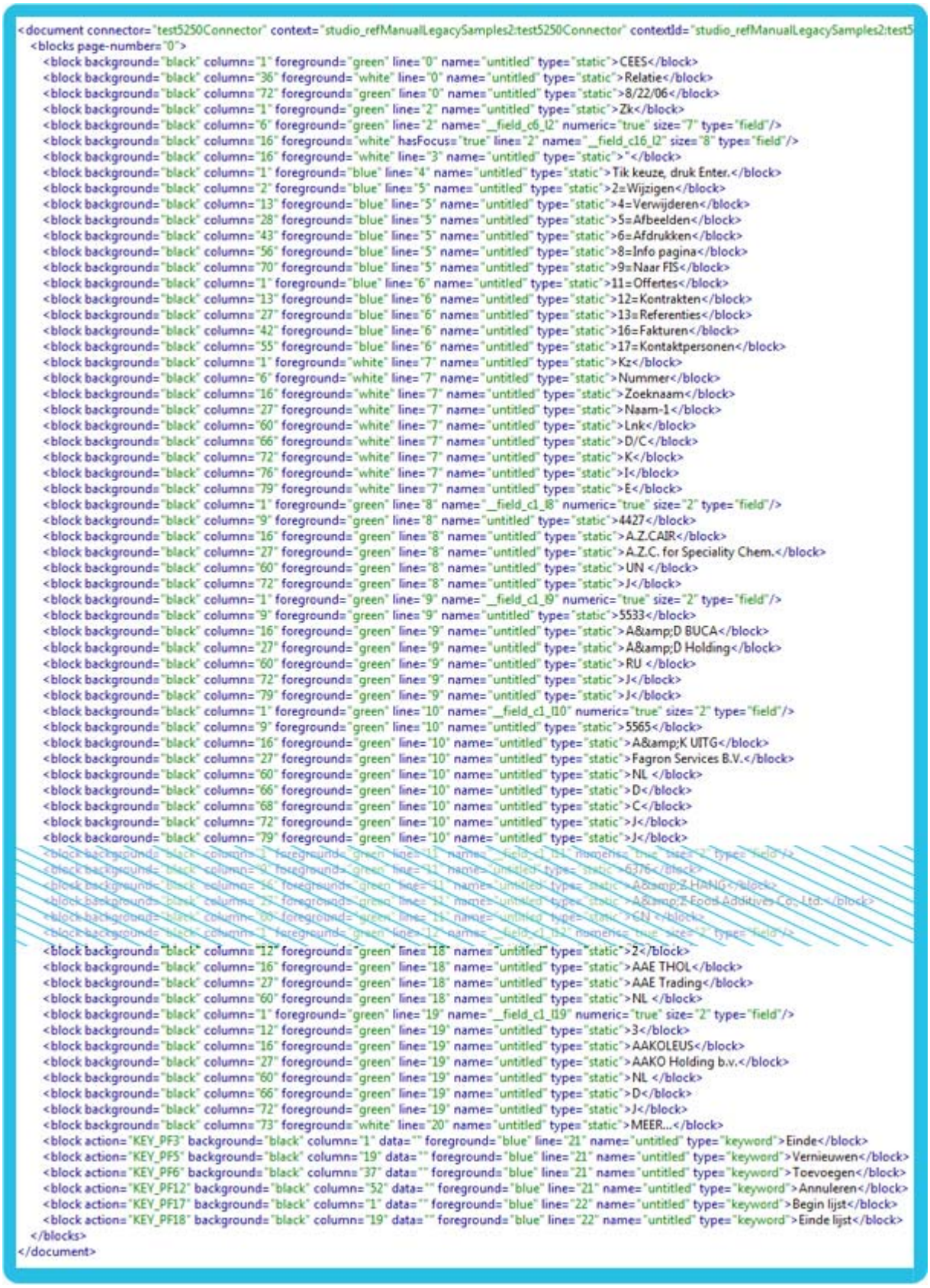
Subfile Zone	Description
Action line	Zone where actions can be launched from in a table. Displayed in the following format: `<ActionCode><Separator><Label>`. Not always present.

Table 2 - 5: Subfile zone description (...)

Subfile Zone	Description
Title line	Zone where titles are displayed in the subfile. Usually displayed with different screen attributes.
Data	Zone where data is displayed. The screen attributes of data is usually different from titles.
Marker string	On the subfile, marker such as `Bottom`, `End`, `More..., Top`. Can also be a "+" sign.
Action column	Zone where users input action code to be applied to associated line in the sufile.

Let's configure this rule on a sample. If we consider the previous legacy screen, we can notice the five subfile zones, which indicates that a *Subfile* extraction rule can be created.

Without the rule, the XML resulting from this screen is as follows:



```
<document connector="test5250Connector" context="studio_refManualLegacySamples2:test5250Connector" contextId="studio_refManualLegacySamples2:test5
  <blocks page-number="0">
    <block background="black" column="1" foreground="green" line="0" name="untitled" type="static">CEES</block>
    <block background="black" column="36" foreground="white" line="0" name="untitled" type="static">Relatie</block>
    <block background="black" column="72" foreground="green" line="0" name="untitled" type="static">8/22/06</block>
    <block background="black" column="1" foreground="green" line="2" name="untitled" type="static">Zk</block>
    <block background="black" column="6" foreground="green" line="2" name="__field_c6_l2" numeric="true" size="7" type="field"/>
    <block background="black" column="16" foreground="white" hasFocus="true" line="2" name="__field_c16_l2" size="8" type="field"/>
    <block background="black" column="16" foreground="white" line="3" name="untitled" type="static">*</block>
    <block background="black" column="1" foreground="blue" line="4" name="untitled" type="static">Tik keuze, druk Enter.</block>
    <block background="black" column="2" foreground="blue" line="5" name="untitled" type="static">2=Wijzigen</block>
    <block background="black" column="13" foreground="blue" line="5" name="untitled" type="static">4=Verwijderen</block>
    <block background="black" column="28" foreground="blue" line="5" name="untitled" type="static">5=Afbeelden</block>
    <block background="black" column="43" foreground="blue" line="5" name="untitled" type="static">6=Afdrukken</block>
    <block background="black" column="56" foreground="blue" line="5" name="untitled" type="static">8=Info pagina</block>
    <block background="black" column="70" foreground="blue" line="5" name="untitled" type="static">9=Naar FIS</block>
    <block background="black" column="1" foreground="blue" line="6" name="untitled" type="static">11=Offertes</block>
    <block background="black" column="13" foreground="blue" line="6" name="untitled" type="static">12=Kontrakten</block>
    <block background="black" column="27" foreground="blue" line="6" name="untitled" type="static">13=Referenties</block>
    <block background="black" column="42" foreground="blue" line="6" name="untitled" type="static">16=Fakturen</block>
    <block background="black" column="55" foreground="blue" line="6" name="untitled" type="static">17=Kontaktpersonen</block>
    <block background="black" column="1" foreground="white" line="7" name="untitled" type="static">Kz</block>
    <block background="black" column="6" foreground="white" line="7" name="untitled" type="static">Nummer</block>
    <block background="black" column="16" foreground="white" line="7" name="untitled" type="static">Zoeknaam</block>
    <block background="black" column="27" foreground="white" line="7" name="untitled" type="static">Naam-1</block>
    <block background="black" column="60" foreground="white" line="7" name="untitled" type="static">Lnk</block>
    <block background="black" column="66" foreground="white" line="7" name="untitled" type="static">D/C</block>
    <block background="black" column="72" foreground="white" line="7" name="untitled" type="static">K</block>
    <block background="black" column="76" foreground="white" line="7" name="untitled" type="static">I</block>
    <block background="black" column="79" foreground="white" line="7" name="untitled" type="static">E</block>
    <block background="black" column="1" foreground="green" line="8" name="__field_c1_l8" numeric="true" size="2" type="field"/>
    <block background="black" column="9" foreground="green" line="8" name="untitled" type="static">4427</block>
    <block background="black" column="16" foreground="green" line="8" name="untitled" type="static">A.Z.CAIR</block>
    <block background="black" column="27" foreground="green" line="8" name="untitled" type="static">A.Z.C. for Speciality Chem.</block>
    <block background="black" column="60" foreground="green" line="8" name="untitled" type="static">UN </block>
    <block background="black" column="72" foreground="green" line="8" name="untitled" type="static">J</block>
    <block background="black" column="1" foreground="green" line="9" name="__field_c1_l9" numeric="true" size="2" type="field"/>
    <block background="black" column="9" foreground="green" line="9" name="untitled" type="static">5533</block>
    <block background="black" column="16" foreground="green" line="9" name="untitled" type="static">A&amp;amp;D BUCA</block>
    <block background="black" column="27" foreground="green" line="9" name="untitled" type="static">A&amp;amp;D Holding</block>
    <block background="black" column="60" foreground="green" line="9" name="untitled" type="static">RU </block>
    <block background="black" column="72" foreground="green" line="9" name="untitled" type="static">J</block>
    <block background="black" column="79" foreground="green" line="9" name="untitled" type="static">J</block>
    <block background="black" column="1" foreground="green" line="10" name="__field_c1_l10" numeric="true" size="2" type="field"/>
    <block background="black" column="9" foreground="green" line="10" name="untitled" type="static">5565</block>
    <block background="black" column="16" foreground="green" line="10" name="untitled" type="static">A&amp;amp;K UITG</block>
    <block background="black" column="27" foreground="green" line="10" name="untitled" type="static">Fagron Services B.V.</block>
    <block background="black" column="60" foreground="green" line="10" name="untitled" type="static">NL </block>
    <block background="black" column="66" foreground="green" line="10" name="untitled" type="static">D</block>
    <block background="black" column="68" foreground="green" line="10" name="untitled" type="static">C</block>
    <block background="black" column="72" foreground="green" line="10" name="untitled" type="static">J</block>
    <block background="black" column="79" foreground="green" line="10" name="untitled" type="static">J</block>
    <block background="black" column="1" foreground="green" line="11" name="__field_c1_l11" numeric="true" size="2" type="field"/>
    <block background="black" column="9" foreground="green" line="11" name="untitled" type="static">6376</block>
    <block background="black" column="16" foreground="green" line="11" name="untitled" type="static">A&amp;amp;Z HANG</block>
    <block background="black" column="27" foreground="green" line="11" name="untitled" type="static">A&amp;amp;Z Food Additives Co., Ltd.</block>
    <block background="black" column="60" foreground="green" line="11" name="untitled" type="static">CN </block>
    <block background="black" column="1" foreground="green" line="12" name="__field_c1_l12" numeric="true" size="2" type="field"/>
    <block background="black" column="12" foreground="green" line="18" name="untitled" type="static">2</block>
    <block background="black" column="16" foreground="green" line="18" name="untitled" type="static">AAE THOL</block>
    <block background="black" column="27" foreground="green" line="18" name="untitled" type="static">AAE Trading</block>
    <block background="black" column="60" foreground="green" line="18" name="untitled" type="static">NL </block>
    <block background="black" column="1" foreground="green" line="19" name="__field_c1_l19" numeric="true" size="2" type="field"/>
    <block background="black" column="12" foreground="green" line="19" name="untitled" type="static">3</block>
    <block background="black" column="16" foreground="green" line="19" name="untitled" type="static">AAKOLEUS</block>
    <block background="black" column="27" foreground="green" line="19" name="untitled" type="static">AAKO Holding b.v.</block>
    <block background="black" column="60" foreground="green" line="19" name="untitled" type="static">NL </block>
    <block background="black" column="66" foreground="green" line="19" name="untitled" type="static">D</block>
    <block background="black" column="72" foreground="green" line="19" name="untitled" type="static">J</block>
    <block background="black" column="73" foreground="white" line="20" name="untitled" type="static">MEER...</block>
    <block action="KEY_PF3" background="black" column="1" data="" foreground="blue" line="21" name="untitled" type="keyword">Einde</block>
    <block action="KEY_PF5" background="black" column="19" data="" foreground="blue" line="21" name="untitled" type="keyword">Vernieuwen</block>
    <block action="KEY_PF6" background="black" column="37" data="" foreground="blue" line="21" name="untitled" type="keyword">Toevoegen</block>
    <block action="KEY_PF12" background="black" column="52" data="" foreground="blue" line="21" name="untitled" type="keyword">Annuleren</block>
    <block action="KEY_PF17" background="black" column="1" data="" foreground="blue" line="22" name="untitled" type="keyword">Begin lijst</block>
    <block action="KEY_PF18" background="black" column="19" data="" foreground="blue" line="22" name="untitled" type="keyword">Einde lijst</block>
  </blocks>
</document>
```

Figure 2 - 512: Subfile extraction rule - Resulting XML without rule

After XSL transformation, it appears webized:

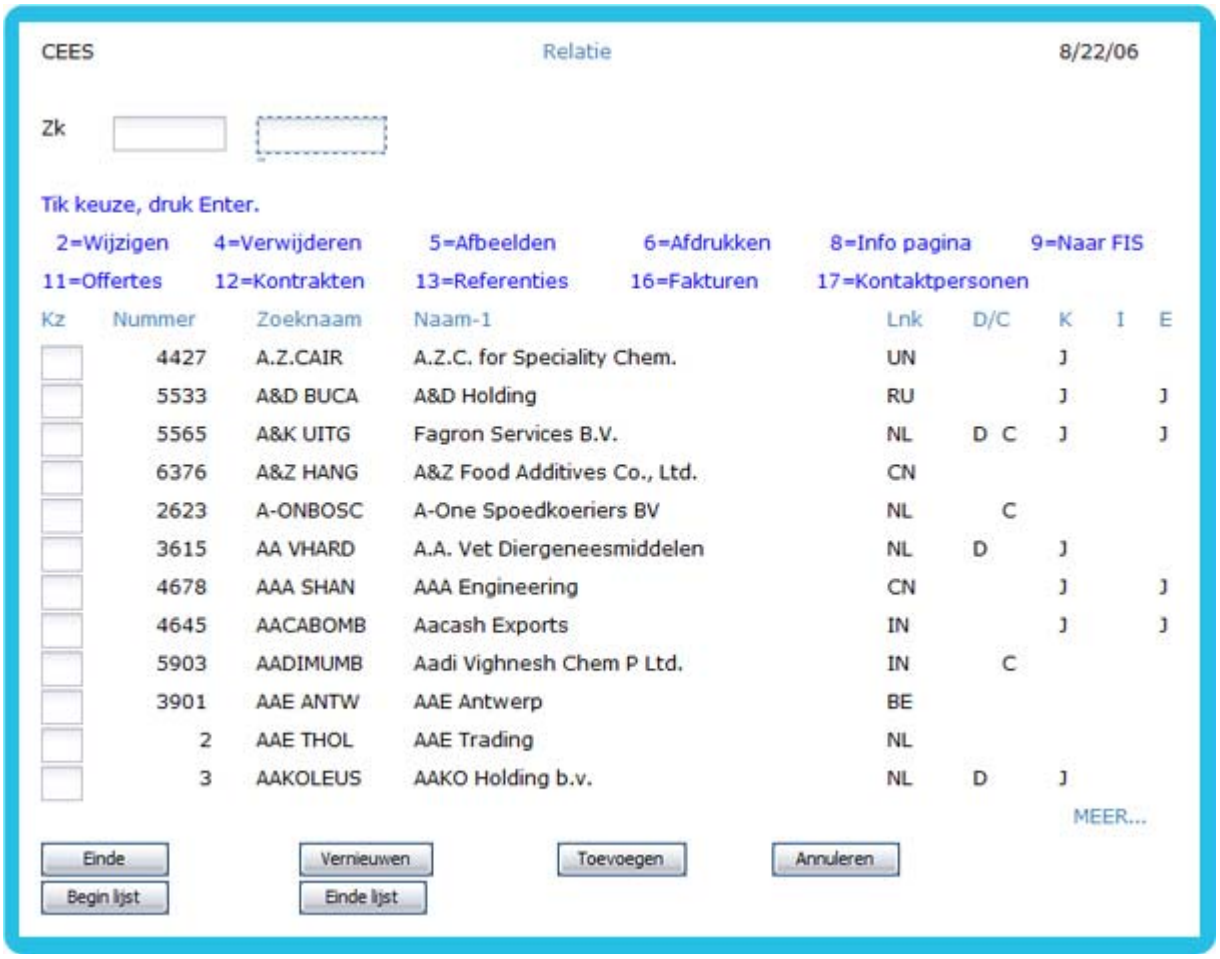


Figure 2 - 513: Table extraction rule - Webized page without rule

In this example, a *Subfile* extraction rule is created with the following parameters:

```
Subfile [
   actions label separator="="
   actions line attributes=[foreground=blue, background=black]
   auto-validate=false
   end marker string attributes=[foreground=white, background=black]
   start detection from line=0
   subfile end marker strings="MEER..."
   remove actions line=true
   title row atrtibutes=[foreground=white, background=black]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

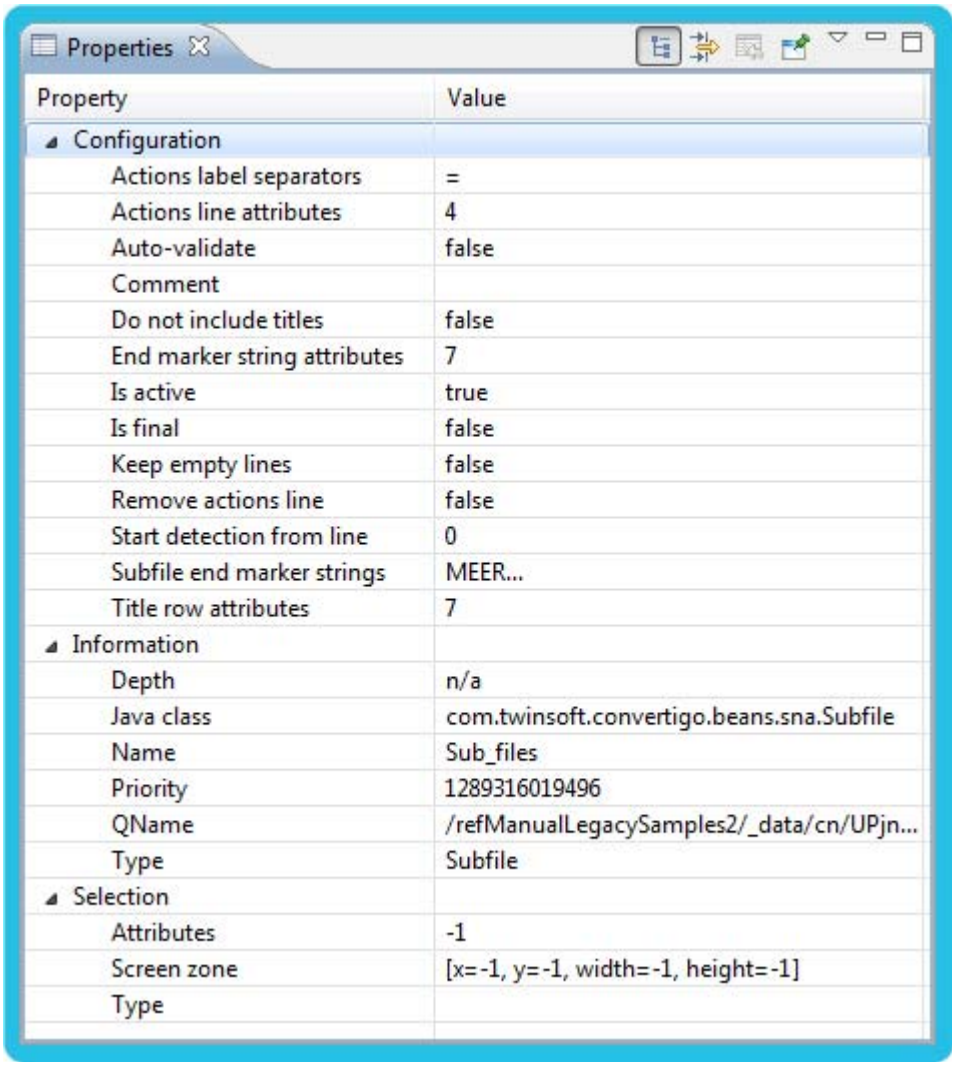


Figure 2 - 514: Subfile extraction rule - Configuration example

The **Title row attributes**, **Actions line attributes** and **End marker string attributes** properties are edited in the **Attributes** editor (for example here the **End marker string attributes** property):

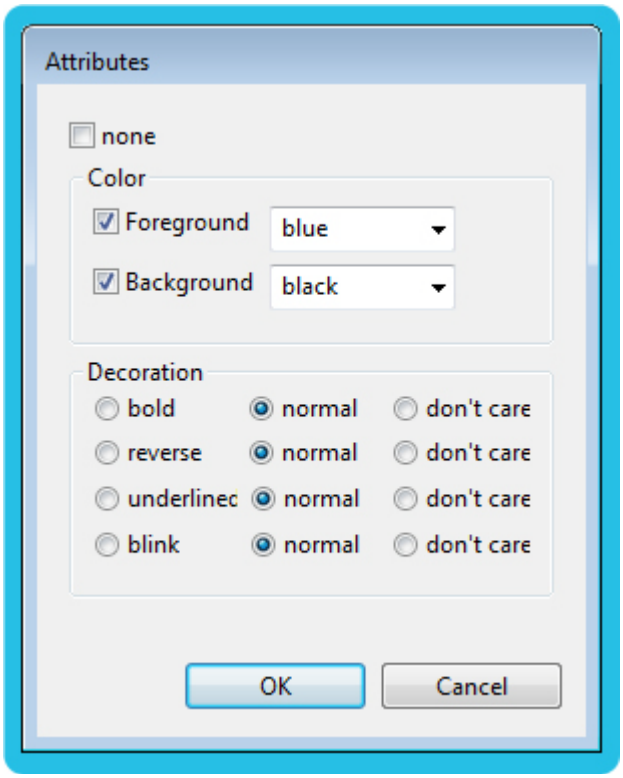


Figure 2 - 515: Subfile extraction rule - Attributes editior

In this sample case, the **Sufbfile end marker string** property is set only with one string, `MEER...`, corresponding to the one marker that can be found on this screen. The *Subfile* extraction rule is then specific to the screens having this marker.

The **Screen zone** property is not set in this example for the *Subfile* detection to be performed on the whole screen.

After data have been extracted, the corresponding XML is generated as follows:

```
<document connector="test5250Connector" context="studio_refManualLegacySamples2:test5250Connector" contextId="studio_refManualLegacySamples2:test5250Connector" cursorColumn="16"
  <blocks page-number="0">
    <block background="black" column="1" foreground="green" line="0" name="untitled" type="static">CEES</block>
    <block background="black" column="36" foreground="white" line="0" name="untitled" type="static">Relatie</block>
    <block background="black" column="72" foreground="green" line="0" name="untitled" type="static">8/22/06</block>
    <block background="black" column="1" foreground="green" line="2" name="untitled" type="static">Zk</block>
    <block background="black" column="6" foreground="green" line="2" name="__field_c6_l2" numeric="true" size="7" type="field"/>
    <block background="black" column="16" foreground="white" hasFocus="true" line="2" name="__field_c16_l2" size="8" type="field"/>
    <block background="black" column="16" foreground="white" line="3" name="untitled" type="static">*</block>
    <block background="black" column="1" foreground="blue" line="4" name="untitled" type="static">Tik keuze, druk Enter.</block>
    <table background="black" column="1" foreground="black" height="12" line="8" name="table" offset="2" rows="12" size="80" subfile="true" titleHeight="1" type="table" width="80">
      <actionsTable background="black" column="2" foreground="black" line="5" name="" type="actionsTable">
        <action char="2" key="null" label="Wijzigen"/>
        <action char="4" key="null" label="Verwijderen"/>
        <action char="5" key="null" label="Afbeelden"/>
        <action char="6" key="null" label="Afdrukken"/>
        <action char="8" key="null" label="Info pagina"/>
        <action char="9" key="null" label="Naar FIS"/>
        <action char="11" key="null" label="Offertes"/>
        <action char="12" key="null" label="Kontrakten"/>
        <action char="13" key="null" label="Referenties"/>
        <action char="16" key="null" label="Fakturen"/>
        <action char="17" key="null" label="Kontaktpersonen"/>
      </actionsTable>
      <row background="black" column="0" foreground="black" index="0" line="0" name="" page="1" type="row">
        <Title background="black" column="1" foreground="black" line="0" name="" size="5" type="item">
          <block background="black" column="1" foreground="white" line="7" name="untitled" type="static">Kz </block>
        </Title>
        <Title background="black" column="6" foreground="black" line="0" name="" size="10" type="item">
          <block background="black" column="6" foreground="white" line="7" name="untitled" type="static">Nummer </block>
        </Title>
        <Title background="black" column="16" foreground="black" line="0" name="" size="11" type="item">
          <block background="black" column="16" foreground="white" line="7" name="untitled" type="static">Zoeknaam </block>
        </Title>
        <Title background="black" column="27" foreground="black" line="0" name="" size="33" type="item">
          <block background="black" column="27" foreground="white" line="7" name="untitled" type="static">Naam1 </block>
        </Title>
        <Title background="black" column="60" foreground="black" line="0" name="" size="6" type="item">
          <block background="black" column="60" foreground="white" line="7" name="untitled" type="static">Lnk </block>
        </Title>
        <Title background="black" column="66" foreground="black" line="0" name="" size="6" type="item">
          <block background="black" column="66" foreground="white" line="7" name="untitled" type="static">D/C </block>
        </Title>
        <Title background="black" column="72" foreground="black" line="0" name="" size="4" type="item">
          <block background="black" column="72" foreground="white" line="7" name="untitled" type="static">K </block>
        </Title>
        <Title background="black" column="76" foreground="black" line="0" name="" size="3" type="item">
          <block background="black" column="76" foreground="white" line="7" name="untitled" type="static">I </block>
        </Title>
        <Title background="black" column="79" foreground="black" line="0" name="" size="2" type="item">
          <block background="black" column="79" foreground="white" line="7" name="untitled" type="static">E </block>
        </Title>
      </row>
      <row background="black" column="1" foreground="black" index="1" line="8" name="" page="1" type="row">
        <Kz background="black" column="1" columnSelection="true" foreground="green" line="8" name="__field_c1_l8" numeric="true" size="2" type="field"/>
        <Nummer background="black" column="9" foreground="green" line="8" name="untitled" type="static">4427</Nummer>
        <Zoeknaam background="black" column="16" foreground="green" line="8" name="untitled" type="static">A.Z.CAIR</Zoeknaam>
        <Naam1 background="black" column="27" foreground="green" line="8" name="untitled" type="static">A.Z.C. for Speciality Chem.</Naam1>
        <Lnk background="black" column="60" foreground="green" line="8" name="untitled" type="static">UN </Lnk>
        <DC background="black" column="0" foreground="black" line="0" name="" type="static"/>
        <K background="black" column="72" foreground="green" line="8" name="untitled" type="static">J</K>
        <I background="black" column="0" foreground="black" line="0" name="" type="static"/>
        <E background="black" column="0" foreground="black" line="0" name="" type="static"/>
      </row>
      <row background="black" column="1" foreground="black" index="12" line="19" name="" page="1" type="row">
        <Kz background="black" column="1" columnSelection="true" foreground="green" line="19" name="__field_c1_l19" numeric="true" size="2" type="field"/>
        <Nummer background="black" column="12" foreground="green" line="19" name="untitled" type="static">3</Nummer>
        <Zoeknaam background="black" column="16" foreground="green" line="19" name="untitled" type="static">AAKOLEUS</Zoeknaam>
        <Naam1 background="black" column="27" foreground="green" line="19" name="untitled" type="static">AAKO Holding b.v.</Naam1>
        <Lnk background="black" column="60" foreground="green" line="19" name="untitled" type="static">NL </Lnk>
        <DC background="black" column="66" foreground="green" line="19" name="untitled" type="static">D</DC>
        <K background="black" column="72" foreground="green" line="19" name="untitled" type="static">J</K>
        <I background="black" column="0" foreground="black" line="0" name="" type="static"/>
        <E background="black" column="0" foreground="black" line="0" name="" type="static"/>
      </row>
    </table>
    <block background="black" column="73" foreground="white" line="20" name="untitled" type="static">MEER...</block>
    <block action="KEY_PF3" background="black" column="1" data="" foreground="blue" line="21" name="untitled" type="keyword">Einde</block>
    <block action="KEY_PF5" background="black" column="19" data="" foreground="blue" line="21" name="untitled" type="keyword">Vernieuwen</block>
    <block action="KEY_PF6" background="black" column="37" data="" foreground="blue" line="21" name="untitled" type="keyword">Toevoegen</block>
    <block action="KEY_PF12" background="black" column="52" data="" foreground="blue" line="21" name="untitled" type="keyword">Annuleren</block>
    <block action="KEY_PF17" background="black" column="1" data="" foreground="blue" line="22" name="untitled" type="keyword">Begin lijst</block>
    <block action="KEY_PF18" background="black" column="19" data="" foreground="blue" line="22" name="untitled" type="keyword">Einde lijst</block>
  </blocks>
</document>
```


Figure 2 - 516: Subfile extraction rule - Resulting XML

After XSL transformation, thanks to `table` XSL template, it appears webized:

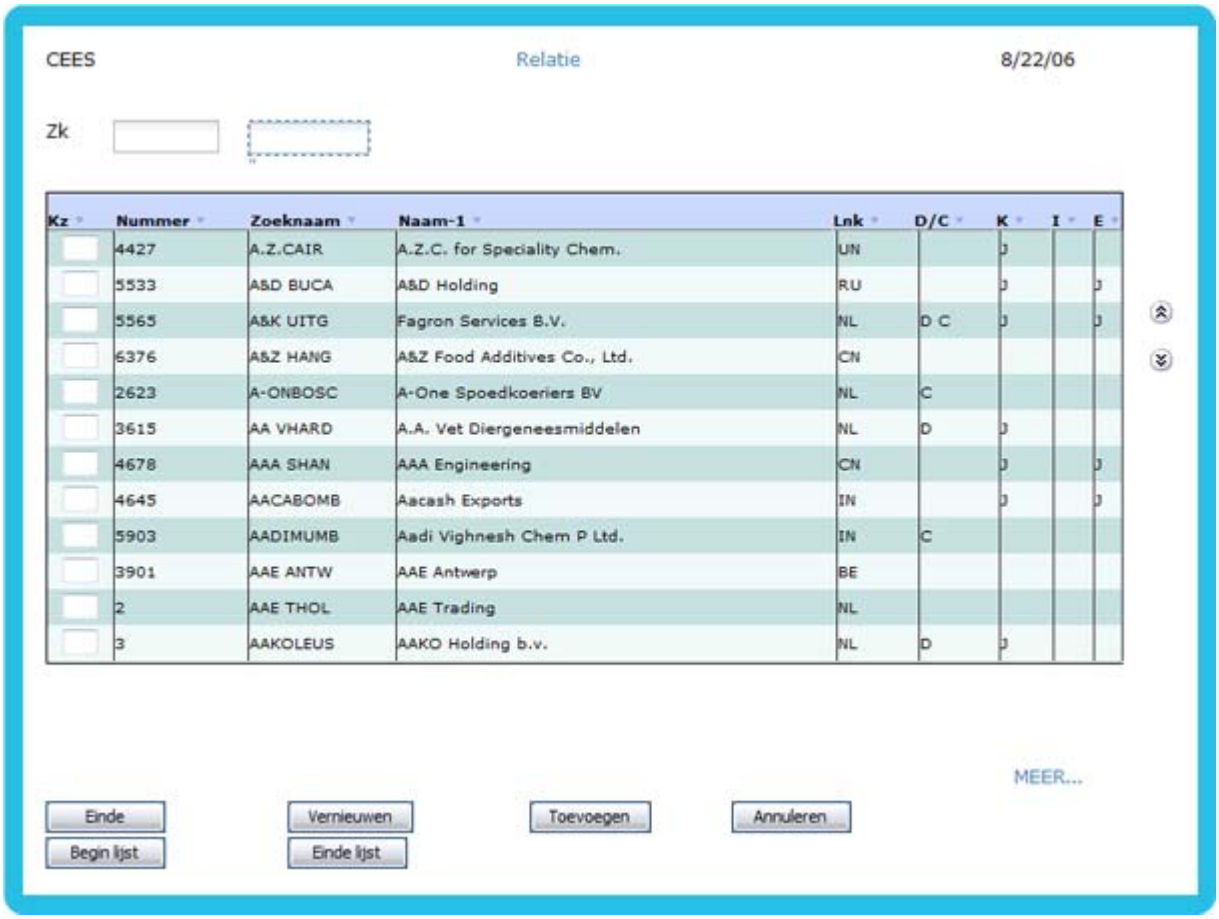


Figure 2 - 517: Subfile extraction rule - Webized page with rule

We can notice the selection fields contextual menu that opens when clicking on a field of the action column:

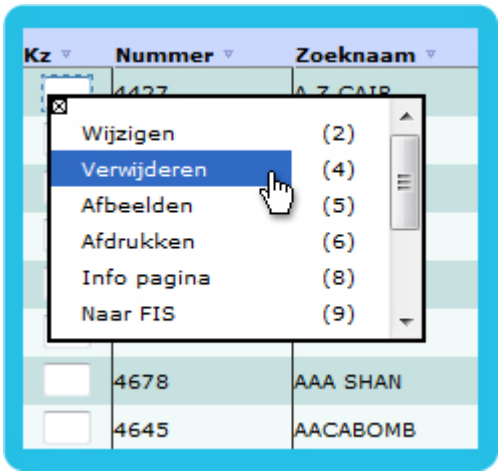


Figure 2 - 518: Subfile extraction rule - Selection field contextual menu

5250 EXTENDED OBJECTS 

OBJECT DESCRIPTION

Automatic `5250` extended objects extraction.

This rule automatically detects `Extended TN5250 NPTUI` objects in mainframe legacy screens.

Unlike standard extraction rules, the *5250 extended objects* rule doens't visually identify objects in displayed screen. In fact, extended objects are well-defined structures and their presence in a screen is embedded inside the `TN5250` stream. One instance of the rule is then enough to correctly detect all NPTUI objects in the screen class.

There are various extended, or NPTUI, elements available in such an application. Usual NPTUI objects are the following:

- **Radio buttons**: they are extracted as XML elements of `choice` type, which are handled by the `choice` XSL template described in the `choice.xsl` file. To change the way choices are displayed in the HTML page, edit this file.
- **Checkboxes**: they are extracted as XML elements of the `checkbox` type, which are handled by the `checkbox` XSL template described in the `checkbox.xsl` file. To change the way checkboxes are displayed in the HTML page, edit this file.
- **Windows**: they are extracted as XML elements of the `panel` type, which are handled by the `panel` XSL template described in the `panel.xsl` file. To change the way panels are displayed in the HTML page, edit this file.
- **Continuous fields**: they are extracted as XML elements of `filed` type, which are handled by the `field` XSL template described in the `field.xsl` file. To change the way fields are displayed in the HTML page, edit this file.
- **Buttons**: they are extracted as XML elements of `keyword` type, which are handled by the `keyword` XSL template described in the `keyword.xsl` file. To change the way keywords are displayed in the HTML page, edit this file.
- **Sliders**: they are extracted as XML elements of `slider` type, which are handled by the `slider` XSL template described in the `slider.xsl` file. To change the way sliders are displayed in the HTML page, edit this file.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Button	boolean	configuration	Activates the button extraction. If set to `true`, the buttons are extracted.
Checkbox	boolean	configuration	Activates the checkbox extraction. If set to `true`, the checkboxes are extracted.
Choice	boolean	configuration	Activates the choice field extraction. If set to `true`, the choice fields are extracted.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Menu	boolean	configuration	Activates the menu extraction. If set to `true`, the menus are extracted.
Radio	boolean	configuration	Activates the radio button extraction. If set to `true`, the radio buttons are extracted.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Scrollbar	boolean	configuration	Activates the scrolling bar extraction. If set to `true`, the scrolling bars are extracted.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Window	boolean	configuration	Activates the window extraction. If set to `true`, the windows are extracted.

VDX GUI COMPONENTS

VIDEOTEX COMMANDS 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

EDIT FIELD 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

MENU 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

BLOCK MANAGEMENT

MERGE BLOCKS 

OBJECT DESCRIPTION

Merges multiple blocks into one or several bigger blocks.

The *Merge blocks* extraction rule:

- looks for series of blocks matching the following pattern: `[any block #1] [separator string block] [any block #2]`
- merges their content in the first block : `['block #1 content' 'separator string' 'block #2 content']`

Separator strings are optional. If not specified, the content of blocks is merged whatsoever, provided that they match rule selection parameters (**Screen zone**, **Presentation attributes**).

By default, the *Merge blocks* extraction rule works on separate lines, meaning that two blocks must be on the same line to be merged. However, the rule can be set so that blocks are merged even if they belong to separate lines. In this case, a line break string can be set.

When two blocks from different lines are merged, this line break string is inserted in their content: `[block nb 1] [separator string block] [block nb 2]`

Once merged, the resulting block is as follows:`['block nb 1 content' 'separator string' 'line break string' 'block nb 2 content']`.

Notes:

- This rule does not add any specific XML attribute. However, the resulting block inherits the attributes (column, line, colors...) from the first merged block of the row.
- Since this rule does not create a new block type, it does not involve any specific XSL stylesheet.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Line separator	String	configuration	Defines the character to add to distinct lines. If **Multiline** property is set to `true`, and if two blocks belonging to separate lines are merged, then the string value of this property is inserted in-between.
Multiline	boolean	configuration	Defines whether the merge should be multiline. Enables to merge blocks located on different lines. If set to `false`, one block will result on each line; if set to `true`, one block will result from all blocks from every lines.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Separator	String	configuration	Defines a separator string, case independent (optional). Merges only blocks separated by a third block, the content of which is equal to the value of the separator string. If this property is not filled, the rule merges matching blocks that are not separated by any blocks.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

Figure 2 - 519: Merge blocks extraction rule - Legacy screen

We notice that the two blocks on the top right corner of the screen (containing the text strings `System` and `DWH01`) are related.

Without the rule, the resulting XML is as follows:

```
<block background="black" column="61" foreground="green" line="1" name="untitled" type="static">System: </block>
<block background="black" column="71" foreground="green" line="1" name="untitled" type="static">DWH01 </block>
```

Figure 2 - 520: Merge blocks extraction rule - Resulting XML without rule

We want to merge the two blocks previously described. To do so in this example, a *Merge blocks* extraction rule is created with the following parameters:

```
Merge blocks [
   screen zone=[x=61, y=1, width=15, height=1]
   attributes=[foreground="green", background="black"]
   separator=""
   multiline=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

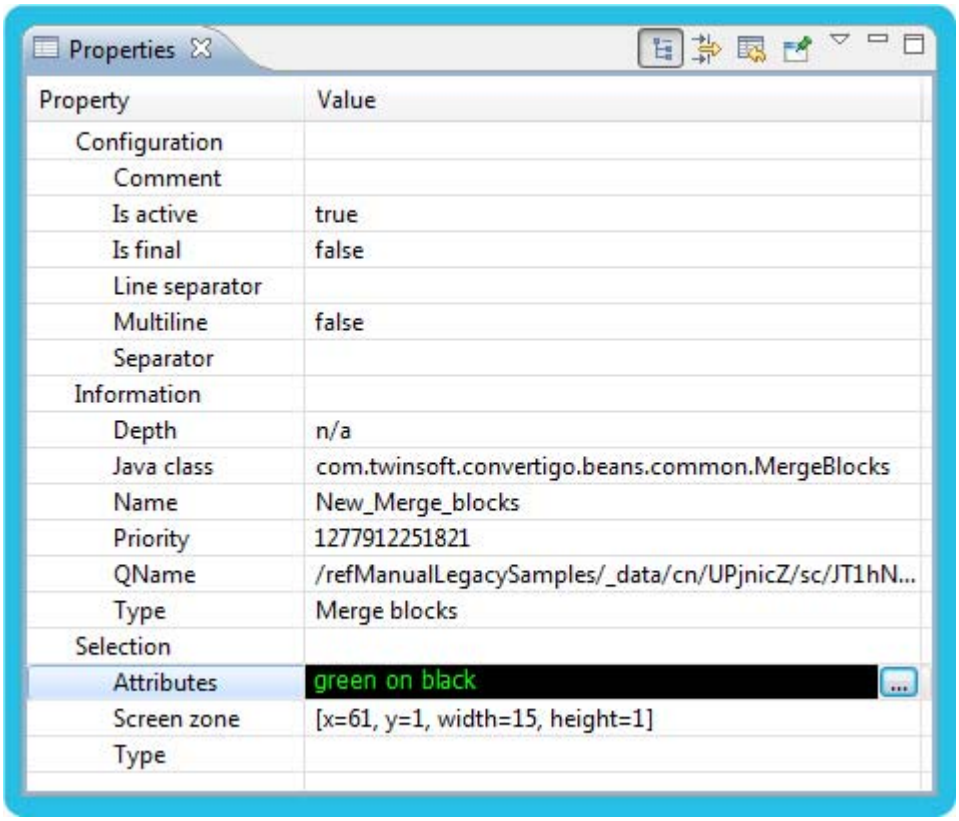


Figure 2 - 521: Merge blocks extraction rule - Configuration example

When the *Merge blocks* extraction rule is executed, it merges the two blocks into one:

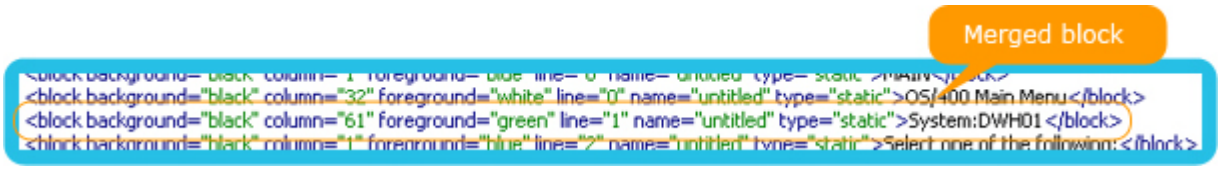


Figure 2 - 522: Merge blocks extraction rule - Resulting XML with rule

DELETE BLOCKS 

OBJECT DESCRIPTION

Removes blocks from a legacy screen, based on presentation attributes or block position.

To simplify and speed up the process time of other rules, removing useless blocks can be an option. It limits the quantity of generated XML data and optimizes network traffic.

By default, the rule is created with the **Type** property set to `[^field]`, meaning that the rule does not apply to blocks of `field` type.

The *Delete blocks* extraction rule is also useful for cleaning up purposes when creating Data integration projects. Data to be kept are tagged using *XML Tagname* extraction rules, tables are tagged using a *Table* or *Subfile* extraction rule, and other resulting blocks are tagged as `block`. They can then be removed by configuring the **Tag name** property to `block`.

The *Delete blocks* extraction rule can also be used to remove portions of a screen not to be displayed (in webization projects).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Property	Type	Category	Description
Content	String	configuration	Defines the regular expression matching the content of block(s) to be removed. Blocks which content corresponds to this regular expression are removed. For example, if set to TEST, all blocks containing "TEST" are removed. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Length	int	configuration	Defines the length of blocks (without trailing and ending spaces) to be removed. `-1` means an unspecified length (the length criterion is not used), `0` means that all blocks containing only spaces are deleted.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Tag name	String	configuration	Defines the tag name of blocks to be removed.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

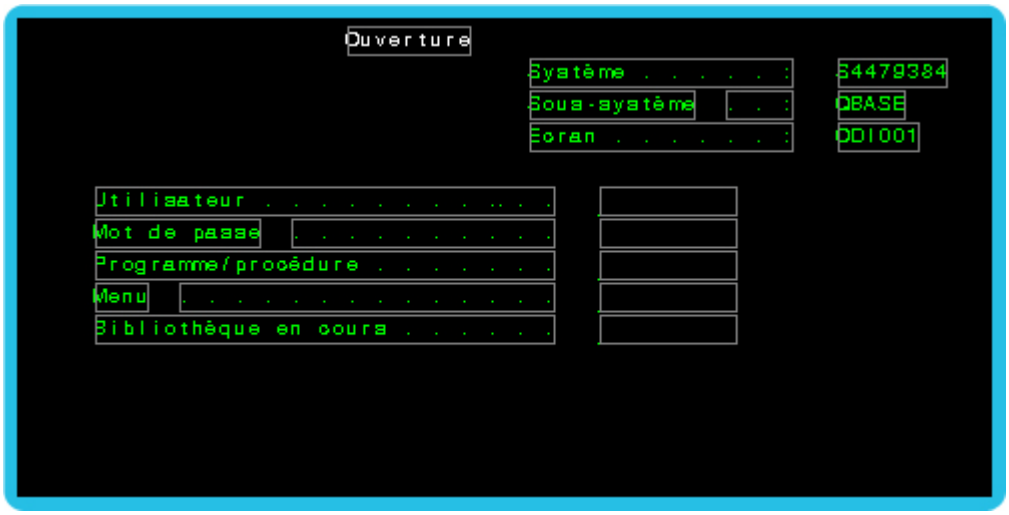


Figure 2 - 523: Delete blocks extraction rule - Legacy Screen

We want to remove useless text located between the screen title ("Ouverture") and login detail. Without a *Delete blocks* extraction rule, the XML resulting from this screen is as follows:

```
<blocks page-number="0">
  <block background="black" column="34" foreground="white" line="0" name="untitled" type="static">Ouverture</block>
  <block background="black" column="47" foreground="green" line="1" name="untitled" type="static">Système :</block>
  <block background="black" column="69" foreground="green" line="1" name="untitled" type="static">54479384</block>
  <block background="black" column="47" foreground="green" line="2" name="untitled" type="static">Sous-système</block>
  <block background="black" column="61" foreground="green" line="2" name="untitled" type="static"> :</block>
  <block background="black" column="69" foreground="green" line="2" name="untitled" type="static">QBASE</block>
  <block background="black" column="47" foreground="green" line="3" name="untitled" type="static">Ecran :</block>
  <block background="black" column="69" foreground="green" line="3" name="untitled" type="static">ODI001</block>
  <block background="black" column="16" foreground="green" line="5" name="untitled" type="static">Utilisateur </block>
```

Figure 2 - 524: Delete blocks extraction rule - Resulting XML without rule

After XSL transformation, they appear webized:

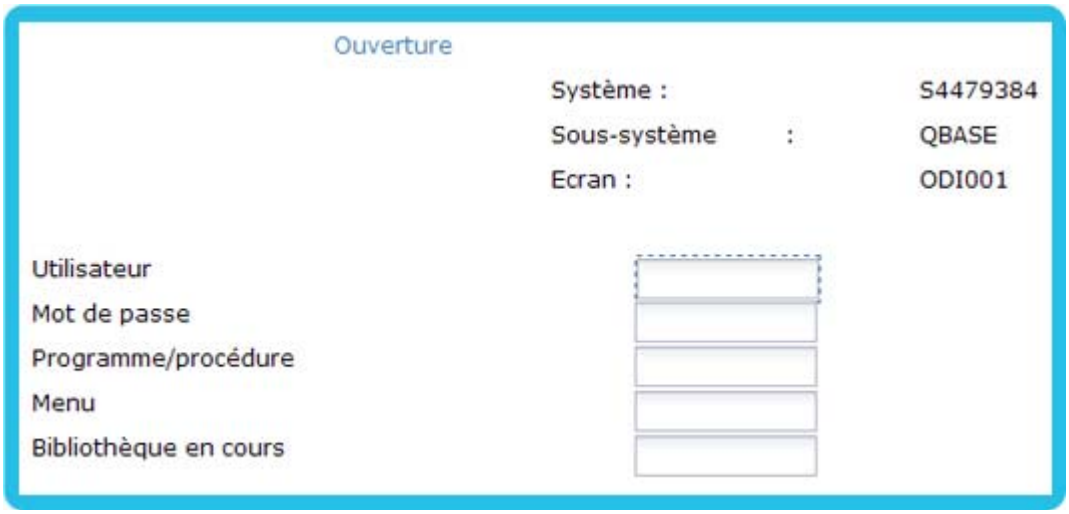


Figure 2 - 525: Delete blocks extraction rule - Webized page without rule

In this example, a *Delete blocks* rule is created with the following parameters:

```
Delete blocks [
   screen zone=[x=47, y=1, width=31, height=3]
   type="static"
   tag name="block"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

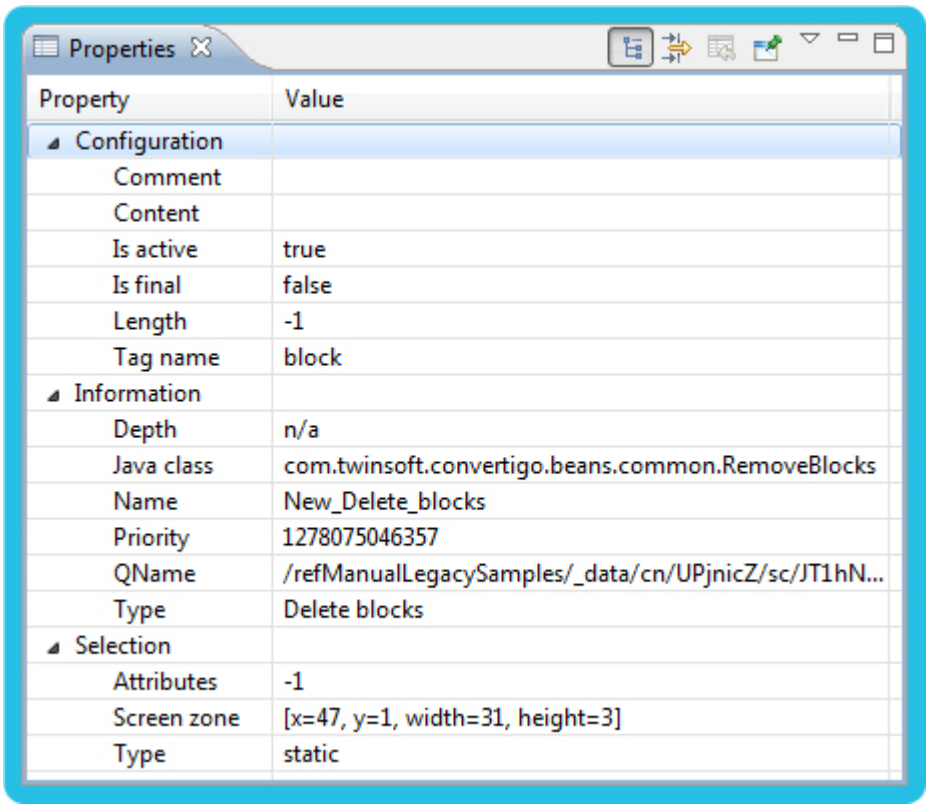


Figure 2 - 526: Delete blocks extraction rule - Configuration example

When the rule is executed, the resulting XML doesn't include anymore the top blocks elements:

```
<blocks page-number="0">
   <block background="black" column="34" foreground="white" line="0" name="untitled" type="static">Ouverture</block>
   <block background="black" column="16" foreground="green" line="5" name="untitled" type="static">Utilisateur </block>
   <block background="black" column="52" foreground="green" hasFocus="true" line="5" name="__field_c52_l5" size="10" type="field"/>
```

Figure 2 - 527: Delete blocks extraction rule - Resulting XML with rule

As a result, these elements don't appear in the dynamically webized screen:

Figure 2 - 528: Delete blocks extraction rule - Webized page with rule

SPLIT BLOCK 

OBJECT DESCRIPTION

Splits blocks depending on defined delimiter characters.

The *Split block* extraction rule allows splitting a block into parts, breaking blocks including a specified delimiter character into smaller ones. This can be useful when other rules need separate blocks to be properly processed.

Notes:

- The *Split block* extraction rule can match any text on a green screen.
- This rule does not add or create new attributes. Resulting blocks inherit most of the attributes from the original splitted block. Only position attributes (i.e. column and line) are updated to match actual position on the screen.
- Since this rule does not create a new block type, it does not involve any specific XSL stylesheet.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Delimiters	String	configuration	Defines a concatenated list of delimiter characters. Caution: Default value for this parameter is a blank space (not easily visible). When setting a new *Split block* rule, make sure to delete this default space before adding new delimiter character(s).
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Keep delimiters	boolean	configuration	Defines if delimiters should remain included in generated sub-blocks. If set to `true`, delimiter characters are kept upon splitting, remaining as new blocks of size 1. Otherwise, delimiter characters are discarded.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

Figure 2 - 529: Split block rule - Legacy Screen

Without the rule, the resulting XML is as follows:

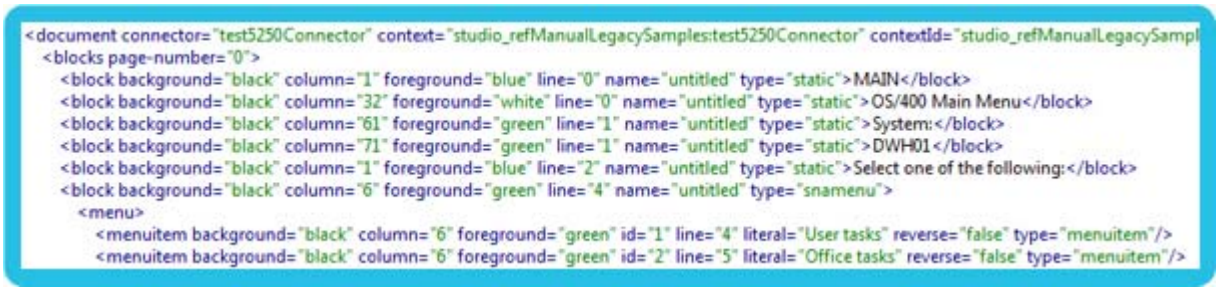


Figure 2 - 530: Split block rule - Resulting XML without rule

In this example, we want to split the title block on "/" character. A *Split block* extraction rule is created with the following parameters:

```
Split block [
   screen zone=[x=0, y=0, width=80, height=1]
   delimiters="/"
   keep delimiters=true
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

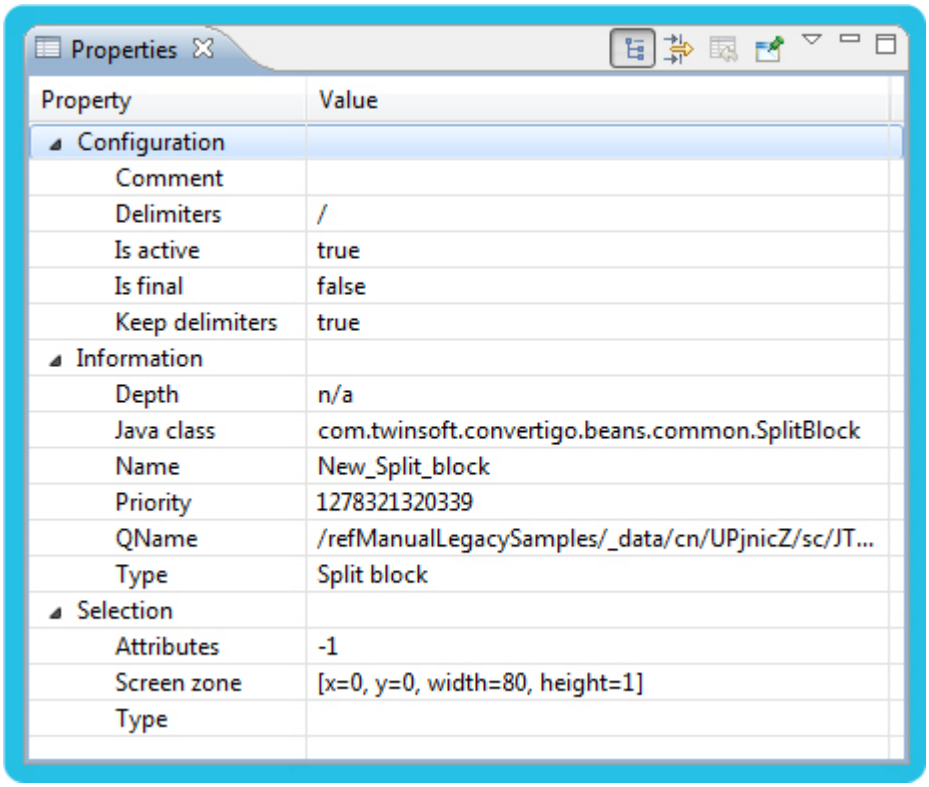


Figure 2 - 531: Split block extraction rule - Configuration example

With the rule, the resulting XML shows the title is splitted into three blocks, one with the text before the "/" character, one with the "/" character, and one with the text after the "/" character:

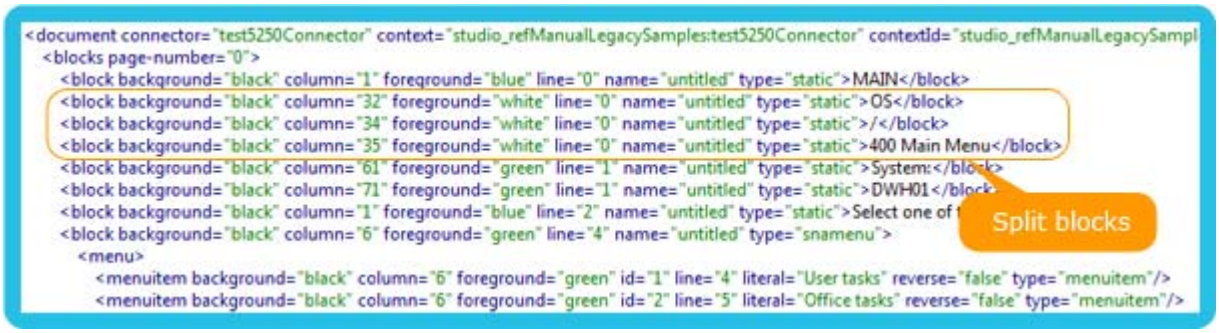


Figure 2 - 532: Split block rule - Resulting XML with rule

TRIM SPACES 

OBJECT DESCRIPTION

Removes heading and trailing space characters.

This rules trims blank spaces in blocks. This can be useful in data integration projects where data extracted from screens contains unwanted heading and trailing spaces.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Left trim	boolean	configuration	Defines whether heading space characters must be removed. If set to `true`, heading space characters are removed.
Right trim	boolean	configuration	Defines whether trailing space characters must be removed. If set to `true`, trailing space characters are removed.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

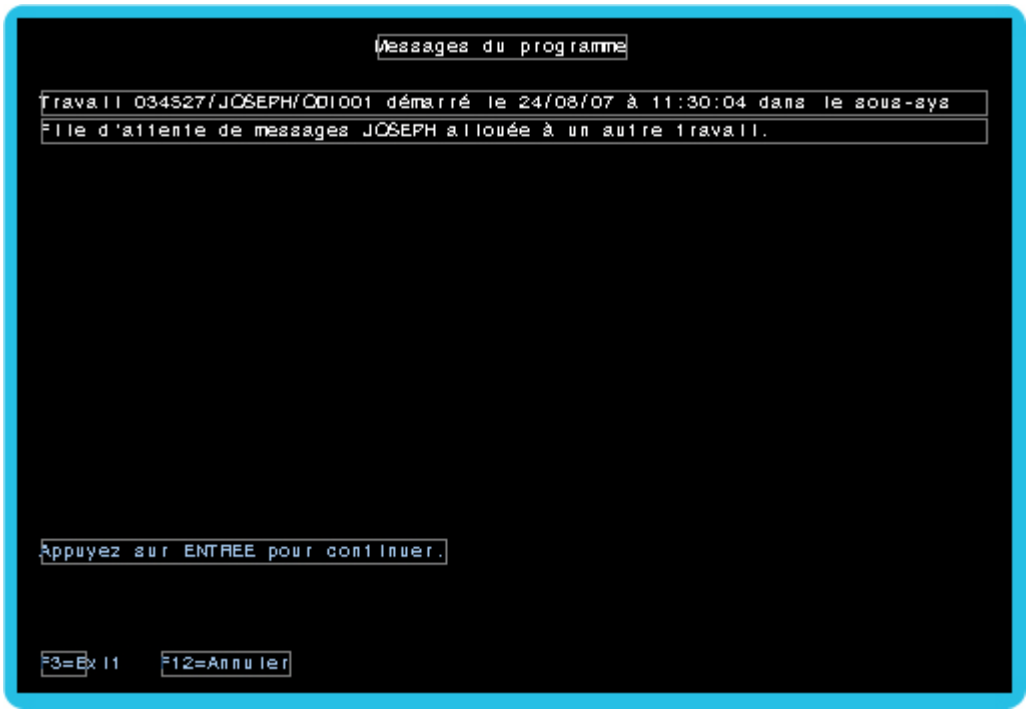


Figure 2 - 533: Trim spaces extraction rule - Legacy screen

We can notice the spaces ath the end of the two lines of the message.

Without rule, the resulting XML is as follows:

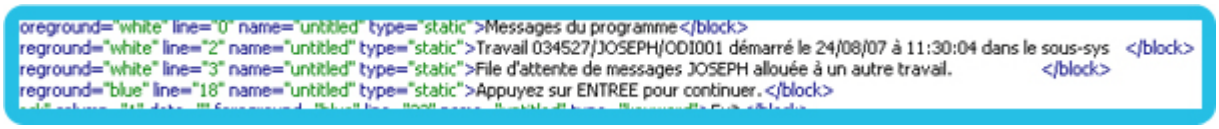


Figure 2 - 534: Trim spaces extraction rule - Resulting XML without rule

A *Trim spaces* extraction rule is created with the following parameters:

```
Trim spaces [
   screen zone=[x=0, y=2, width=80, height=2]
   attributes=[foreground="white", background="black"]
   left trim=false
   right trim=true
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

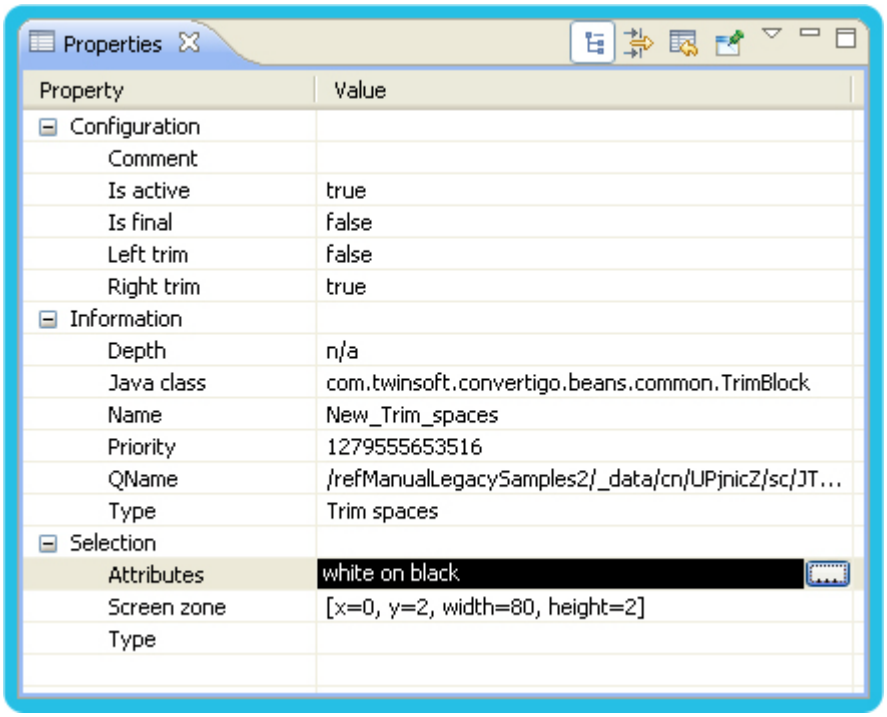


Figure 2 - 535: Trim spaces extraction rule - Configuration example

When the *Trim spaces* extraction rule is executed, the spaces that were present at the end of the two lines are deleted from the blocks:

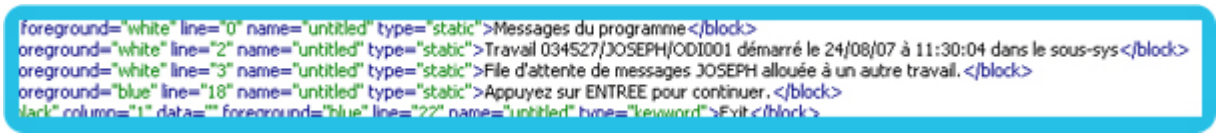


Figure 2 - 536: Trim spaces extraction rule - Resulting XML with rule

MOVE BLOCKS 

OBJECT DESCRIPTION

Moves blocks in a legacy screen.

The *Move blocks* extraction rule moves blocks extracted from a legacy screen to a different target position in the screen (defined through the **Moving layout** property).

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Moving layout	XMLRectangle	configuration	Defines the target screen zone for moved blocks. This property allows to position the moved blocks to a specific area of the screen. The moved blocks will be updated with the specified screen zone values as positioning attributes. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value. These positioning attributes have to be handled by the XSL template rule that displays the blocks.

Property	Type	Category	Description
Relative move	boolean	configuration	Defines whether the move is relative to the original blocks position or not relative. If set to `true`, blocks are moved from their original position to a number of columns and a number of lines from it. If set to `false`, blocks are moved at the exact defined position.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

Figure 2 - 537: Move blocks extraction rule - Legacy screen

Without a *Move blocks* extraction rule, the XML resulting from this screen is as follows:

```
<block background="black" column="34" foreground="white" line="0" name="untitled" type="static">Ouverture</block>
<block background="black" column="47" foreground="green" line="1" name="untitled" type="static">Système :</block>
<block background="black" column="69" foreground="green" line="1" name="untitled" type="static">S4479384</block>
<block background="black" column="47" foreground="green" line="2" name="untitled" type="static">Sous-système</block>
<block background="black" column="61" foreground="green" line="2" name="untitled" type="static">:</block>
<block background="black" column="69" foreground="green" line="2" name="untitled" type="static">QBASE</block>
<block background="black" column="47" foreground="green" line="3" name="untitled" type="static">Ecran :</block>
<block background="black" column="69" foreground="green" line="3" name="untitled" type="static">ODI001</block>
```

Figure 2 - 538: Move blocks extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

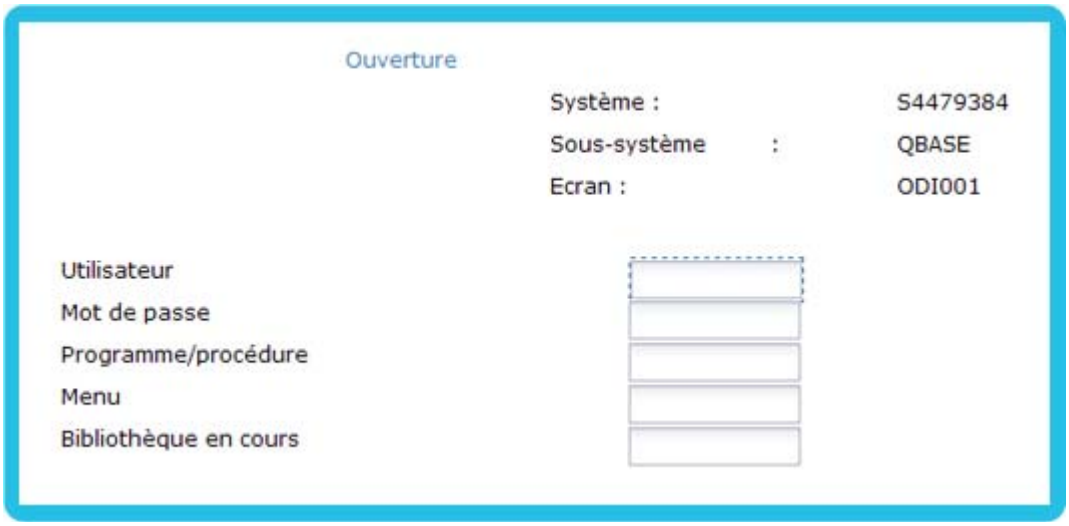


Figure 2 - 539: Move blocks extraction rule - Webized page without rule

In this example, we want to move the top right corner blocks to the top left corner of the screen. A *Move blocks* extraction rule is created with the following parameters:

```
Move blocks [
   screen zone=[x=46, y=1, width=33, height=3]
   moving layout=[x=4, y=1, width=33, height=3]
   relative move=true
```

]

These parameters are edited in the **Properties** view of the Convertigo Studio:

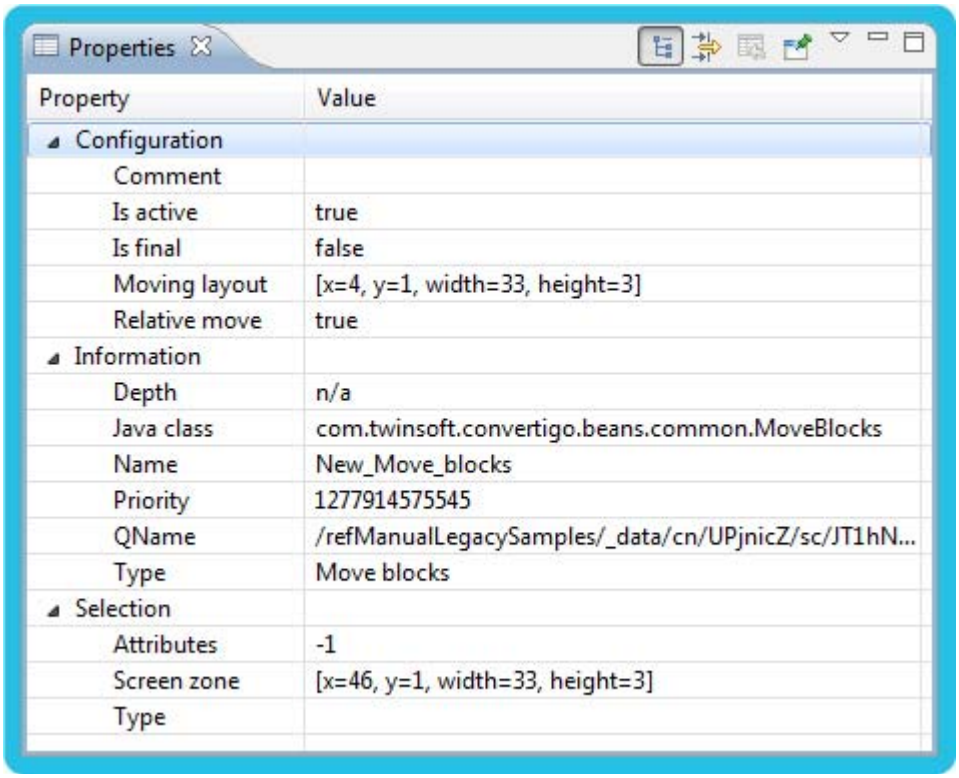


Figure 2 - 540: Move blocks extraction rule - Configuration example

Moving layout and *Screen zone* properties are edited in the **Screen zone** editor:

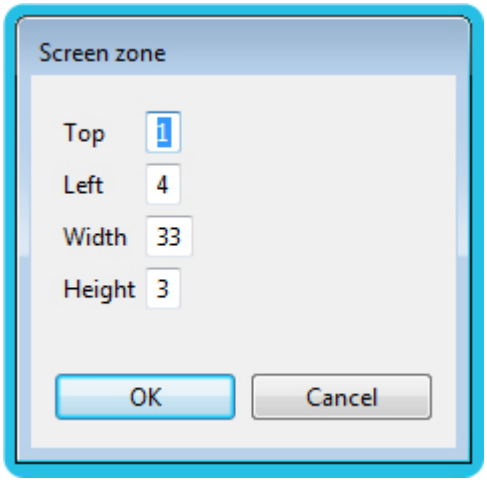


Figure 2 - 541: Move blocks extraction rule - Moving layout property edition

When the rule is executed, the resulting XML includes the new positioning attributes on moved blocks:

```
<blocks page-number="0">
  <block background="black" column="34" foreground="white" line="0" name="untitled" type="static">Ouverture</block>
  <block background="black" column="5" foreground="green" line="1" name="untitled" type="static">Système :</block>
  <block background="black" column="27" foreground="green" line="1" name="untitled" type="static">S4479384</block>
  <block background="black" column="5" foreground="green" line="2" name="untitled" type="static">Sous-système</block>
  <block background="black" column="19" foreground="green" line="2" name="untitled" type="static"> :</block>
  <block background="black" column="27" foreground="green" line="2" name="untitled" type="static">QBASE</block>
  <block background="black" column="5" foreground="green" line="3" name="untitled" type="static">Ecran :</block>
  <block background="black" column="27" foreground="green" line="3" name="untitled" type="static">ODI001</block>
  <block background="black" column="16" foreground="green" line="5" name="untitled" type="static">Utilisateur</block>
```

Figure 2 - 542: Move blocks extraction rule - Resulting XML with rule

After XSL transformation, thanks to XSL templates, the screen appears webized:

Figure 2 - 543: Move blocks extraction rule - Webized page with rule

TEXT HANDLING

LETTER CASE 

OBJECT DESCRIPTION

Changes the letter case.

When extracting information from the screen, this rule changes the data letter case before it is added to the XML document.

Notes:

- The *Letter case* extraction rule does not create new XML attributes.
- Since this rule does not create a new block type, it does not involve any specific XSL stylesheet.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Letter case	int	configuration	Defines the letter case policy. This property helps you choosing the letter case policy to apply on matching blocks between: • `Upper case`: matching blocks text will be transformed to upper case. • `Lower case`: matching blocks text will be transformed to lower case. • `Lower case with first letter upper case`: matching blocks text will be transformed to lower case with first letter upper case.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

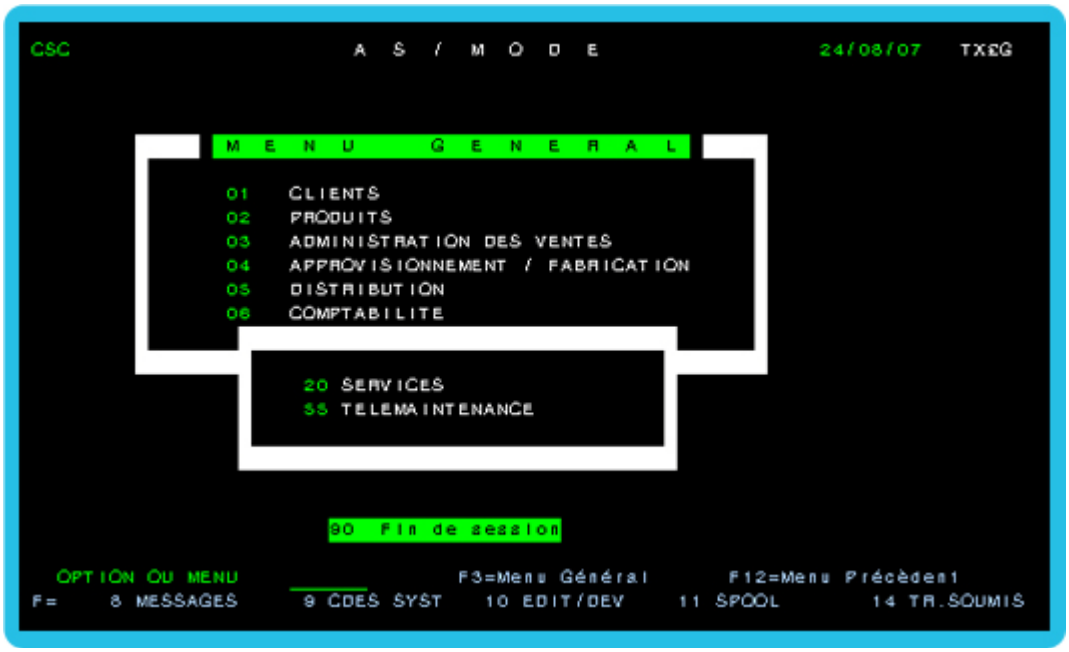


Figure 2 - 544: Letter case extraction rule - Legacy screen

Without the rule, the resulting XML is as follows:

```
<block column="16" line="6" name="untitled" type="static">01</block>
<block column="21" line="6" name="untitled" type="static">CLIENTS</block>
<block column="16" line="7" name="untitled" type="static">02</block>
<block column="21" line="7" name="untitled" type="static">PRODUITS</block>
<block column="16" line="8" name="untitled" type="static">03</block>
<block column="21" line="8" name="untitled" type="static">ADMINISTRATION DES VENTES</block>
<block column="16" line="9" name="untitled" type="static">04</block>
<block column="21" line="9" name="untitled" type="static">APPROVISIONNEMENT / FABRICATION</bloc
<block column="16" line="10" name="untitled" type="static">05</block>
<block column="21" line="10" name="untitled" type="static">DISTRIBUTION</block>
<block column="16" line="11" name="untitled" type="static">06</block>
<block column="21" line="11" name="untitled" type="static">COMPTABILITE</block>
```

Figure 2 - 545: Letter case extraction rule - Resulting XML without rule

A *Letter Case* extraction rule is set on menu items so that output blocks appear in lower case. In this example, the rule is created with the following parameters:

```
ChangeLetterCase [
   screen zone=[x=19, y=5, width=25, height=8]
   letter case="Lower case"
]
```

When the rule is executed, the resulting XML is then as follows:

```
<block column="16" line="6" name="untitled" type="static">01</block>
<block column="21" line="6" name="untitled" type="static">clients</block>
<block column="16" line="7" name="untitled" type="static">02</block>
<block column="21" line="7" name="untitled" type="static">produits</block>
<block column="16" line="8" name="untitled" type="static">03</block>
<block column="21" line="8" name="untitled" type="static">administration des ventes</block>
<block column="16" line="9" name="untitled" type="static">04</block>
<block column="21" line="9" name="untitled" type="static">approvisionnement / fabrication</block>
<block column="16" line="10" name="untitled" type="static">05</block>
<block column="21" line="10" name="untitled" type="static">distribution</block>
<block column="16" line="11" name="untitled" type="static">06</block>
<block column="21" line="11" name="untitled" type="static">comptabilite</block>
```

Figure 2 - 546: Letter case extraction rule - Resulting XML with rule with "Lower case" property value

Then, the rule is set on the same blocks so that output blocks appear in lower case with the first letter in upper case.

In this example, the rule is updated with the following parameters:

```
Letter case [
   screen zone=[x=19, y=5, width=25, height=8]
   letter case="Lower case with first letter upper case"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

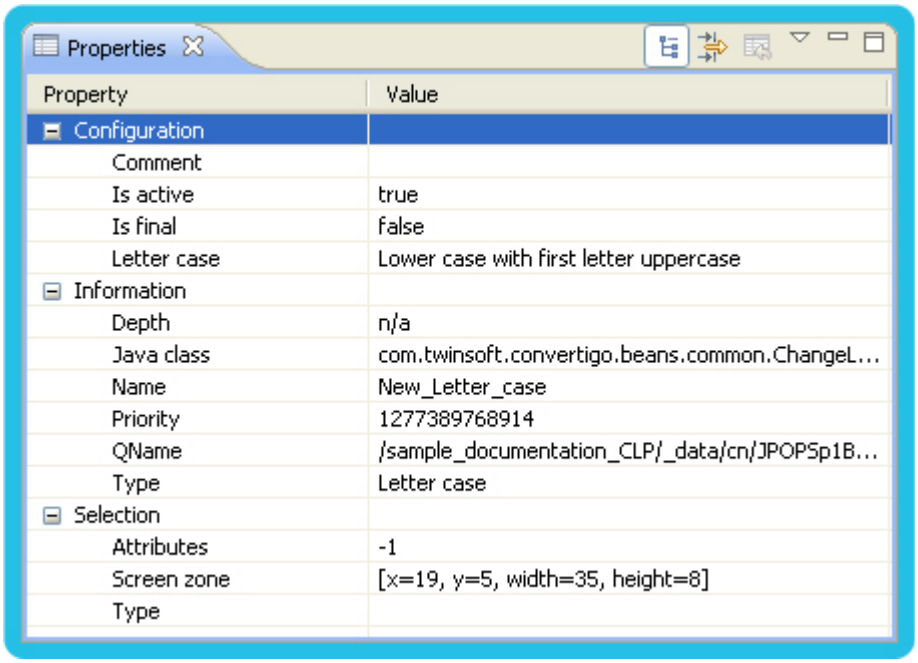


Figure 2 - 547: Letter case extraction rule - Configuration example

When the rule is executed, the resulting XML is then as follows:

```
<block column="16" line="6" name="untitled" type="static">01</block>
<block column="21" line="6" name="untitled" type="static">Clients</block>
<block column="16" line="7" name="untitled" type="static">02</block>
<block column="21" line="7" name="untitled" type="static">Produits</block>
<block column="16" line="8" name="untitled" type="static">03</block>
<block column="21" line="8" name="untitled" type="static">Administration des ventes</block>
<block column="16" line="9" name="untitled" type="static">04</block>
<block column="21" line="9" name="untitled" type="static">Approvisionnement / fabrication</block>
<block column="16" line="10" name="untitled" type="static">05</block>
<block column="21" line="10" name="untitled" type="static">Distribution</block>
<block column="16" line="11" name="untitled" type="static">06</block>
<block column="21" line="11" name="untitled" type="static">Comptabilite</block>
```

Figure 2 - 548: Letter case extraction rule - Resulting XML with "Lower case with first letter upper case" property value

After XSL transformation, thanks to XSL templates, it appears webized:

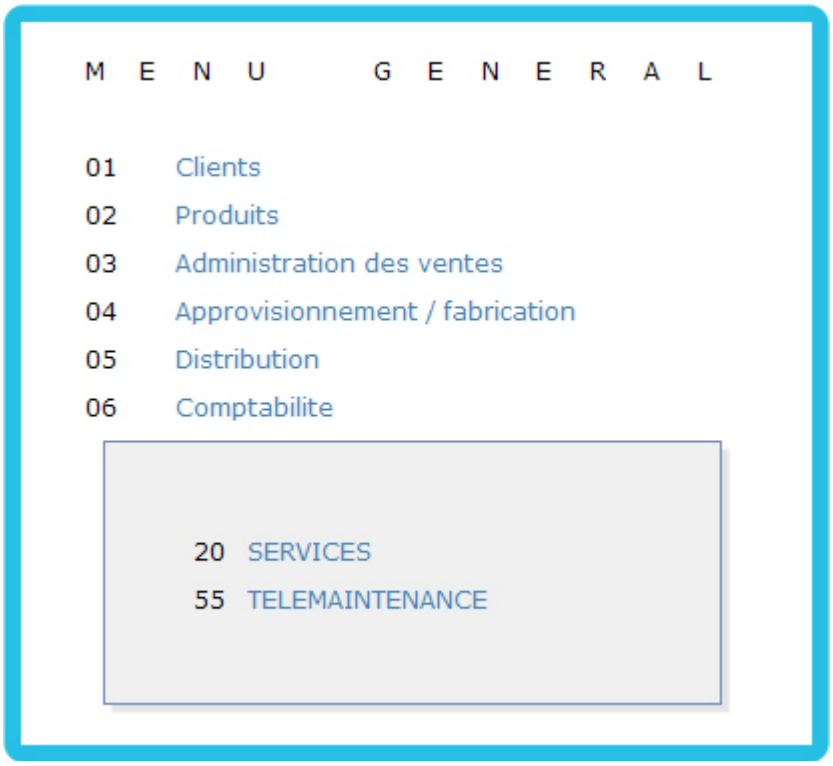


Figure 2 - 549: Letter case extraction rule - Webized page with rule

REPLACE TEXT 

OBJECT DESCRIPTION

Replaces occurrences of a specific text in a block.

The *Replace text* extraction rule replaces texts matching the **Searched text** property in blocks defined by presentation attributes or block position.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Regular expression	boolean	configuration	Defines whether the searched text is a regular expression or not. If set to `true`, the searched text parameter is a regular expression. This allows more flexibility for text replacements. For example, with 5250 platforms, it is frequent to remove trailing dots at the end of field descriptions. To do this, set this parameter to `true` and set the searched for text parameter to: `.(s.)+` If set to `false`, the searched text parameter is a simple text string that will be replaced when present.
Replacement text	String	configuration	Defines the text that will be used to replace the searched text.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Searched text	String	configuration	Defines the text to be replaced or the regular expression defining text to be replaced. Depending on **Regular expression** property, this property contains a text or a regular expression to define the searched text. It is possible to code non ASCII characters using the following syntax: `&#<decimal ASCII code>;`. For example, if the searched text is " " and the replaced text is ` `, the rule will replace all regular spaces in a block with the character of ASCII code 160 representing unbreakable spaces in HTML. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen:

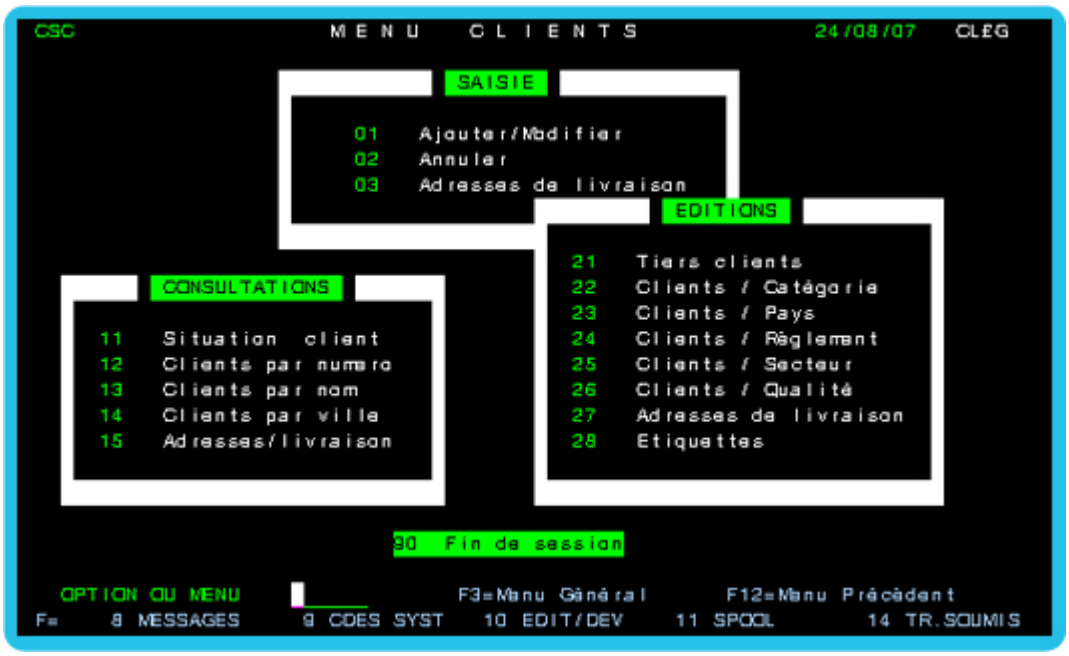


Figure 2 - 550: Replace text extraction rule - Legacy Screen

We notice that a word is often used in this menu: "Clients". We may want to replace this text by another. Without a *Replace text* extraction rule, the XML resulting from this screen is as follows:

```
<block background="black" column="31" foreground="white" line="6" name="untitled" type="static">Adresses de livraison</block>
<block background="green" column="51" foreground="black" line="7" name="untitled" type="static">EDITIONS </block>
<block background="black" column="43" foreground="green" line="9" name="untitled" type="static">21</block>
<block background="black" column="48" foreground="white" line="9" name="untitled" type="static">Tiers clients</block>
<block background="green" column="11" foreground="black" line="10" name="untitled" type="static">CONSULTATIONS </block>
<block background="black" column="43" foreground="green" line="10" name="untitled" type="static">22</block>
<block background="black" column="48" foreground="white" line="10" name="untitled" type="static">Clients / Catégorie</block>
<block background="black" column="43" foreground="green" line="11" name="untitled" type="static">23</block>
<block background="black" column="48" foreground="white" line="11" name="untitled" type="static">Clients / Pays</block>
<block background="black" column="6" foreground="green" line="12" name="untitled" type="static">11</block>
<block background="black" column="11" foreground="white" line="12" name="untitled" type="static">Situation</block>
<block background="black" column="22" foreground="white" line="12" name="untitled" type="static">client</block>
<block background="black" column="43" foreground="green" line="12" name="untitled" type="static">24</block>
<block background="black" column="48" foreground="white" line="12" name="untitled" type="static">Clients / Règlement</block>
<block background="black" column="6" foreground="green" line="13" name="untitled" type="static">12</block>
<block background="black" column="11" foreground="white" line="13" name="untitled" type="static">Clients par numero</block>
<block background="black" column="43" foreground="green" line="13" name="untitled" type="static">25</block>
<block background="black" column="48" foreground="white" line="13" name="untitled" type="static">Clients / Secteur</block>
<block background="black" column="6" foreground="green" line="14" name="untitled" type="static">13</block>
<block background="black" column="11" foreground="white" line="14" name="untitled" type="static">Clients par nom</block>
<block background="black" column="43" foreground="green" line="14" name="untitled" type="static">26</block>
<block background="black" column="48" foreground="white" line="14" name="untitled" type="static">Clients / Qualité</block>
<block background="black" column="6" foreground="green" line="15" name="untitled" type="static">14</block>
<block background="black" column="11" foreground="white" line="15" name="untitled" type="static">Clients par ville</block>
<block background="black" column="43" foreground="green" line="15" name="untitled" type="static">27</block>
<block background="black" column="48" foreground="white" line="15" name="untitled" type="static">Adresses de livraison</block>
<block background="black" column="6" foreground="green" line="16" name="untitled" type="static">15</block>
<block background="black" column="11" foreground="white" line="16" name="untitled" type="static">Adresses/livraison</block>
<block background="black" column="43" foreground="green" line="16" name="untitled" type="static">28</block>
<block background="black" column="48" foreground="white" line="16" name="untitled" type="static">Etiquettes</block>
<block background="green" column="29" foreground="black" line="20" name="untitled" type="static">90</block>
```

Figure 2 - 551: Removing of Blocks extraction rule - Resulting XML without rule

After XSL transformation, the screen appears webized:

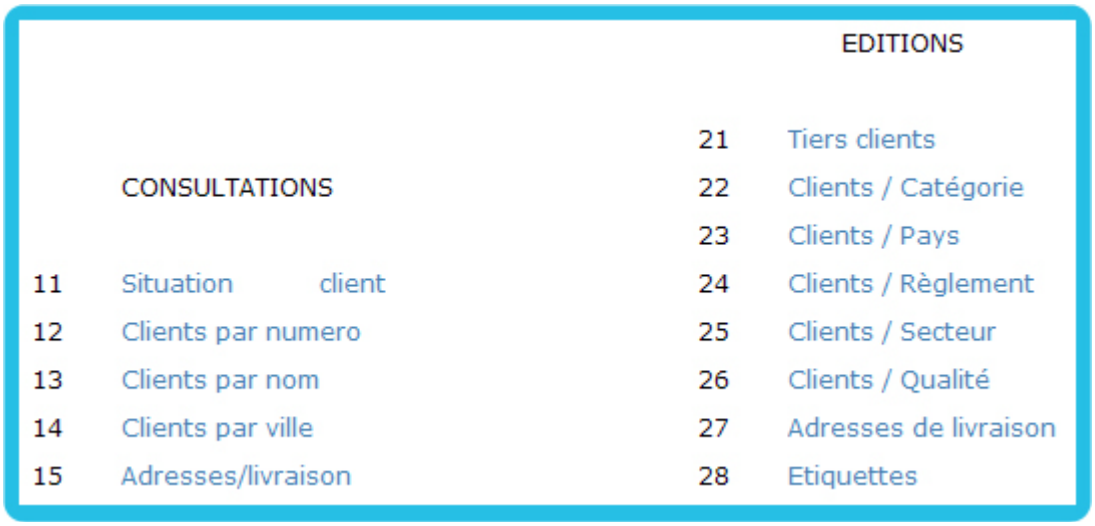


Figure 2 - 552: Replace text extraction rule - Webized page without rule

In this example, a *Replace text* extraction rule is created with the following parameters:

```
Replace text [
   screen zone=[x=4, y=2, width=67, height=16]
   attributes=[foreground="white", background="black"]
   regular expression=true
   searched text="(C|c)lients?"
   replacement text="Customer(s)"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

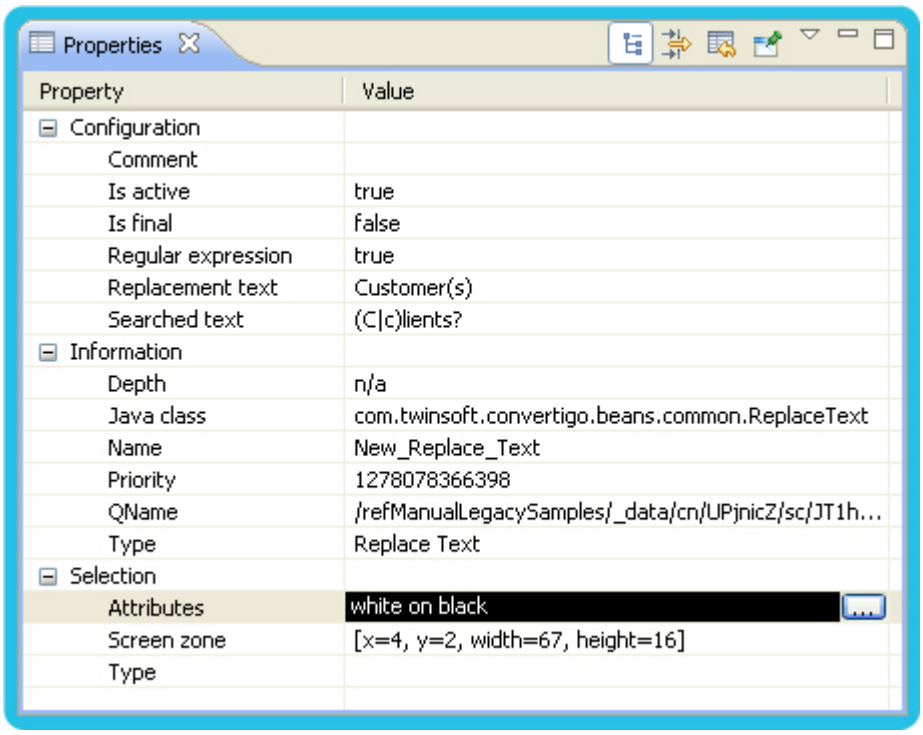


Figure 2 - 553: Replace text extraction rule - Configuration example

When the rule is executed, the resulting XML includes the string replacements in all matching

elements:

```
<block background="black" column="31" foreground="white" line="6" name="untitled" type="static">Adresses de livraison</block>
<block background="green" column="51" foreground="black" line="7" name="untitled" type="static">EDITIONS </block>
<block background="black" column="43" foreground="green" line="9" name="untitled" type="static">21</block>
<block background="black" column="48" foreground="white" line="9" name="untitled" type="static">Tiers Customer(s)</block>
<block background="green" column="11" foreground="black" line="10" name="untitled" type="static">CONSULTATIONS </block>
<block background="black" column="43" foreground="green" line="10" name="untitled" type="static">22</block>
<block background="black" column="48" foreground="white" line="10" name="untitled" type="static">Customer(s) / Catégorie</block>
<block background="black" column="43" foreground="green" line="11" name="untitled" type="static">23</block>
<block background="black" column="48" foreground="white" line="11" name="untitled" type="static">Customer(s) / Pays</block>
<block background="black" column="6" foreground="green" line="12" name="untitled" type="static">11</block>
<block background="black" column="11" foreground="white" line="12" name="untitled" type="static">Situation</block>
<block background="black" column="22" foreground="white" line="12" name="untitled" type="static">Customer(s)</block>
<block background="black" column="43" foreground="green" line="12" name="untitled" type="static">24</block>
<block background="black" column="48" foreground="white" line="12" name="untitled" type="static">Customer(s) / Règlement</block>
<block background="black" column="6" foreground="green" line="13" name="untitled" type="static">12</block>
<block background="black" column="11" foreground="white" line="13" name="untitled" type="static">Customer(s) par numero</block>
<block background="black" column="43" foreground="green" line="13" name="untitled" type="static">25</block>
<block background="black" column="48" foreground="white" line="13" name="untitled" type="static">Customer(s) / Secteur</block>
<block background="black" column="6" foreground="green" line="14" name="untitled" type="static">13</block>
<block background="black" column="11" foreground="white" line="14" name="untitled" type="static">Customer(s) par nom</block>
<block background="black" column="43" foreground="green" line="14" name="untitled" type="static">26</block>
<block background="black" column="48" foreground="white" line="14" name="untitled" type="static">Customer(s) / Qualité</block>
<block background="black" column="6" foreground="green" line="15" name="untitled" type="static">14</block>
<block background="black" column="11" foreground="white" line="15" name="untitled" type="static">Customer(s) par ville</block>
<block background="black" column="43" foreground="green" line="15" name="untitled" type="static">27</block>
<block background="black" column="48" foreground="white" line="15" name="untitled" type="static">Adresses de livraison</block>
<block background="black" column="6" foreground="green" line="16" name="untitled" type="static">15</block>
<block background="black" column="11" foreground="white" line="16" name="untitled" type="static">Adresses/livraison</block>
<block background="black" column="43" foreground="green" line="16" name="untitled" type="static">28</block>
<block background="black" column="48" foreground="white" line="16" name="untitled" type="static">Etiquettes</block>
<block background="green" column="29" foreground="black" line="20" name="untitled" type="static">80</block>
```

Figure 2 - 554: Replace text extraction rule - Resulting XML with rule

After XSL transformation, thanks to XSL templates, the screen appears webized:

Figure 2 - 555: Replace text extraction rule - Webized page with rule

TRANSLATE TEXT 

OBJECT DESCRIPTION

Translates blocks of text according to a given dictionary.

This rule translates the blocks on which it is applied using a dictionary file that is defined by the **Dictionary** property.

The translation is applied block by block, meaning that the block text content is searched as key in the dictionary to find its translation. Blocks have to be split according to dictionary entries.

In order to manage several languages, the dictionary file name can automatically be extended with a "lang" suffix which value is retrieved from the `lang` attribute of the document element of the output XML.

The `lang` attribute of the document element is set by Convertigo using the `__lang` reserved parameter value. Once the `__lang` reserved parameter is received, the context keeps and re-uses this value in every other transaction/sequence output in the same context.

You can also update this attribute by script in a transaction's core. To do so, you can use the following code in one of its handlers:

```
dom.getDocumentElement().setAttribute("lang", "en-us");
```

The dictionary files should always be created with a name of the following form `<baseName>_<lang>.txt`, one file by needed language.

One dictionary file should define all text matches between the original application language and the output language. It should be written using the following format:

- the original text/word on a first line,
- the translated text/word on a second line,
- an empty line as separator,
- etc.

Not found texts/words will appear in the Convertigo engine logs and may be automatically listed in an orphans file, depending on the **Generate orphans list** property value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Dictionary base path	String	configuration	Base path used to retrieve the dictionary file read for the translations. This path is either absolute or relative to Convertigo environment. Relative paths starting with: • `./` are relative to Convertigo workspace, • `.//` are relative to current project folder. The dictionary file path includes the path of the folder where the dictionary file can be found, the base name of the file itself, the language and the file extension. It should be of the following form `<basePath><baseName>_<lang>.txt` with: • the `<basePath>` being the folder where the dictionary file can be found, • the `<baseName>` being the dictionary file base name, • the `<lang>` being overwritten by the `lang` attribute value of the document element of the output XML. Note: This property value should only contain `<basePath><baseName>`, as the language and file extension will be automatically added.
Encoding	String	configuration	Defines the encoding used in the dictionary files. Default value for encoding is `UTF-8`.
Generate orphans list	boolean	configuration	If true, unknown texts/words are written in a `<basePath><baseName>_<lang>_orphans.txt` file. Unknown texts/words are block text contents that are not found in the dictionary. This property can be activated in development environment to help the developer finding the dictionary problems or project inconsistencies. Note: This should be disabled in production environment to preserve resources.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

OTHERS

TAG NAME 

OBJECT DESCRIPTION

Names the selected block(s) with a tag name.

The *Tag name* extraction rule is mostly used in data integration projects. By default, all blocks are given the `block` tag name.

Notes:

- The *Tag name* extraction rule does not create any XML attribute.
- Since this rule does not create a new block type, it does not involve any specific XSL stylesheet.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Field history	boolean	configuration	Saves the field values that the user has entered. Applies to `field` type elements (input fields) only. If set to `true`, any data entered in an input field will be saved in a local history so that when the user inputs data, he can search it and choose from saved data.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description

Label policy	int	configuration	Defines the labeling policy.
			Can take one of the following values:
			• `Explicit`: the tag is named after the **XML tag name** property value
			• `From previous block`: the tag is named after the previous block content. Can be useful in form-like screen to automatically tag multiple data fields with the label preceding the field. For example:
			`Label-01 DATA-01`
			`Label-02 DATA-02`
			will be automatically tagged as:
			`<Label-01>DATA-01<Label-01>`
			`<Label-02>DATA-02<Label-02>`
			• `From next block`: the tag will be set from the next block content

Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.

Property	Type	Category	Description
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
XML tag name	String	configuration	Defines the new XML tag name. If this property is empty, the tag name is automatically found according to the label policy.

EXAMPLES

If we consider the following legacy screen:

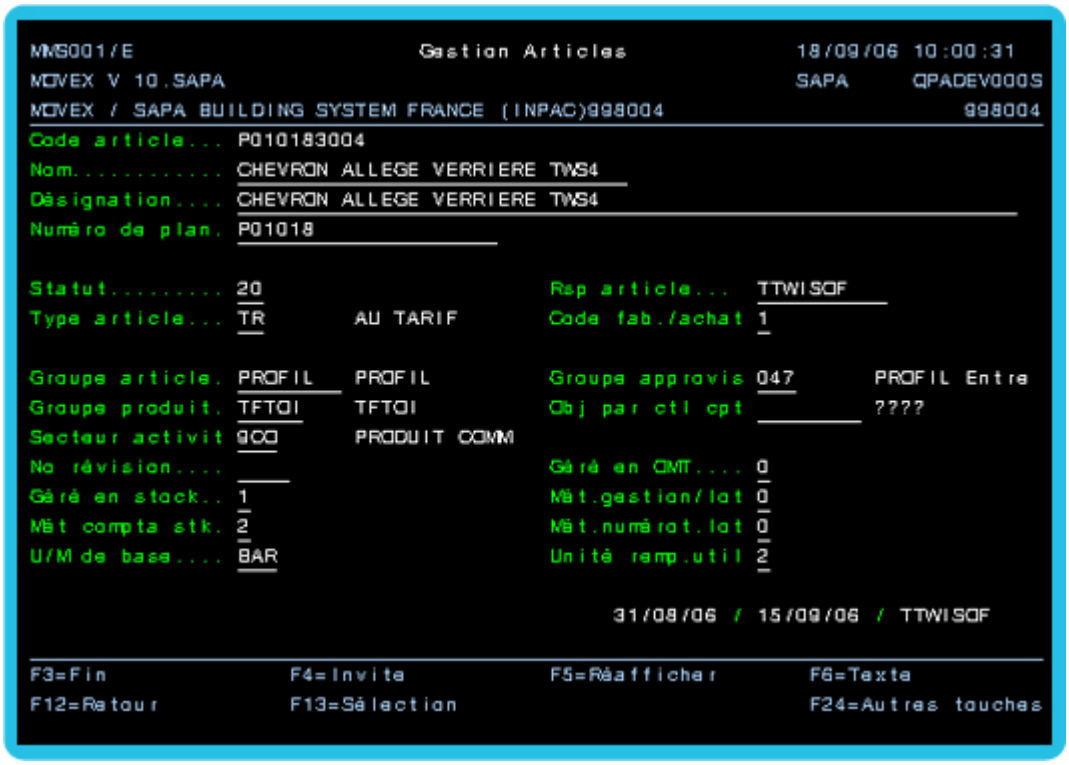


Figure 2 - 556: Tag Name extraction rule - Legacy screen

Without rule, the resulting XML is as follows:

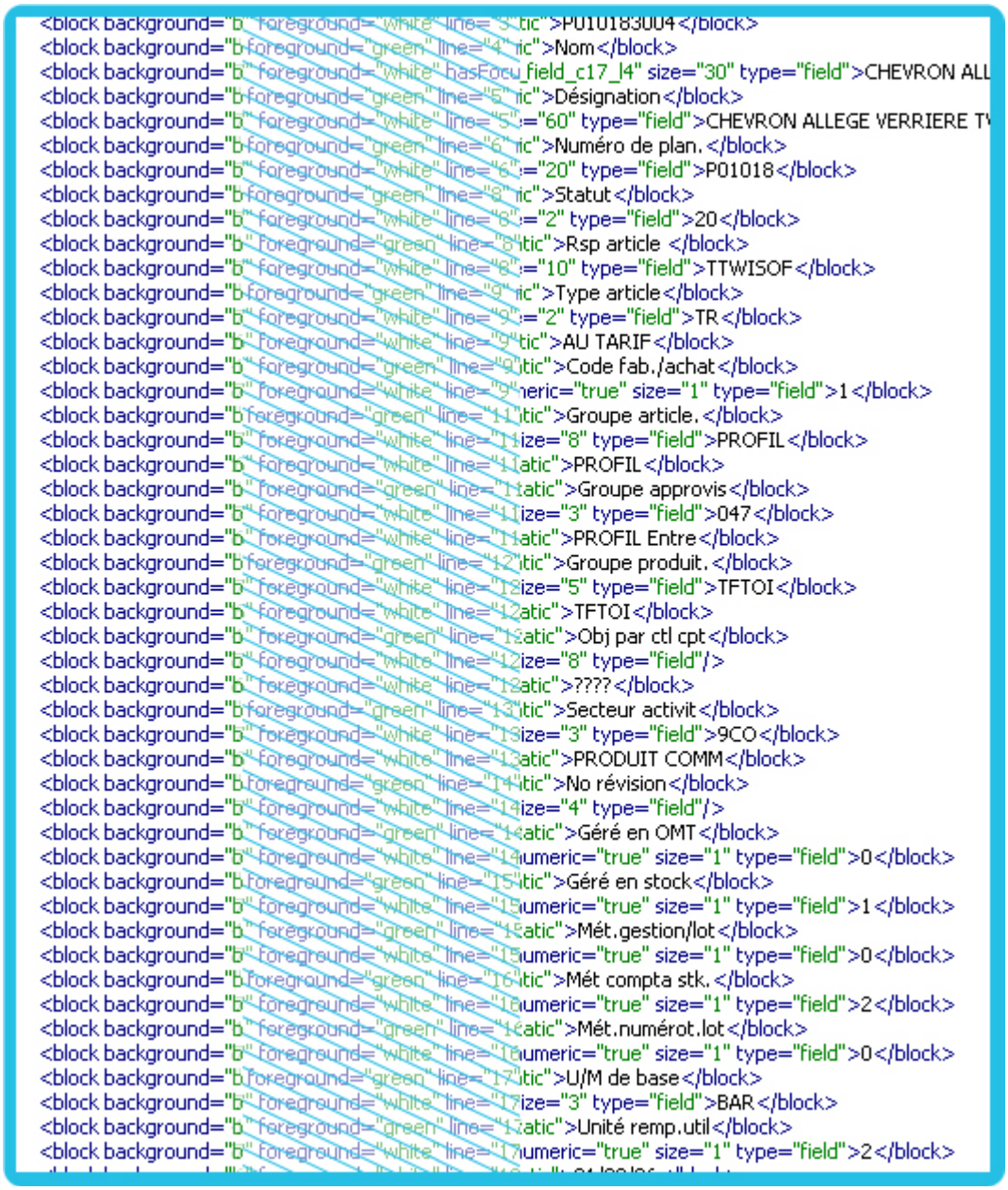


Figure 2 - 557: Tag Name extraction rule - Resulting XML without rule

After having selected a zone in the legacy screen, we apply a *Tag name* extraction rule to the selected zone, with the "input" tag name:

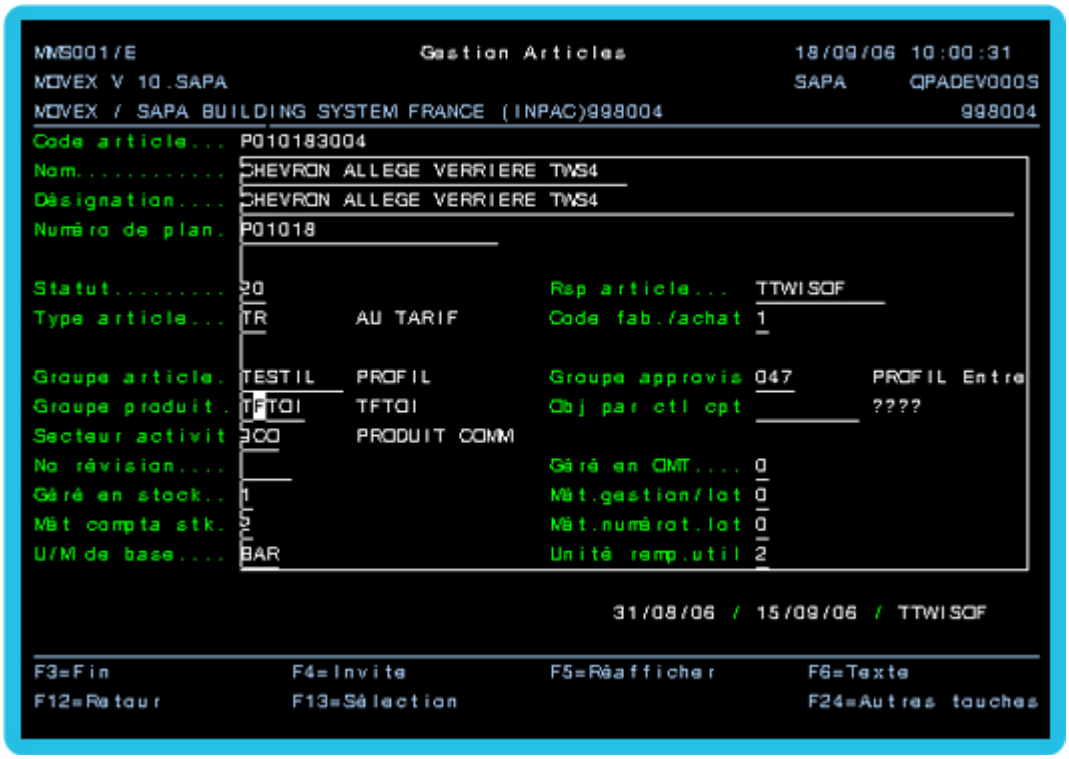


Figure 2 - 558: Tag Name extraction rule - Selected zone

A *Tag name* extraction rule is created with the following parameters:

```
Tag name [
   screen zone=[x=16, y=3, width=63, height=15]
   attributes=[foreground="white", background="black",
   decoration="underlined"]
   label policy=Explicit
   XML tag name="input"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

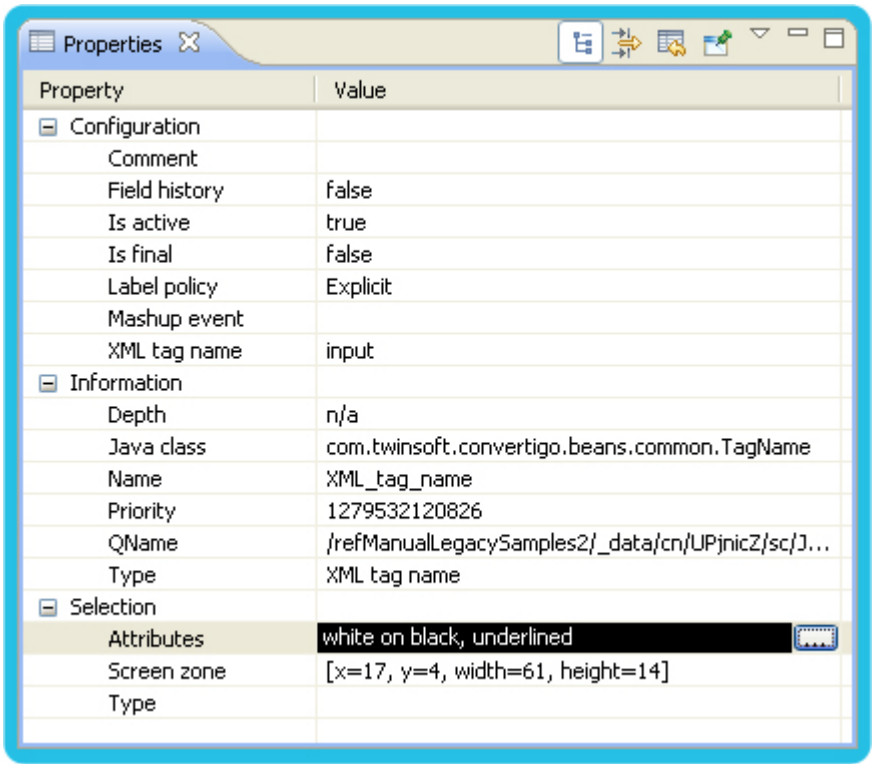


Figure 2 - 559: Tag name extraction rule - Configuration example

We will change some parameters to specify the rule. Here are a few examples of rule usages changing its parameters:

1 The **Label policy** parameter is set to `Explicit` and the **XML Tag name** property to `input`, we add the `field` type to **Type** property:

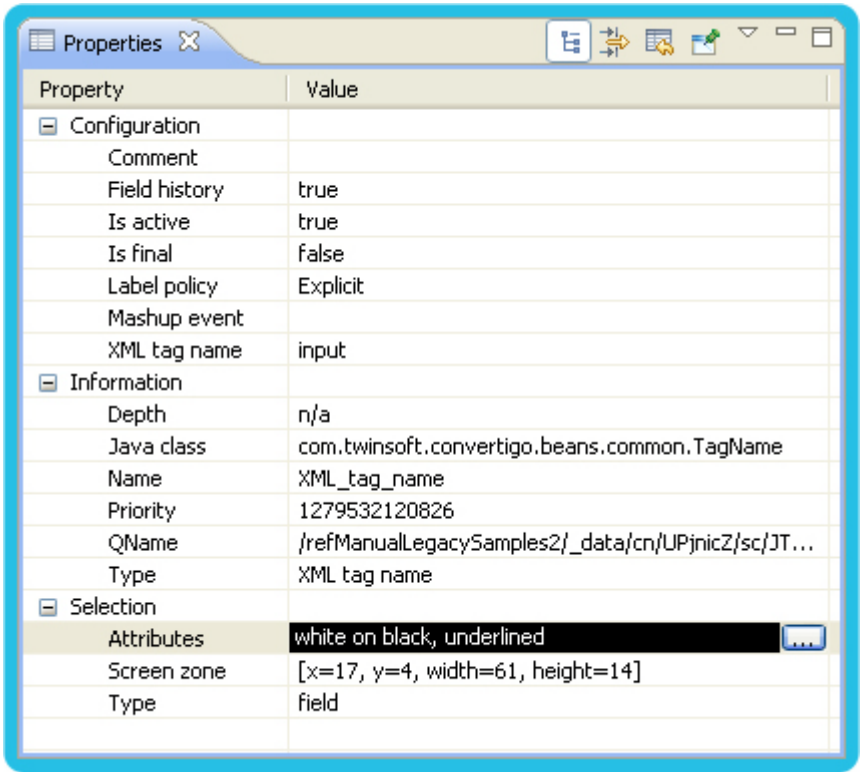


Figure 2 - 560: Tag name extraction rule - Configuration example

As a result, all blocks included in the selected area (see Figure 2 - 558) are tagged with the specified tag name:

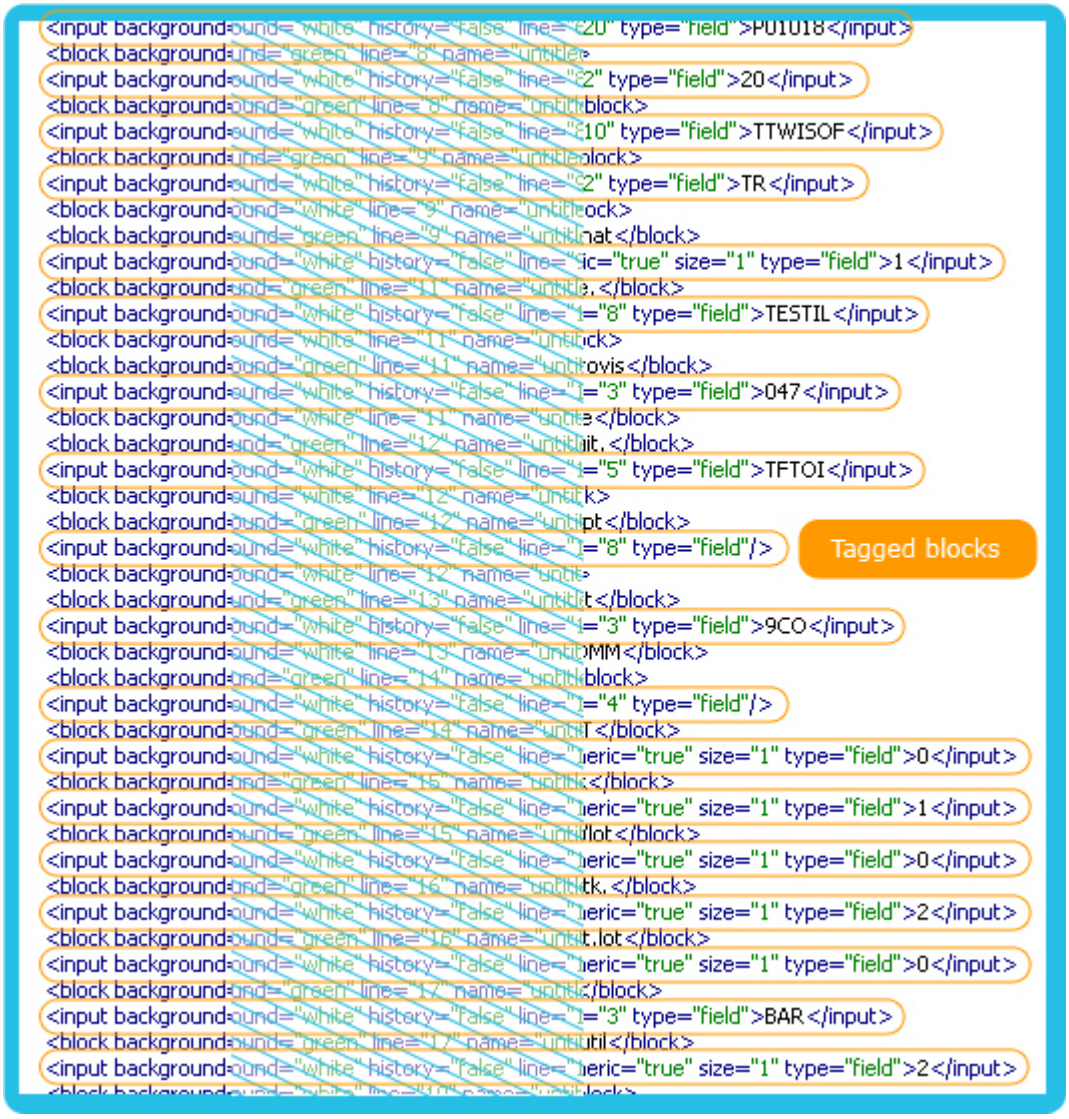


Figure 2 - 561: Tag Name extraction rule - Resulting XML with rule (Label policy as Explicit)

2 The **Label policy** parameter is set to `From previous block` and **XML Tag name** and **Type** properties are reset:

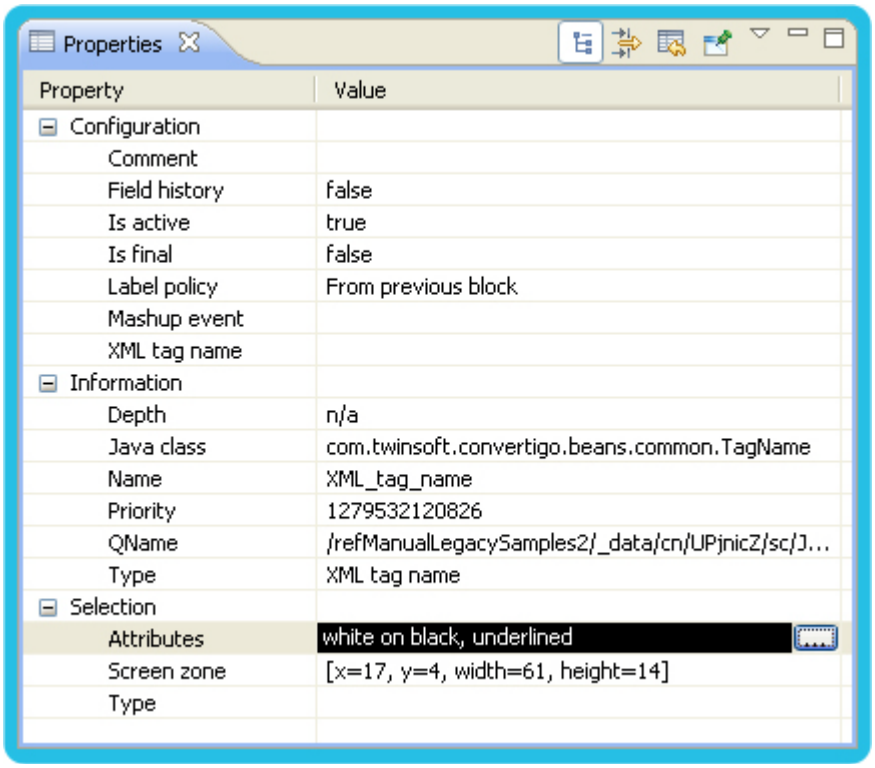


Figure 2 - 562: Tag name extraction rule - Configuration example

As a result, blocks included in the selected area (see Figure 2 - 558) are tagged after the name of the preceding block:

Figure 2 - 563: Tag Name extraction rule - Resulting XML with rule (Label policy as From previous block)

ATTRIBUTE (LEGACY) 

OBJECT DESCRIPTION

Adds an XML attribute to the selected blocks.

The blocks are selected based on usual screen selection properties. The *Attribute* extraction rule is useful for adding attributes to blocks and have it processed by a specific template.

This rule can add another XML attribute than the one specified in the rule parameters, when the **mashup event** property is filled (see **mashup event** property description).

Notes:

- This rule does not involve any specific template.
- If the attribute exists in a matching block, its value is replaced by the new one.
- This rule applies on any block.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Attribute name	String	configuration	Defines the name of the added attribute.
Attribute value	String	configuration	Defines the value of the added attribute.
Attributes	int	selection	Defines the presentation attributes on which the rule applies, i.e. the rule applies on blocks matching these presentation attributes. This property allows to configure the rule so that it applies only to parts of screens having specific attributes, for example green text on black background. Presentation attributes to configure are : • **Color**: `Foreground` color, `Background` color, to choose in a list of predefined colors or "not to take into account". • **Decoration**: `bold`, `reverse`, `underlined`, `blink`, for each decoration choose between "with the decoration", "normal" (i.e. without the decoration), or "not to take into account".
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Is final	boolean	configuration	Defines if the extraction is final, i.e. whether pending extraction rules should try to match on the current extraction rule matching blocks. If set to `true`, once the rule applies on a matching block, Convertigo doesn't apply the following rules on this block. This can be used to prevent a block from being modified by other rules.

Property	Type	Category	Description
Mashup event	String	configuration	Defines mashup events dispatched on click. Mashup events can be of two types: • Calling directly a transaction or a sequence in Convertigo, • Launching an event in Mashup Composer. **Mashup event** property allows to define a combination of one direct call to a Convertigo transaction or sequence and/or one launch of Mashup Composer event. Filling this property adds a `mashup_event` attribute to the block, containing the previous combination in a JSON syntax of one of the following formats: • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"}}` for a transaction call only, • `{"requestable":{"__sequence":"<sequence name>"}}` for a sequence call only, • {{Computer}}{"event":"<event name>"}{{Computer}} for a mashup event only, • `{"requestable":{"__transaction":"<transaction name>","__connector":"<connector name>"},"event":"<event name>"}` for a transaction call and a mashup event, • `{"requestable":{"__sequence":"<sequence name>"},"event":"<event name>"}` for a sequence call and a mashup event. This `mashup_event` attribute and its content have to be handled by the XSL file applying at the end of the transaction to generate a real Convertigo call and/or Mashup Composer event on click on the displayed object.
Screen zone	XMLRectangle	selection	Defines the screen zone on which the rule applies, i.e. the rule applies on blocks completely contained in this screen area. This property allows to configure the rule so that it applies only to areas of screens. All blocks found within the specified perimeter are matching this screen zone and can be processed by the rule. The screen area is defined through four coordinates: • x (area left corner), • y (area upper corner), • w (area width), • h (area height). All values are given in characters, with the upper left corner being (x=0, y=0). `-1` represents an undefined value: `(x=-1, y=-1, w=-1, h=-1)` is an undefined area representing the whole screen, i.e. all blocks, whatever their coordinates, are matching this screen zone and can be processed by the rule.

Property	Type	Category	Description
Type	String	selection	Defines, using a regular expression, to which block types the rule applies. For example, if set to: • `static`, the rule applies to blocks of `static` type only. • `static | field`, the rule applies to blocks of `static` or `field` type only. • `[^field]`, the rule applies to all but `field` type blocks. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.

EXAMPLES

If we consider the following legacy screen, with its title block to be extracted:

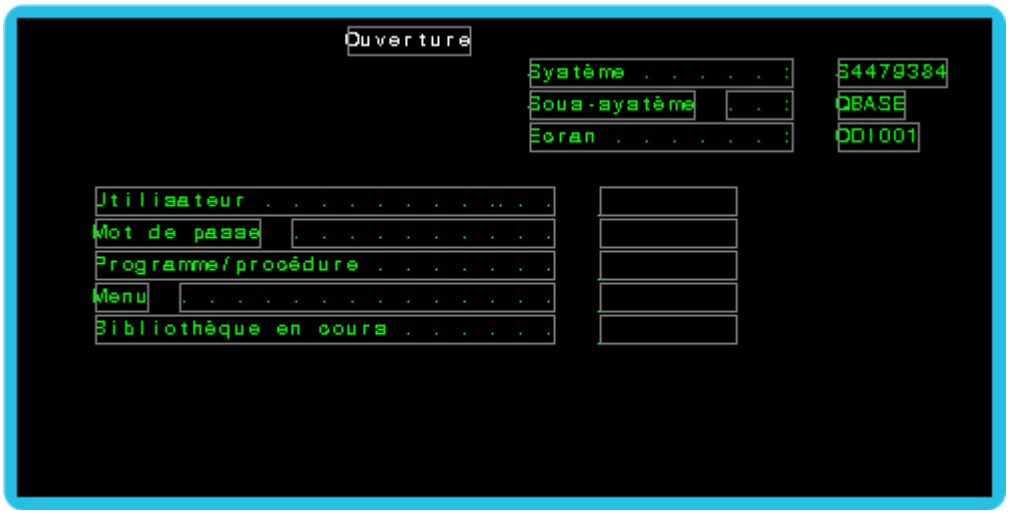


Figure 2 - 564: Attribute extraction rule - Legacy screen

Without the rule, the resulting XML is as follows:

```
<blocks page-number="0">
  <block background="black" column="34" foreground="white" line="0" name="untitled" reverse="false" type="static">Ouverture</block>
  <block background="black" column="47" foreground="green" line="1" name="untitled" reverse="false" type="static">Système . :</block>
```

Figure 2 - 565: Attribute extraction rule - Resulting XML without rule

In this example, an *Attribute* extraction rule is created with the following parameters:

```
Attribute [
   screen zone=[x=34, y=0, width=9, height=1]
   attribute name="isImportant"
   attribute value="true"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

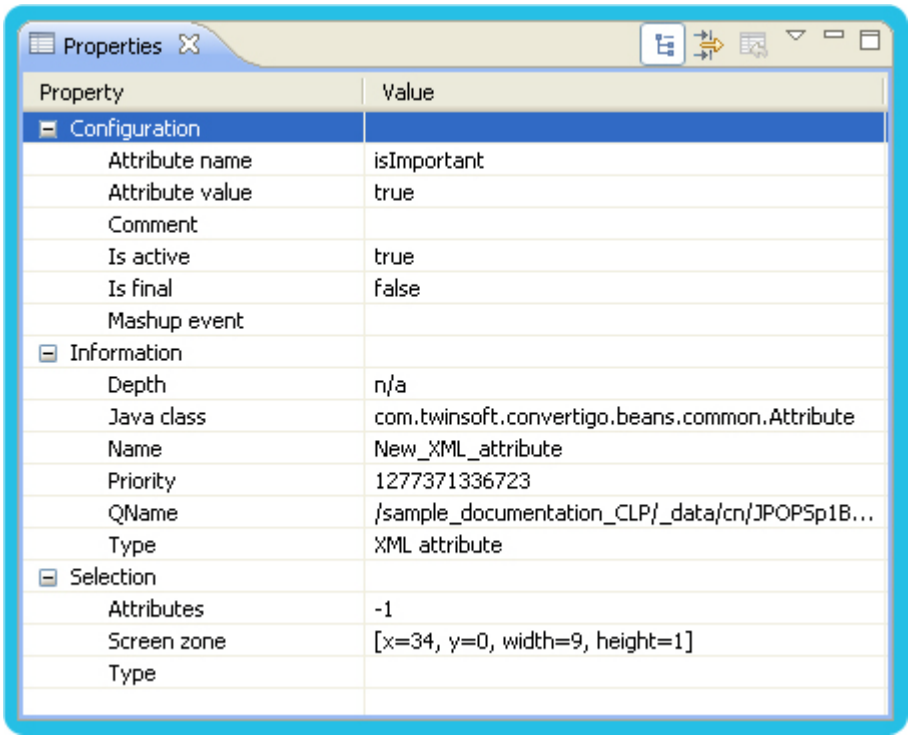


Figure 2 - 566: Attribute extraction rule - Configuration example

When the rule is executed, the resulting XML includes the defined attribute on matching blocks:

Figure 2 - 567: Attribute extraction rule - Resulting XML with rule

SPLIT STRING TO TABLE 

OBJECT DESCRIPTION

Not yet documented.

For more information, do not hesitate to contact us in the forum in our Developer Network website: http://www.convertigo.com/itcenter.html

2.10 SiteClipper

2.10.1 Main objects

SITE CLIPPER CONNECTOR 

OBJECT DESCRIPTION

Establishes connections and clips entire websites through Convertigo.

Site Clipper connector gives access to websites through Convertigo and allows dynamically transforming its pages or resources. It can access websites from several domains.

All tasks (screen classes detection, data transformation, etc.) associated with the *Site Clipper connector* are carried out as defined in the project thanks to several objects:

- Screen classes,
- Criteria,
- Rules,
- Site Clipper transactions.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Billing Java class	String	expert	Defines the Java class name executed for billing pruposes. Convertigo supports a plugin architecture offering billing functionalities. Set the name of the billing class to be called by Convertigo for billing purposes.
Carioca authentication	boolean	expert	Defines whether the connector requires a Carioca authentication. Set to `true` if you require that only Carioca-authenticated users be able to use this connector.
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Default response charset	String	standard	Defines the default charset used to decode/ encode data. Response data specify their charset in a dedicated header: `Content-Type`. This header includes a MIME type and possibly a charset. The **Default response charset** property defines a default charset to use when response data don't specify their charset. Several values are possible, for example `ISO-8859-1` or `UTF-8`.

Property	Type	Category	Description
Domains listing	XMLVector	standard	Defines a white and/or black list of domains. The *Site Clipper connector* can access websites from several domains. This property allows the Convertigo developer to define a list of domains he wants to filter. For each domain of the list, the **Domains listing** table contains two columns: • **Domain**: This property is a regular expression tested against the accessed data or resources' domain. If the domain matches the regular expression defined, the behavior depends on the second column's value, i.e. if the domain is black-listed or white-listed. • **Black listed**: This property can be `true` for black-listed domain or `false` for white-listed domain. Note: A new domain can be added to the list using the blue keyboard icon. The domains defined in the list can be ordered using the arrow up and arrow down buttons, or deleted using the red cross icon. For each resource or piece of data accessed, Convertigo tests its domain against each regular expression defined in the list, one by one. When one regular expression matches, Convertigo stops its tests and acts, depending on the **Black-listed** column value. You can use the "up" and "down" arrows to reorder domains tests priorities. Two behaviors can be defined thanks to this **Domains listing** property: • Default behavior of the *Site Clipper connector* is to white-list all unfiltered domains: every browsing in the accessed website passes through Convertigo. Resources and data matching black-listed domains will be accessed directly, not through Convertigo. • The opposite behavior can be obtained by defining a black-listing regular expression matching all domains. Then, resources and data matching white-listed domains will be the only ones to be accessed through Convertigo. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
End transaction	String	expert	Defines the transaction to execute before removing the context. When a Convertigo context is removed, the specified "End transaction" is executed. Place in this transaction any clean up code, for example a Logout transaction.

Property	Type	Category	Description
Trust all certificates	boolean	standard	Defines whether trusted certificates must be checked. In SSL mode, the server sends existing certificates to Convertigo. In most cases, set this setting to `true` to automatically trust all server certificates. If set to `false` , target server certificates must be installed in Convertigo.

EXAMPLES

Example 1

The following is an example of *Site Clipper connector* set for connecting to every website without exception:

```
Site Clipper connector [
  default charset encoding=UTF-8
]
```

In the context of the Convertigo administration website clipping, this connector is used to implement all mandatory objects: screen classes, criteria, rules, transactions.

These parameters are edited in the **Properties** view of the Convertigo Studio:

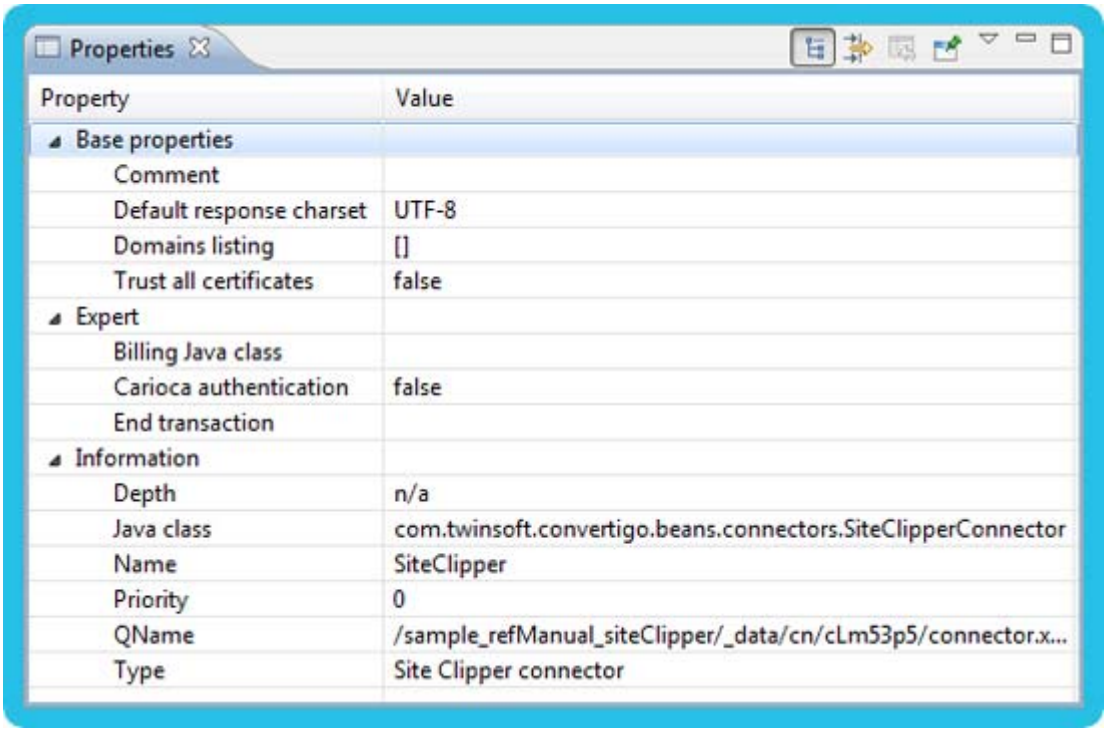


Figure 2 - 568: Site Clipper connector - Configuration example

In the Convertigo Studio, the **Site Clipper connector** editor displaying the generated XML or the accessed website and the HTTP data and resources accessed through Convertigo appears as follows:

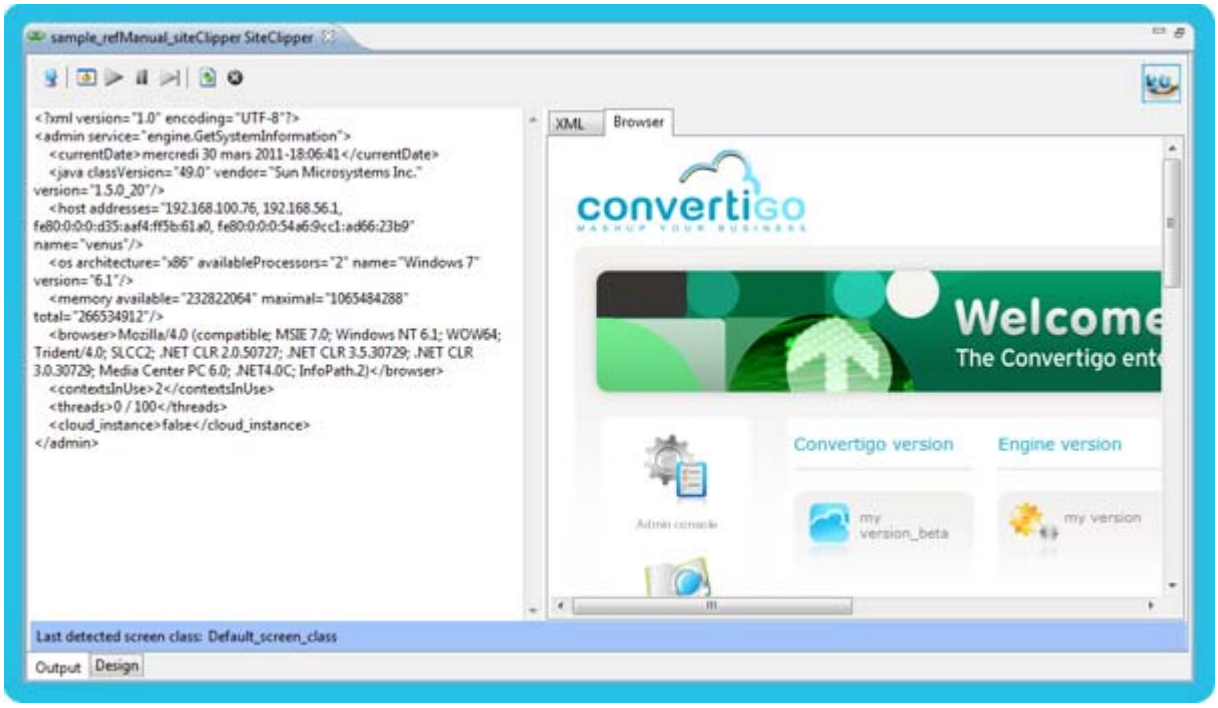


Figure 2 - 569: Site Clipper connector - Connector editor in Studio

Example 2

The following is an example of *Site Clipper connector* set with black listed domains.

Let's consider the french version of the Google website.

Figure 2 - 570: Site Clipper connector - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site, dynamically access all resources and pages from Google website through Convertigo, except the Maps.

To do so, the *Site Clipper connector*, named `Google_and_blacklist`, is configured with the following parameters:

```
Site Clipper connector [
   default charset encoding=UTF-8
   domains listing: [
      [maps\.google\., true]
   ]
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

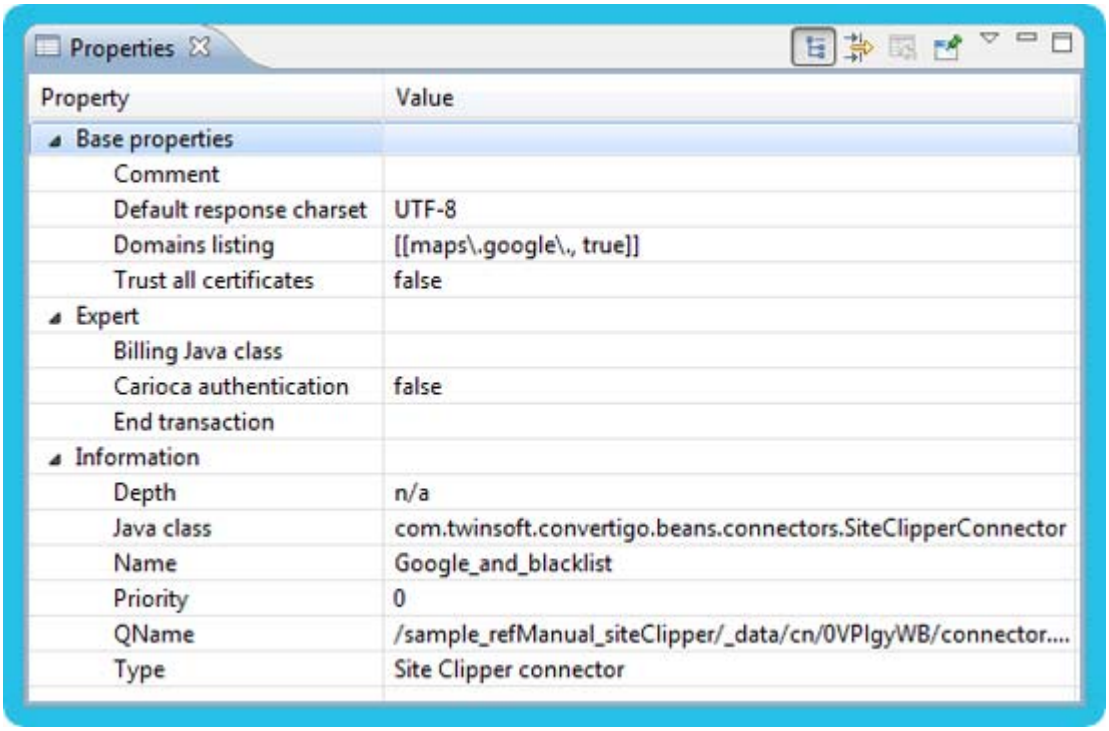


Figure 2 - 571: Site Clipper connector - Configuration example

The **Domains listing** property contains one entry, in order to blacklist all links and resources containing `maps.google.` in their URL.

A screen classes hierarchy is created in this *Site Clipper connector* with a root screen class, named `Google`, defined thanks to a request *URL* criterion to handle Google ressources only. On this screen class, two default extraction rules are created in order to handle target URLs of redirections (*Rewrite location header* extraction rule) and target URLs of links and resources (*Rewrite absolute URL* extraction rule). For more information about this criterion and these extraction rules, see their specific documentation.

A transaction, named `Google_fr_transaction`, is created in the *Site Clipper connector*. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

Switch to a browser displaying the test platform of this project. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google.fr page through Convertigo.

Rolling the mouse over the top left links (Images, Vidéos, etc.) shows links accessing Google resources through Convertigo (URL starting with `http://localhost:18080/convertigo/...`):

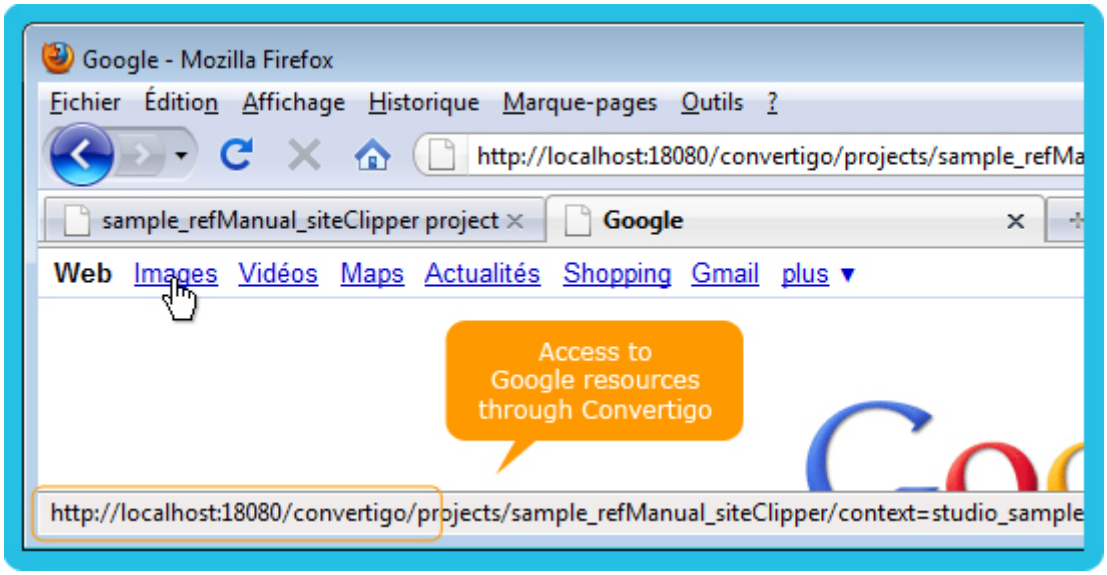


Figure 2 - 572: Site Clipper connector - Accessing Google resources through Convertigo

The only link that is accessed directly (not through Convertigo) is **Maps**. The link URL matches the blacklisted domain: it has not been rewritten by the extraction rule.

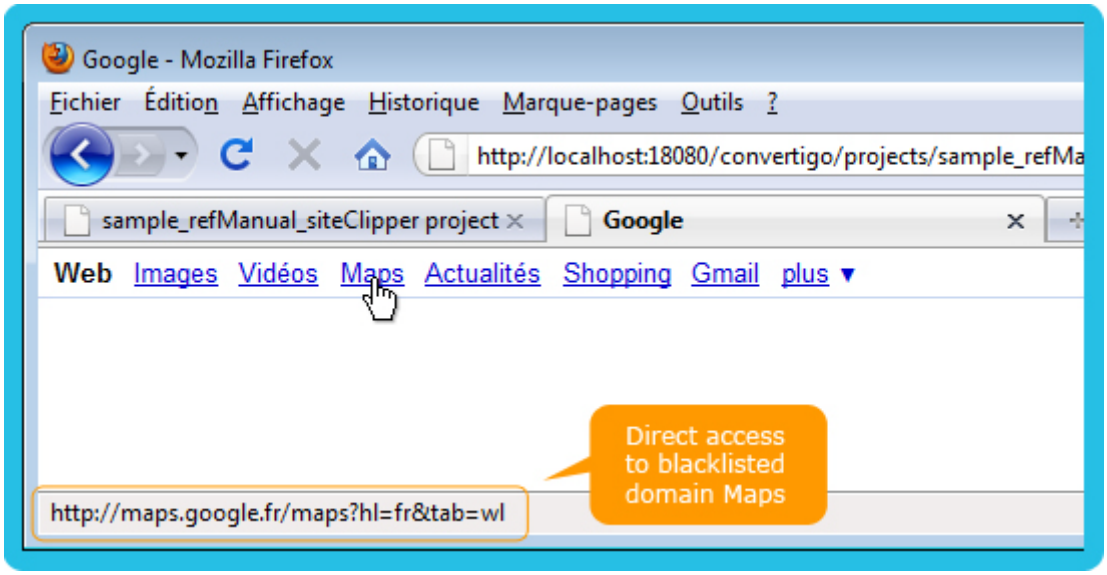


Figure 2 - 573: Site Clipper connector - Maps domain blacklisted

SITE CLIPPER TRANSACTION 

OBJECT DESCRIPTION

Defines a transaction for a *Site Clipper connector*.

A *Site Clipper transaction* allows Convertigo to connect to a remote web server hosted at the URL defined in **Target URL** property.

Its execution does not return data from the target server, but specifies a redirection URL to its parent connector. This rewritten URL is an absolute URL pointing to the current Convertigo project, with a particular syntax:

- it starts with the usual project's path,
- it then specifies the Convertigo context and the *Site Clipper connector* to use,
- it ends with the `.siteclipper` extension,
- after the extension, the target resource URL is concatenated, replacing the '`://`' symbols after the target resource protocol, `http://` for example, by a '`/`' character.

This gives the following URL form:

```
http://<convertigo_server_host>:<convertigo_server_port>/convertigo/
projects/<project_name>/
context=<context_name>,connector=<connector_name>.siteclipper/
<target_resource_protocol>/<target_resource_host>/
<target_resource_URI>.
```

The *Site Clipper connector* accessed thanks to this URL then relays all HTTP messages between the client and the target server.

To sum up, the *Site Clipper transaction* is used to initiate a site clipping process on a website, including the initialization of a context in the Convertigo server.

Note: See *Site Clipper connector* and all related objects (*Screen class*, criteria, extraction rules,etc.) documentation for more information on how to manipulate the relayed HTTP data for site clipping purpose.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Accessibility	Accessibility	standard	Defines the transaction/sequence accessibility. This property can take the following values: • `Public`: The transaction/sequence is runnable from everyone and everywhere, visible in the Test Platform and is also exposed in the SOAP WSDL as a web service method. • `Hidden`: The transaction/sequence is runnable but only from people who know the execution URL, not visible in the Test Platform nor exposed in the SOAP WSDL. • `Private`: The transaction/sequence is only runnable from within the Convertigo engine (*Call Transaction*/(*Call Sequence* steps), is not visible in the Test Platform and cannot be requested as SOAP web service method. This value is used for tests, unfinished transactions/sequences or functionalities not to be exposed. Private transactions/ sequences remain runnable in the Studio, for the developer to be able to test its developments. Note: In the Test Platform: • The administrator user (authenticated in Administration Console or Test Platform) can see and run all transactions / sequences, no matter what their **accessibility** is. • The test user (authenticated in the Test Platform or in case of anonymous access) can see and run public transactions/ sequences and run hidden ones if he knows their execution URL.
Add statistics to response	boolean	expert	Defines whether some statistics of execution of the transaction/sequence should be added as data in the transaction/sequence's response. If this property is set to `true`, the transaction/ sequence response will be enhanced with the statistics data of its execution (total time for the request, time spent waiting for the mainframe, etc.). Note: This property has nothing to do with the general property of the Convertigo engine **Insert statistics in the generated document** that can be edited in the *Configuration* page of the *Administration Console*.

Property	Type	Category	Description
Authenticated context required	boolean	expert	Defines whether an authenticated context is required to execute the transaction/sequence. If this property is set to `true`, the context of execution of the transaction/sequence must have been authenticated. Otherwise, the transaction/sequence is not executed. Default value is `false` for a standard access to transactions/sequences. Notes: • When a context is authenticated, all the contexts in the same HTTP session are also authenticated. For more information about context and HTTP session, see *Context general presentation* paragraph in *JavaScript Objects APIs* chapter. • When executing a transaction/sequence from stub (`__stub` variable passed to `true` in entry), this property is ignored. Indeed, executing from stub is for testing purposes and should not require any authentication: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub.
Authenticated user as cache key	boolean	expert	Defines whether the authenticated user should be used as cache key. When the cache is enabled (**Response lifetime** setting filled with a time-to-live), the **Authenticated user as cache key** property allows to specify to use the authenticated user ID from context/session as an additional key to the cache. It would have as effect that two different identified users cannot retrieve the cached response of the other for the same request. Default value is `false`: the authenticated user is not used as cache key.
Call the biller	boolean	expert	Defines whether the billing management module should be called for each generated XML document. If this property is set to `true`, the applicable billing management module, defined thanks to the connector's billing class name property, is invoqued. **This parameter should never be changed** (Convertigo private use only).
Character set	String	expert	Defines the character set used for operations on the generated XML document (default: `UTF-8`).
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Include certificate group	boolean	expert	Includes the certificate group into the cache key. If set to `true`, the certificate group is added to the cache key which is used to determine whether the transaction's response should be pulled from the cache or not. A transaction's cached response is pulled from the cache when all cache key values are corresponding to a stored cache entry.

Property	Type	Category	Description
Response timeout	long	standard	Defines the response maximum waiting time (in seconds). Maximum time (in seconds) for a transaction/ sequence to run. When specified time is reached, the transaction/sequence ends and returns a timeout error. If requested through the SOAP interface, the error is returned as a SOAP exception.
Secure connection required	boolean	expert	Defines whether the transaction/sequence should be called through a secured connection (e.g. HTTPS). Depending on the requester, if this property is set to `true`, the transaction/sequence must be accessed through a secure connection (e.g. HTTPS in case of HTTP access). Default value is `false` for a standard access to transactions/ sequences.
Target URL	String	standard	Defines the URL of a remote website to be clipped by Convertigo. This property defines the URL to which connect when starting the site clipping process thanks to this transaction.

EXAMPLES

Let's consider the Convertigo test platform Home page, listing available projects and giving the ability to access several applications such as Convertigo Administration website.

Figure 2 - 574: Response header criterion - Convertigo test platformHome page

In the context of a *Site Clipper connector*, we would like to clip the entire site, dynamically access all resources and pages from Convertigo test platform website through Convertigo. To

do so, a screen classes hierarchy is defined thanks to several criteria to identify accesses data and resources, which are possibly modified by extraction rules defined on these screen classes.

A transaction, named `connectionLocalIP`, is defined to connect to this page in the *Site Clipper connector* named `ConvertigoAdminConnector`. It is created with the following parameters:

```
Site Clipper transaction [
  target URL=http://127.0.0.1:18080/convertigo/index.html
  response timeout=60
  character set=ISO-8859-1
]
```

It appears as follows in the **Properties** view of the Convertigo Studio:

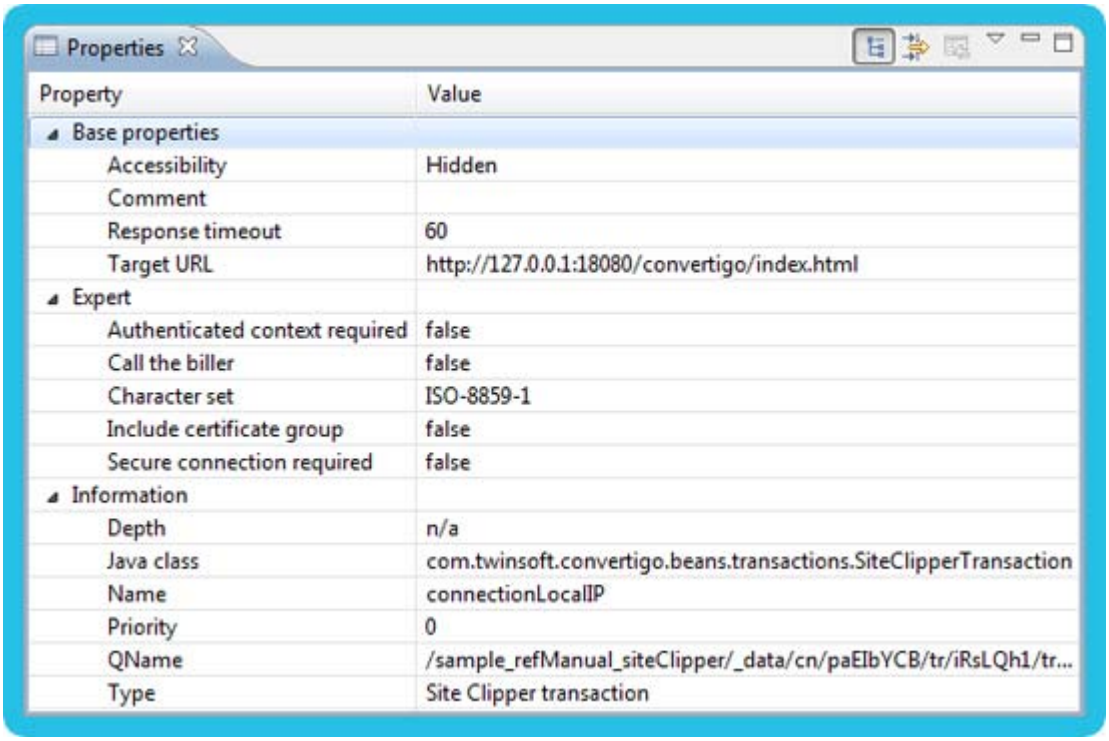


Figure 2 - 575: Site Clipper transaction - Configuration example

The **Target URL** property is set to the local IP address for localhost `127.0.0.1` to match certain screen classes defined in the connector.

The transaction appears as follows in the **Projects** view:

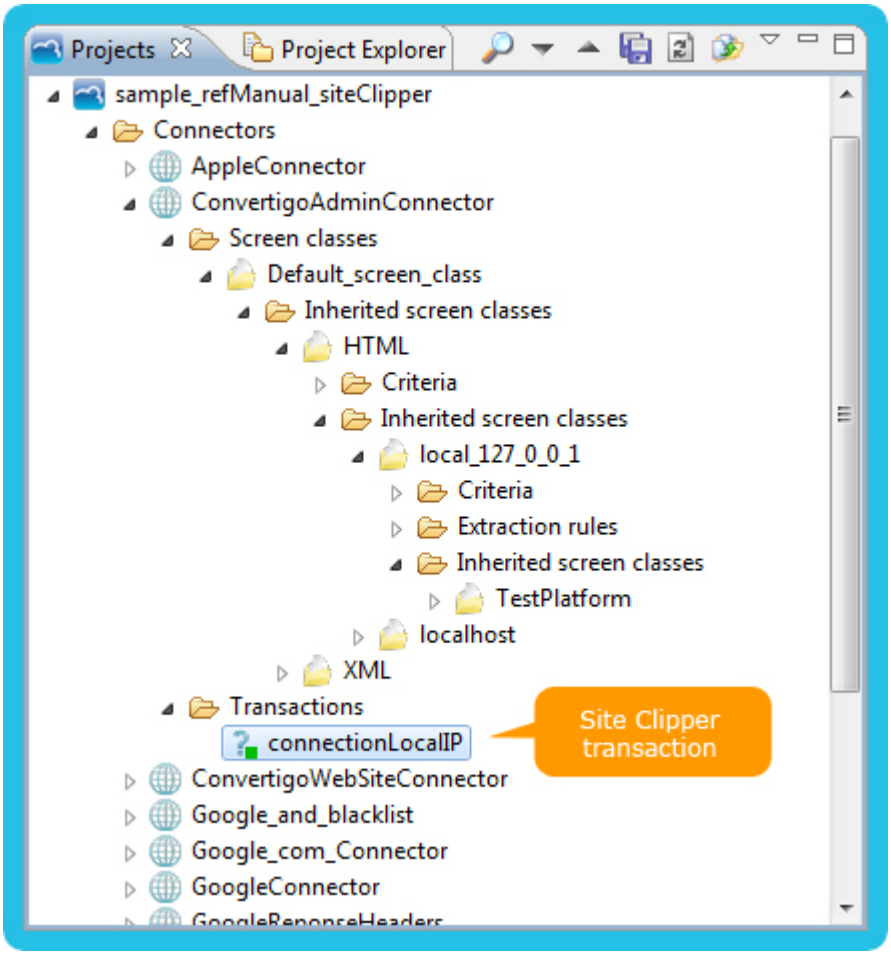


Figure 2 - 576: Site Clipper transaction - Object in Projects view

Switch to a browser displaying the test platform of this project. Executing the `connectionLocalIP` transaction (in a new tab thanks to **Execute full screen** button) reaches the Convertigo test platform Home page through Convertigo:

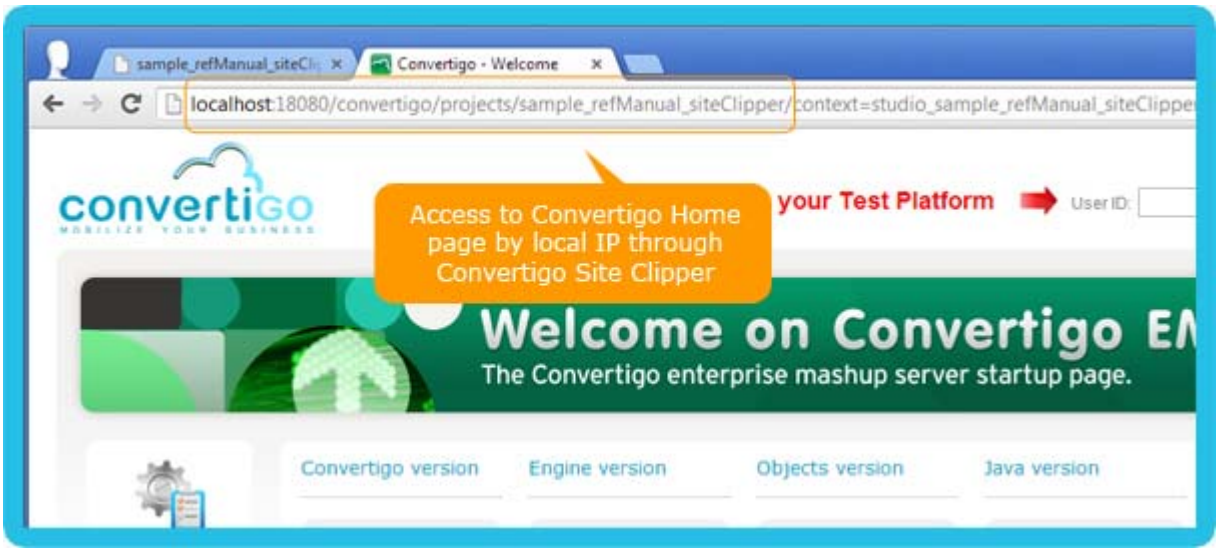


Figure 2 - 577: Site Clipper transaction - Convertigo test platform Home page accessed through Convertigo

Switching back to the Convertigo Studio, the **XML** tab of the connector editor shows the returned XML document which contains the redirection URL to initiate the site clipping:

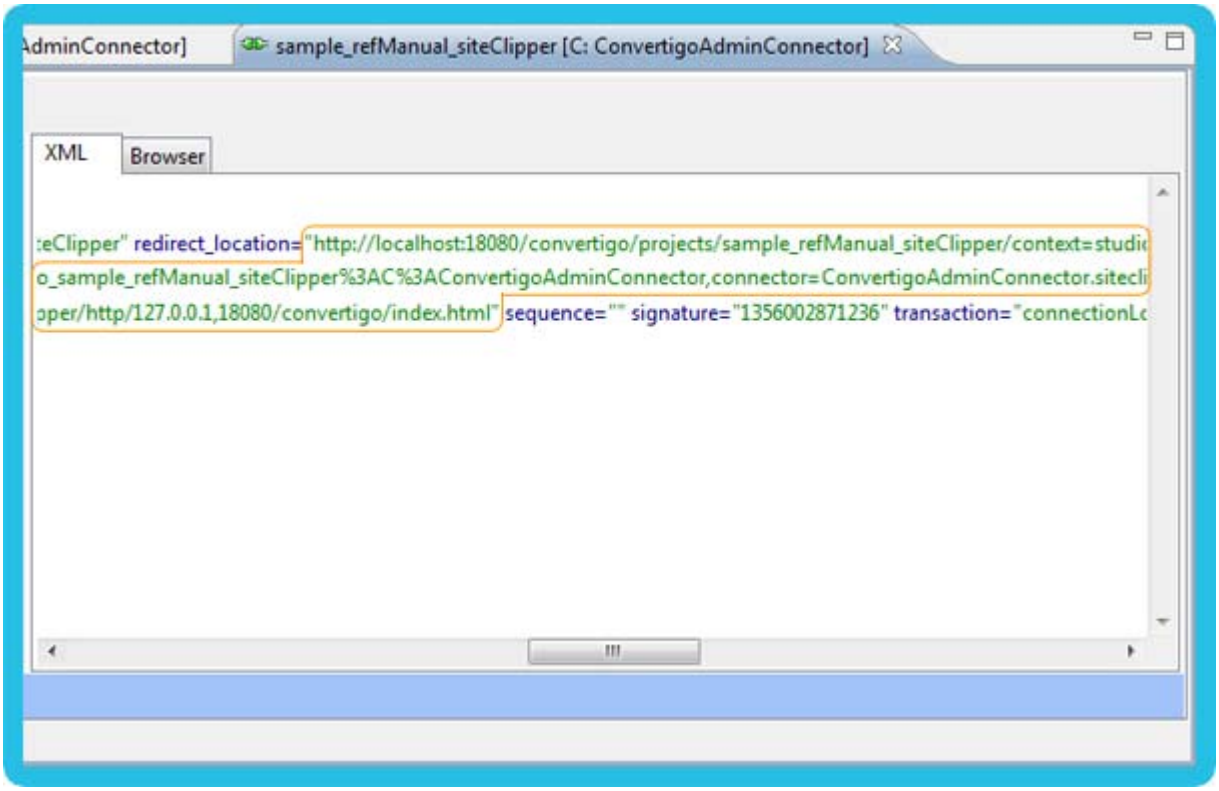


Figure 2 - 578: Site Clipper transaction - Redirection URL in XML

Site Clipper screen class 

Object Description

Defines a group of screens with common features in a Site Clipper connector.

By the term "screen" is meant a set of identifiable data which may be rendered to the user or not. It is generally used regardless of the resource accessed by Convertigo (web page, Legacy screen, HTTP stream, etc.).

Thus, in the case of *Site Clipper connector* projects, a screen may be defined by the data contained in an HTTP message, for a resource request.

A *Site Clipper screen class* is identified by a set of criteria which are dedicated to screen's data detection. When accessing a screen (i.e. a web resource), Convertigo looks for detection criteria defined for screen classes.

Convertigo considers that the accessed screen belongs to the *Site Clipper screen class* which all criteria match and which have the greatest number of criteria matching. For screen classes that would have the same number of matching criteria, Convertigo considers that the screen belongs to the screen class that has the greatest depth. And if screen classes also have the same depth, Convertigo considers that the screen belongs to the first screen class in alphabetical order.

For Site Clipper projects (web applications and HTTP streams in *Site Clipper connector*), detection criteria are *MIME type*, *Regular expression*, *Request header*, *Response header* and *URL*.

A *Site Clipper screen class* can also be associated with extraction rules executed on its detection by Convertigo. Extraction rules define which data are to be modified from a screen and turned into an HTTP request or response.

Site Clipper screen classes are pivotal in the execution of transactions, since their detection triggers the execution of screen class handlers (including actions to be performed on detected screens) and extraction rules (modifying HTTP data).

Note: A *Site Clipper screen class* do not define one screen only, but all screens matching the specified criteria. It is up to the Convertigo programmer to set detection criteria.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.

Examples

HTTP messages consist of requests from client to server and responses from server to client.

The Site Clipper works on this client - server conversation and handles each message exchanged as a "screen" of identifiable data.

HTTP messages are thus of two types (request or response), they declare headers and may have a body content. Site Clipper detection criteria are based on those information and their validation identifies a Site Clipper *Screen class*.

Example 1

Let's consider the Convertigo test platform Home page, listing available projects and giving the ability to access several applications such as Convertigo Administration website, Developer network website, etc.

Figure 2 - 579: Site Clipper Screen class - Convertigo test platformHome page

This page is a dynamical page, listing projects currently deployed on the server and displaying information about Convertigo platform on the top of the page (Convertigo version, Engine version, etc.). These last data are XML resources getted from a web service exposed by the Convertigo server.

In this example, we want to define a *Screen class* matching on this XML resources. A *Screen class* object has no properties to configure, it is defined by its criteria:

```
Screen class [
]
```

The *Screen class* object is created in the **Screen classes** folder of the connector, inherited from the *Default_screen_class* screen class. It is created together with its first criterion and appears as follows in the **Projects** view:

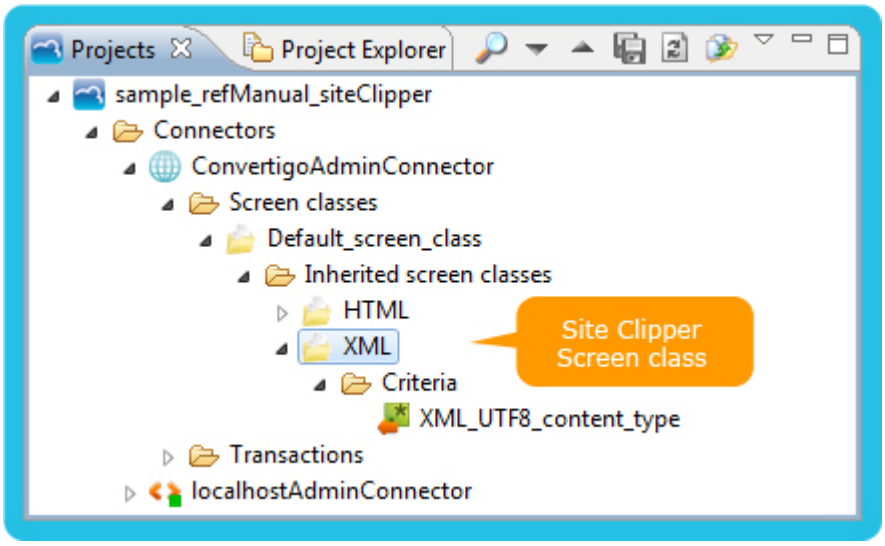


Figure 2 - 580: Site Clipper Screen class - Screen class and first criterion in Projects view

Thanks to its criterion defining the remarquable characteristics of the "screen", this *Screen class* matches the previously described web service response data. For more information about this criterion, see "*Response header"* criterion documentation and examples.

Example 2

When you create a new Site Clipper project in Convertigo studio, its default connector is created with:

- a default transaction, named `Default_transaction`, for which you may modify the **Target URL** property value in the creation wizard,
- a default root screen class, named `Default_screen_class`, including predefined inherited screen classes to start using and developing a Site Clipper project.

This architecture should be suitable in most cases.

The following table summarizes all the implemented screen classes with their criteria:

Table 2 - 6: `newSiteClipperProject` project screen classes

Name	Description	Detection criterion
`CSS`	This screen class will detect all HTTP responses of CSS type returned by the server.	A *MIME type* criterion to identify stylesheet resources only.
`HTML_pages`	This screen class will detect all HTTP responses of HTML type returned by the server.	A *MIME type* criterion to identify HTML pages only.
`Javascript`	This screen class will detect all HTTP responses of script type returned by the server.	A *MIME type* criterion to identify JavaScript resources only.
`googleResultPageCurrent`	Detected when accessing a current Google result page (every page but the last).	Root screen class criterion + Final Result page criterion + "Next" link.

The root screen class defines a *Rewrite location header* rule to process any request of redirection and an additional *Rewrite absolute url* rule to rewrite URLs found in the HTTP response returned by the server. For more information, see the "*Rewrite location header*" and

"*Rewrite absolute URL*" documentations and examples.

Site Clipper screen classes appear together with their detection criteria and rules in the **Projects** view of the Convertigo Studio:

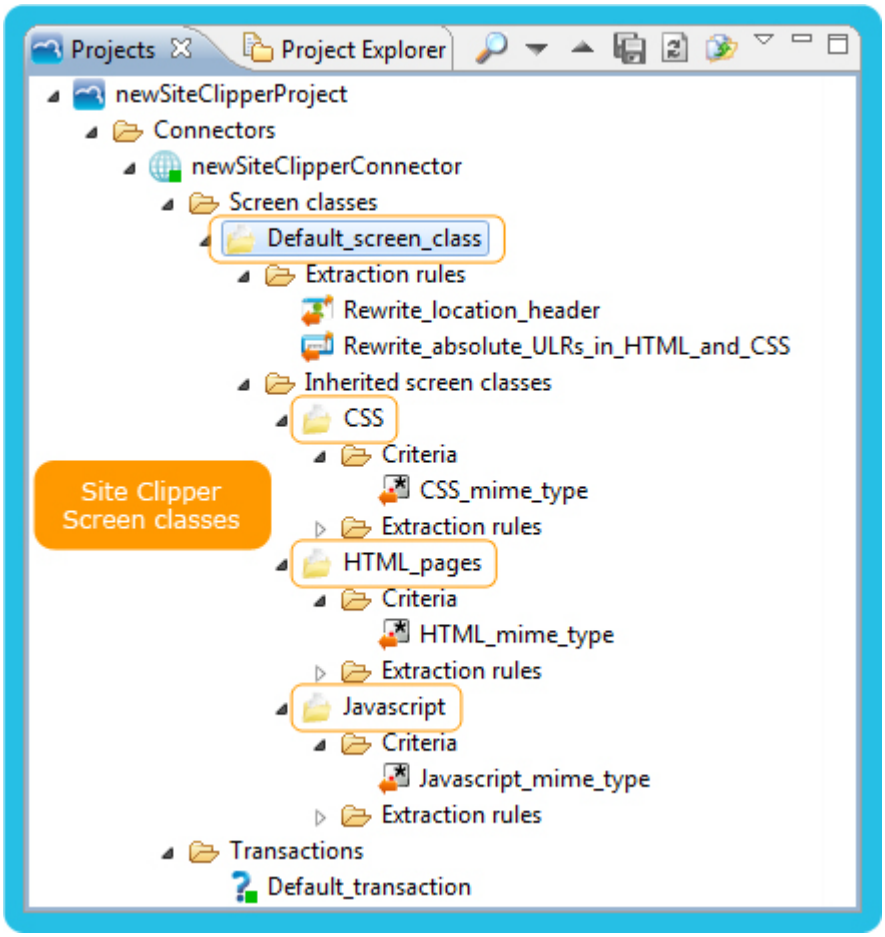


Figure 2 - 581: Site Clipper Screen class - Project's screen classes, respective criteria and rules

Screen class appear as follows in the **Properties** view (here, the `HTML_pages` screen class):

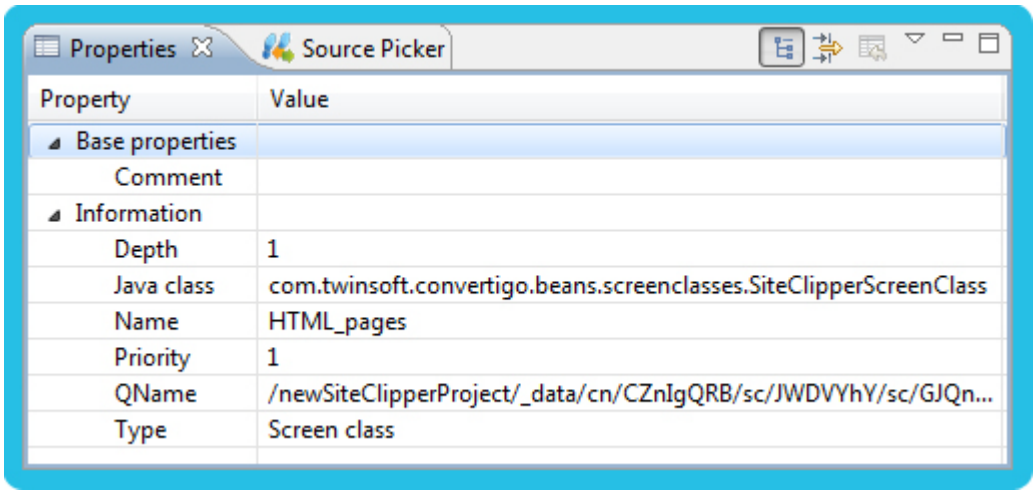


Figure 2 - 582: Site Clipper Screen class - HTML_pages screen class properties

For more information about the different criteria defined to detect these screen classes, see "*MIME type*" criterion documentation and examples.

2.10.2 Criteria

REQUEST CRITERIA

URL (SiteClipper) 

Object Description

Defines a request criterion based on URL for Site Clipper screen classes.

The *URL* criterion allows defining a regular expression that is applied on request URL.

By default, the regular expression is tested on base URL. The query string can be added to the tested URL thanks to the **Include query string** property.

Matching condition: The *URL* criterion matches when the regular expression defined in **Regular expression** property matches request URL, i.e. if the string pattern described by the regular expression is found in the request URL.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Include query string	boolean	standard	Defines whether the query string should be concatenated to the base URL for detection. If this property is set to `true`, the query string, if existing, is concatenated to the base URL before the **Regular expression** is applied.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

Examples

Let's consider the Convertigo administration website. In the context of this website clipping, the

login phase is automated thanks to an *HTML transaction*, named `LoginAdmin`, and then the user gets the browsing control back on the Configuration page of Convertigo administration website.

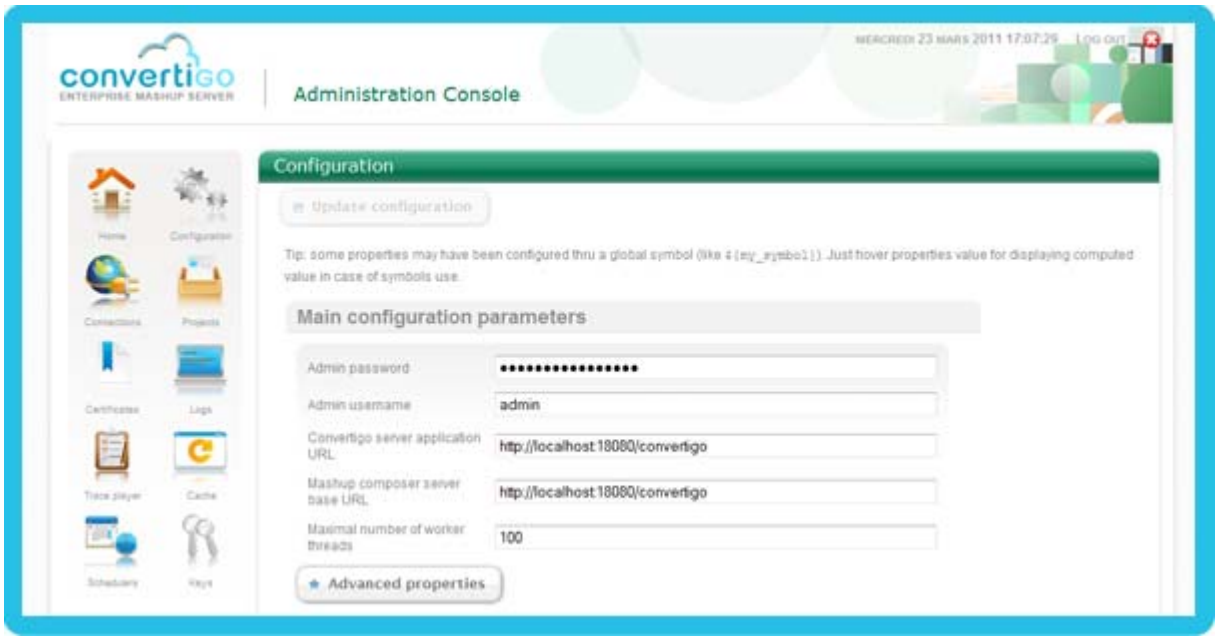


Figure 2 - 583: URL criterion - Convertigo Administration Configuration page

A *Site Clipper connector* is defined in the same project for the administration website to be accessed through Convertigo. It contains a screen classes hierarchy, defined thanks to several criteria, that identifies accessed data and resources.

A first *URL* criterion is created to define a root screen class with the following parameters:

```
URL [
   regular expression=\.html$
   include query string=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

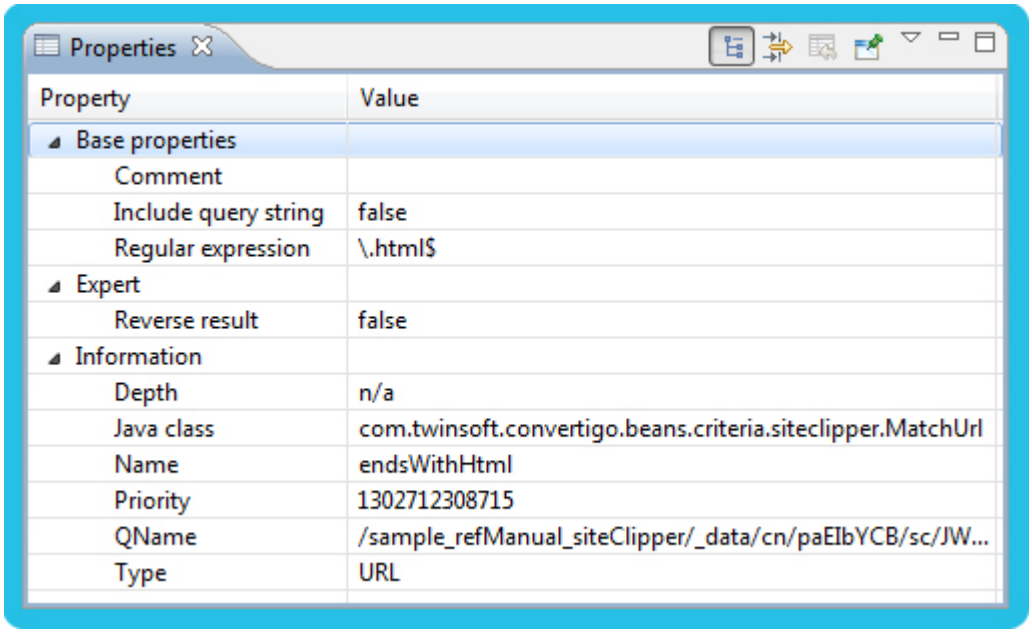


Figure 2 - 584: URL criterion - Configuration example

The **Regular expression** property is defined to match every requested URL ending by "`.html`". The **Include query string** property is thus set to `false` in order to test the base URL only.

The screen class defined by this criterion is matching every HTML page accessed through the *Site Clipper connector*.

Two other *URL* criterion are created to define two sister screen classes inherited from previous one. Criteria are set with the following parameters:

```
URL [
   regular expression=localhost
   include query string=false
]
URL [
   regular expression=127\.0\.0\.1
   include query string=false
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

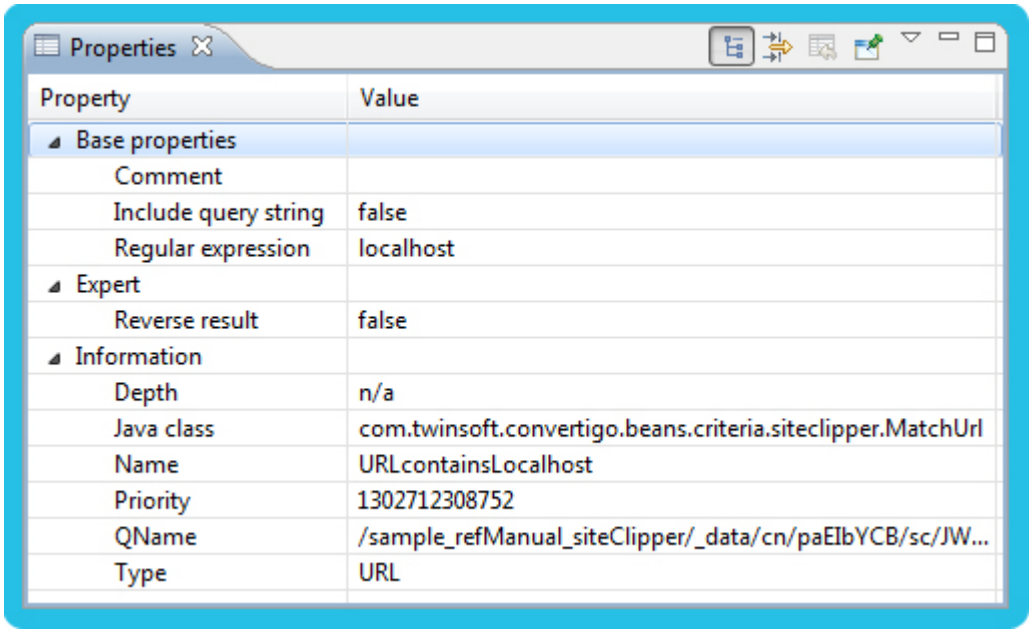


Figure 2 - 585: URL criterion - Configuration example

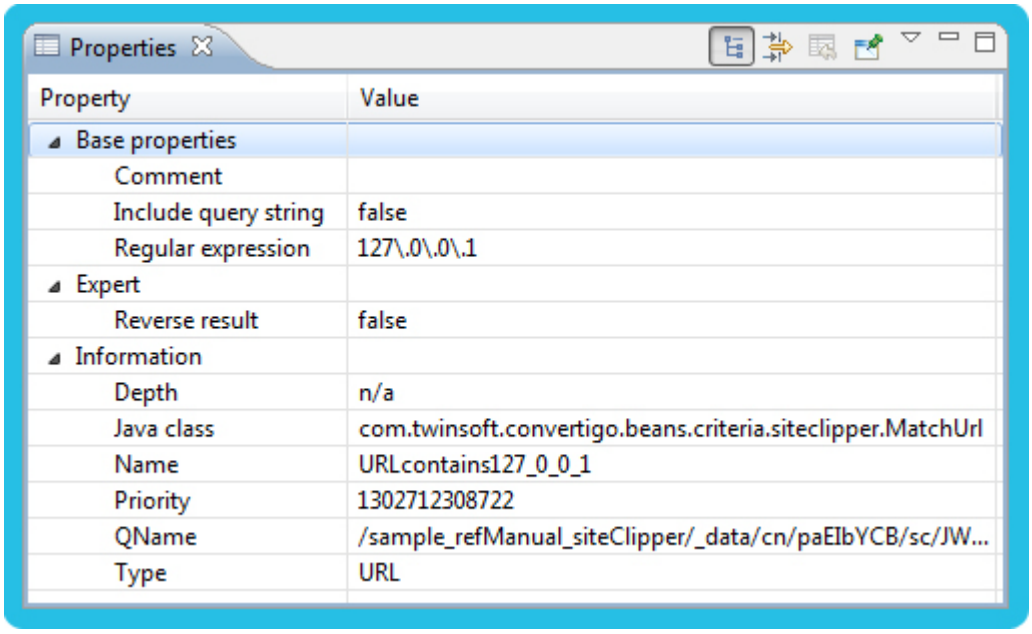


Figure 2 - 586: URL criterion - Configuration example

The **Regular expression** property is defined for both criteria to match every requested URL containing "`localhost`" or "`127.0.0.1`" (which is the IP address for localhost).

The screen classes defined by these criteria are matching every HTML web page (criterion inherited from parent screen class) from local domain accessed through the *Site Clipper connector*, differentiating them by the way they are called in the URL: name or IP address.

By definition, these screen classes are matching Convertigo administration website pages accessed through the *Site Clipper connector*, possibly connected by the `LoginAdmin` HTML transaction.

Both criteria with screen classes they define appear as follows in the **Projects** view:

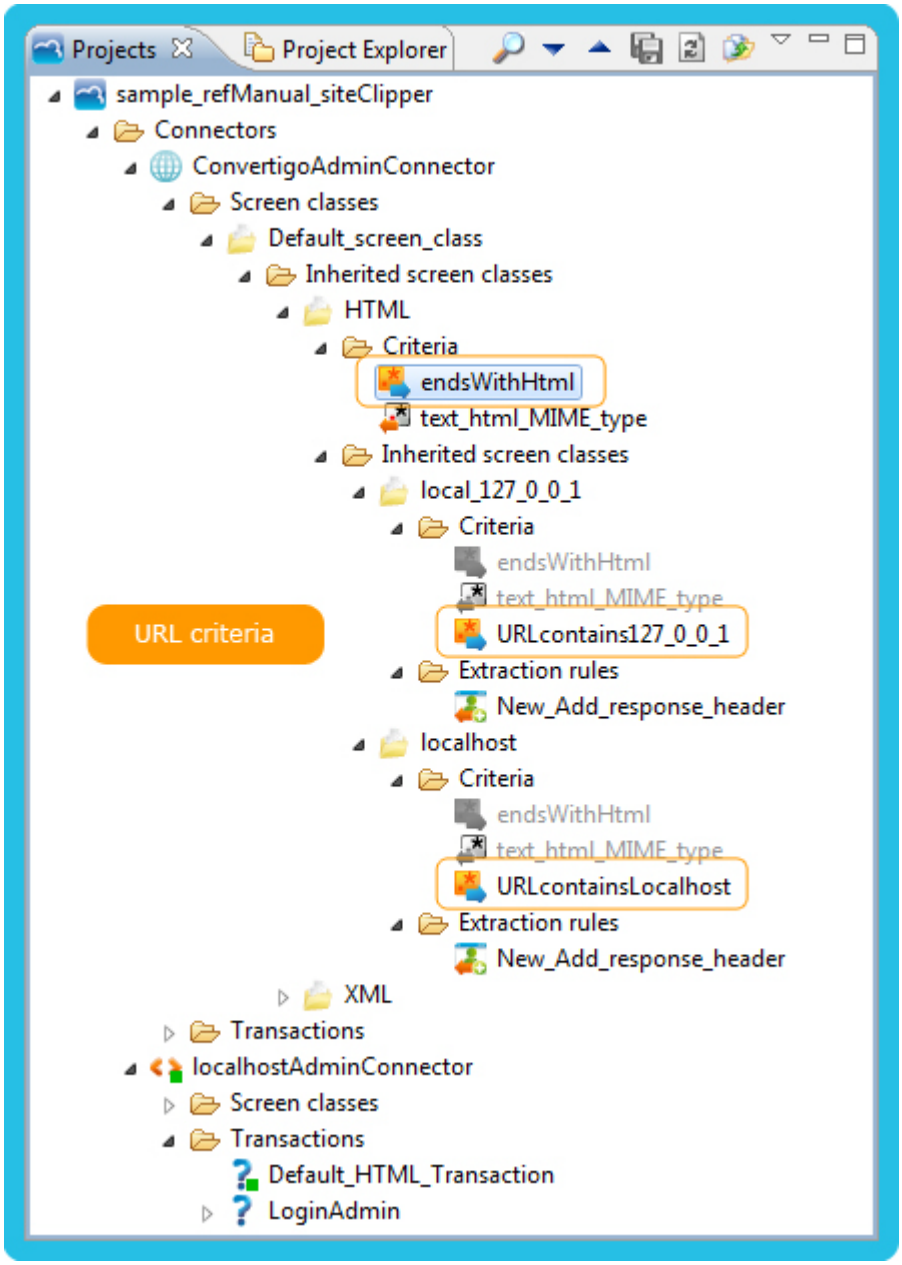


Figure 2 - 587: URL criterion - Objects in Projects view

Swith to the test platform of this project in a Firefox web browser, activate Firebug extension. Executing the `LoginAdmin` *HTML transaction* (in a new tab thanks to **Execute full screen** button), the user gets the browsing control back on the Configuration page of Convertigo administration website.

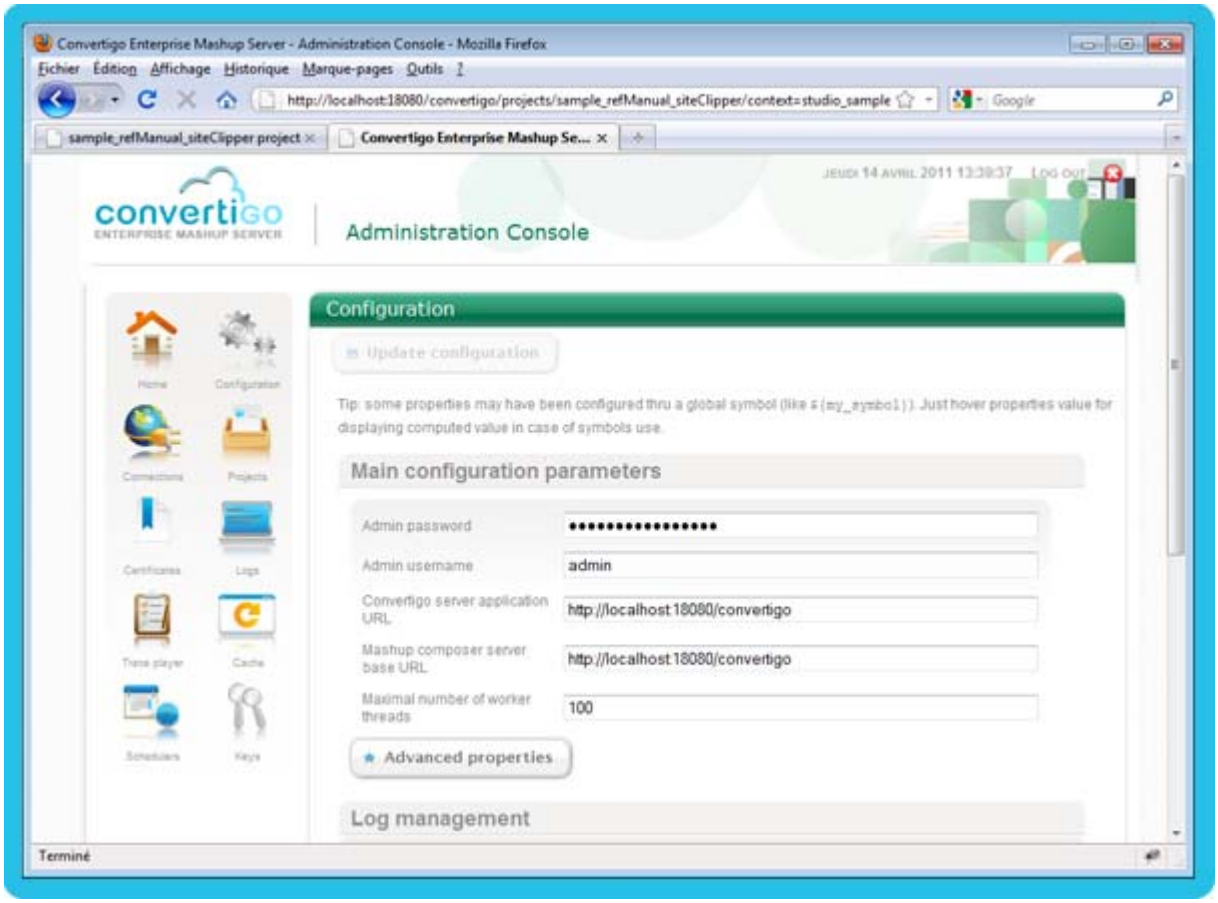


Figure 2 - 588: URL criterion - Browser after the execution of the `LoginAdmin` transaction

In Firebug, we can see that a response HTTP header, named `myConvertigoHeader`, was added by Convertigo to the HTML page resource. This HTTP header is added thanks to an *Add response header* extraction rule, one is positionned on each screen class. The header values are parametered with the screen classes names, which differ between both screen classes. Thus, this header value reflects the detected screen class.

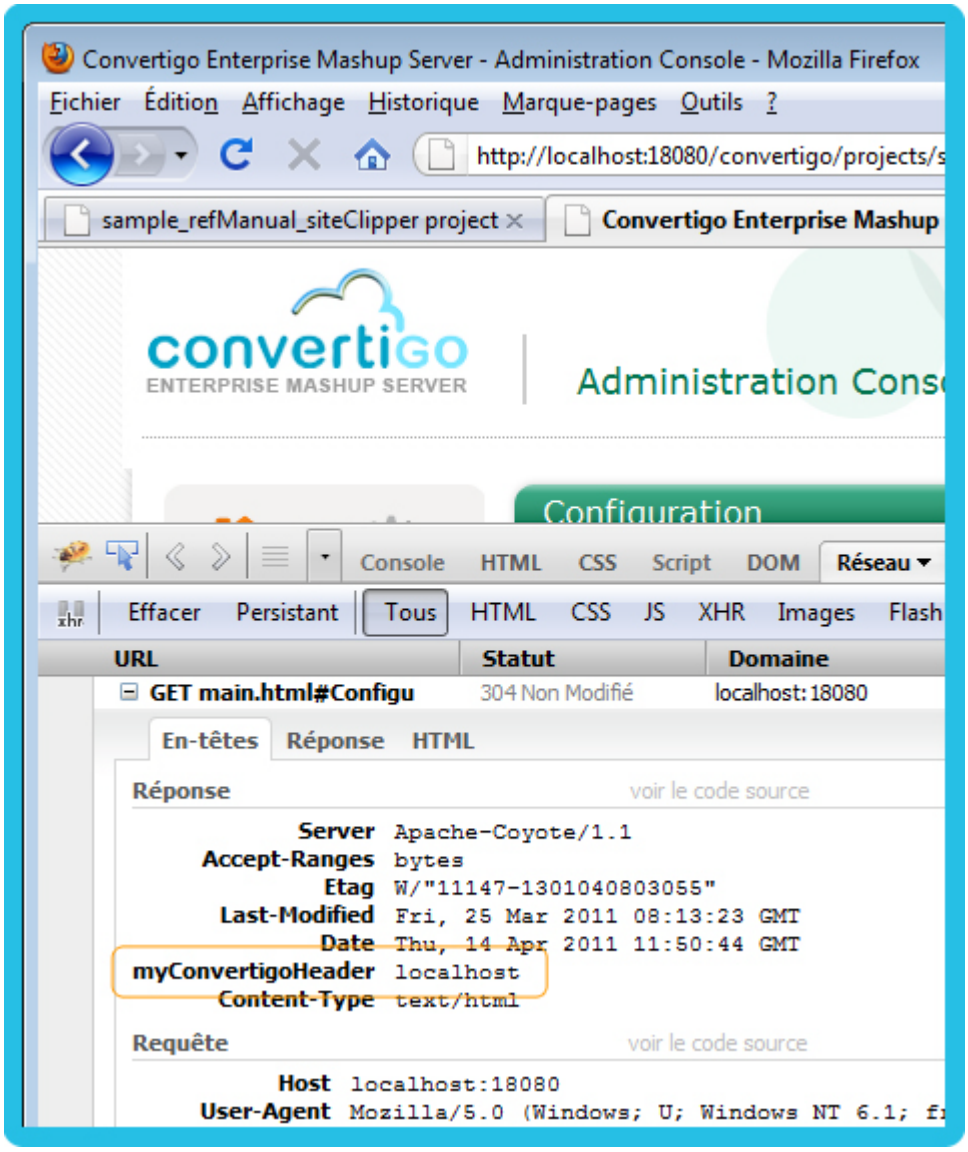


Figure 2 - 589: URL criterion - Added HTTP header containing detected screen class name

In the test browser, the header value is "`localhost`" meaning that the screen class defined by the *URL* criterion for localhost has matched.

For more information about *Add response header* extraction rule, see "*Add response header*" extraction rule documentation and examples.

REQUEST HEADER 

OBJECT DESCRIPTION

Defines a request criterion based on HTTP headers for Site Clipper screen classes.

The *Request header* criterion allows defining a regular expression that is applied on a request header. Request header to test is defined in **Header name** property.

Matching condition: The *Request header* criterion matches when the regular expression defined in **Regular expression** property matches defined request header, i.e. if the string pattern described by the regular expression is found in the request header value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	standard	Defines the name of the header which content is tested against the regular expression.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EXAMPLES

Let's consider the Convertigo administration website. In the context of this website clipping, the login phase is automated thanks to an HTML transaction, named `LoginAdmin`, and then the user gets the browsing control back on the Configuration page of Convertigo administration website.

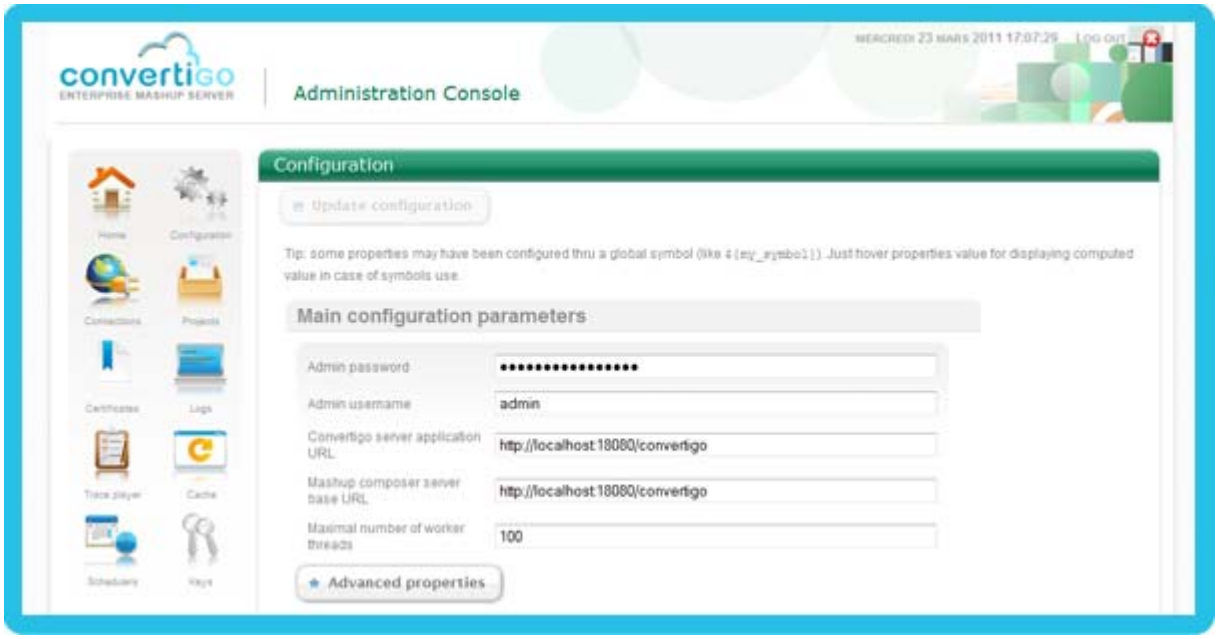


Figure 2 - 590: Match request header criterion - Convertigo Administration Configuration page

A *Site Clipper connector* is defined in the same project for the administration website to be accessed through Convertigo. It contains a screen classes hierarchy, defined thanks to several criteria.

A root screen class is defined to match HTML pages that are accessed through Convertigo. Inherited screen classes are defined to differentiate localhost named server from local IP calls in the URL. In "localhost" branch, a child screen class is defined to match every page from Convertigo Administration website accessed through the *Site Clipper connector*.

Then, two *Request header* criteria are created to define two sister screen classes inherited from previous one. Criteria are set with the following parameters:

```
Request header [
   header name=User-Agent
   regular expression=Chrom
]
Request header [
   header name=User-Agent
   regular expression=Firefox
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

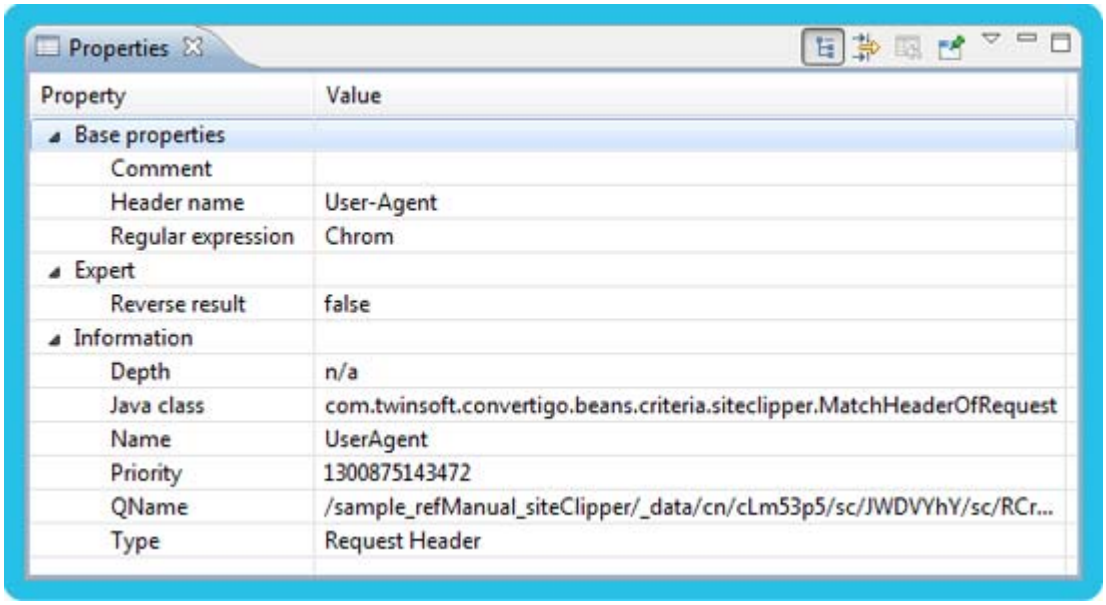


Figure 2 - 591: Request header criterion - Configuration example

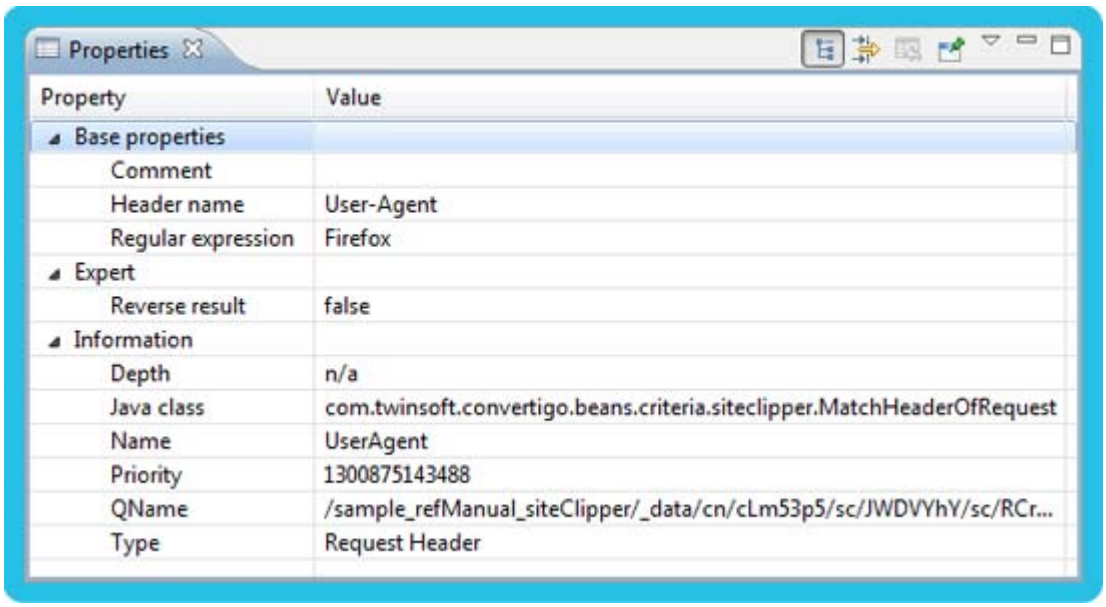


Figure 2 - 592: Request header criterion - Configuration example

For both criteria, the **Header name** property is defined to analyse content of `User-Agent` request header. The **Regular expression** property is differentiating contents to find.

The screen classes defined by these criteria are matching every Convertigo Administration (parent screen class criterion) HTML web page (criterion inherited from root screen class) from localhost domain (criterion inherited from parent's parent screen class) accessed through the *Site Clipper connector*, differentiating them by the client requester, i.e. requests coming from a Chrome browser and requests coming from a Firefox browser.

Both criteria with screen classes they define appear as follows in the **Projects** view:

Figure 2 - 593: Request header criterion - Objects in Projects view

Swithch to the test platform of this project in a Firefox web browser. Executing the `LoginAdmin` HTML transaction, the user gets the browsing control back on the Configuration page of Convertigo administration website.

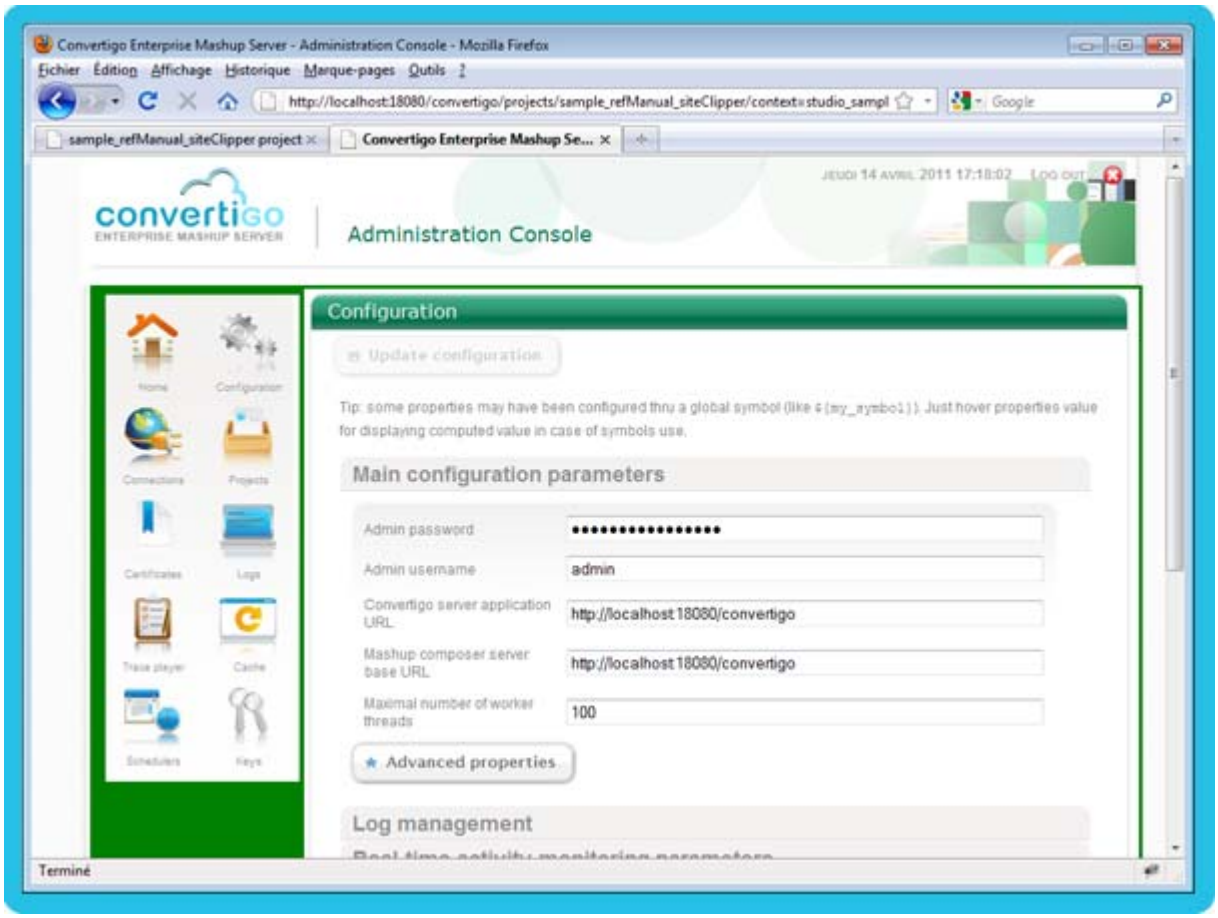


Figure 2 - 594: Request header criterion - Browser after the execution of the `LoginAdmin` transaction in Firefox

Using Firefox browser, we can see that a CSS update, changing background color to green, is performed in the web page. This update is added thanks to a *CSS injector* extraction rule, one is positionned on each screen class.

On the `Firefox` screen class, the CSS background color is changed to `green`. On the `Chrome` screen class, the CSS background color is changed to `cyan`.

Swithch to the test platform of this project in a Chrome web browser. Executing the `LoginAdmin` HTML transaction again, the user gets the browsing control back on the Configuration page of Convertigo administration website, with cyan background.

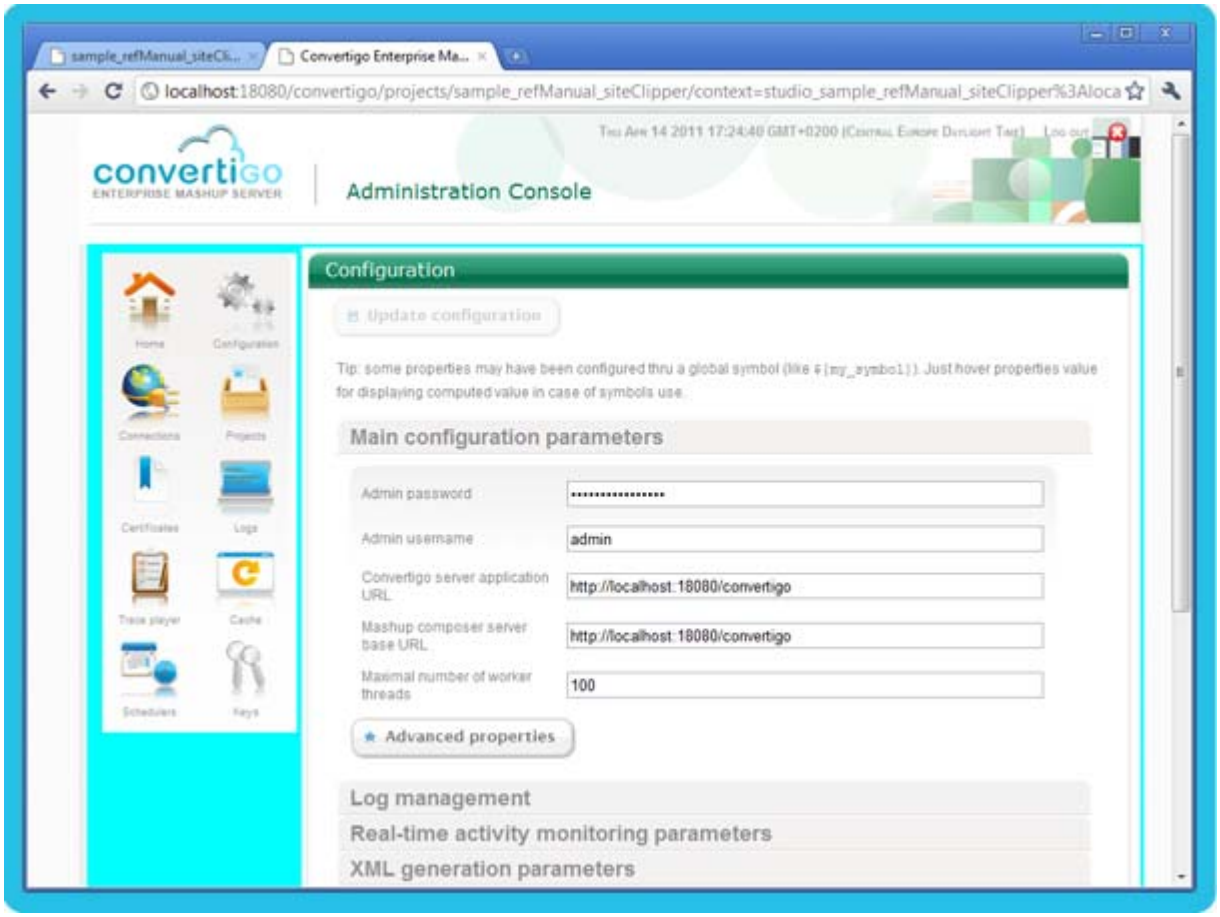


Figure 2 - 595: Request header criterion - Browser after the execution of the `LoginAdmin` transaction in Chrome

Depending on the browser, the background color is changed to a different color, meaning that both screen classes defined by *Request header* criteria are matching for the appropriate case.

For more information about *CSS injector* extraction rule, see "*CSS injector*" extraction rule documentation and examples.

RESPONSE CRITERIA

MIME TYPE 

OBJECT DESCRIPTION

Defines a response criterion based on MIME type for Site Clipper screen classes.

The *MIME type* criterion allows defining a regular expression that is applied on response MIME type. MIME type is a part of `content-type` HTTP header.

Matching condition: The *MIME type* criterion matches when the regular expression defined in **Regular expression** property matches response MIME type, i.e. if the string pattern described by the regular expression is found in the `MIME type` substring of the `content-type` header value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EXAMPLES

Let's consider the Convertigo administration website. In the context of this website clipping, the login phase is automated thanks to an HTML transaction, named `LoginAdmin`, and then the user gets the browsing control back on the Configuration page of Convertigo administration website.

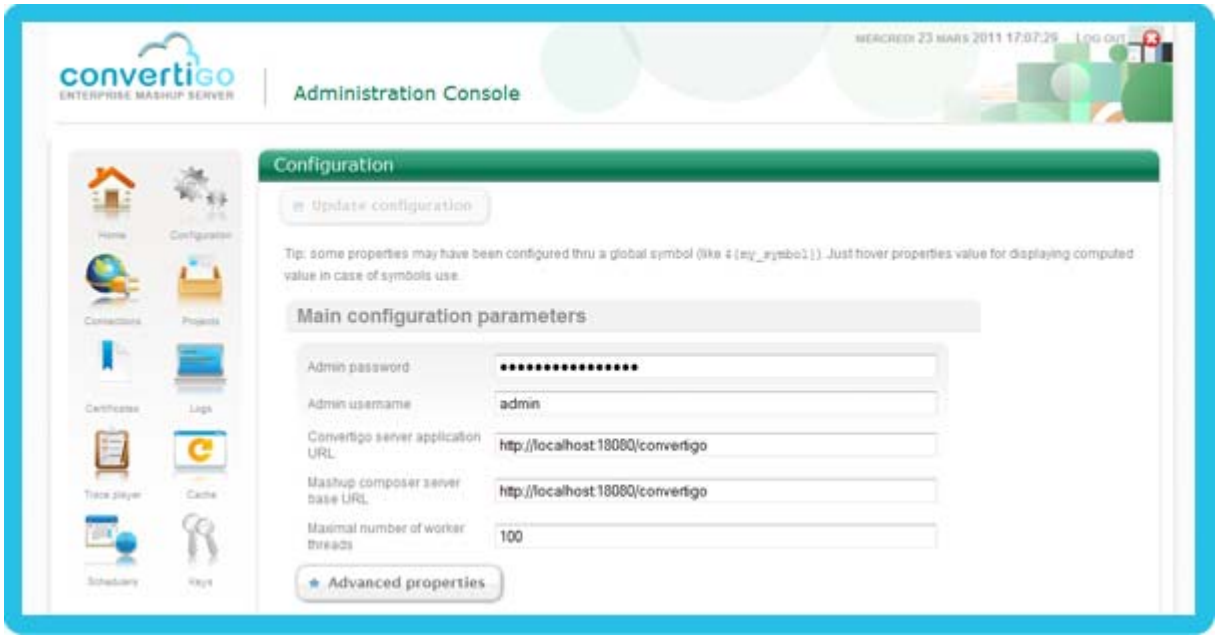


Figure 2 - 596: MIME type criterion - Convertigo Administration Configuration page

A *Site Clipper connector* is defined in the same project for the administration website to be accessed through Convertigo. It contains a screen classes hierarchy, defined thanks to several criteria.

A *MIME type* criterion is created to define a root screen class with the following parameters:

```
MIME type [
   regular expression=text/html
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

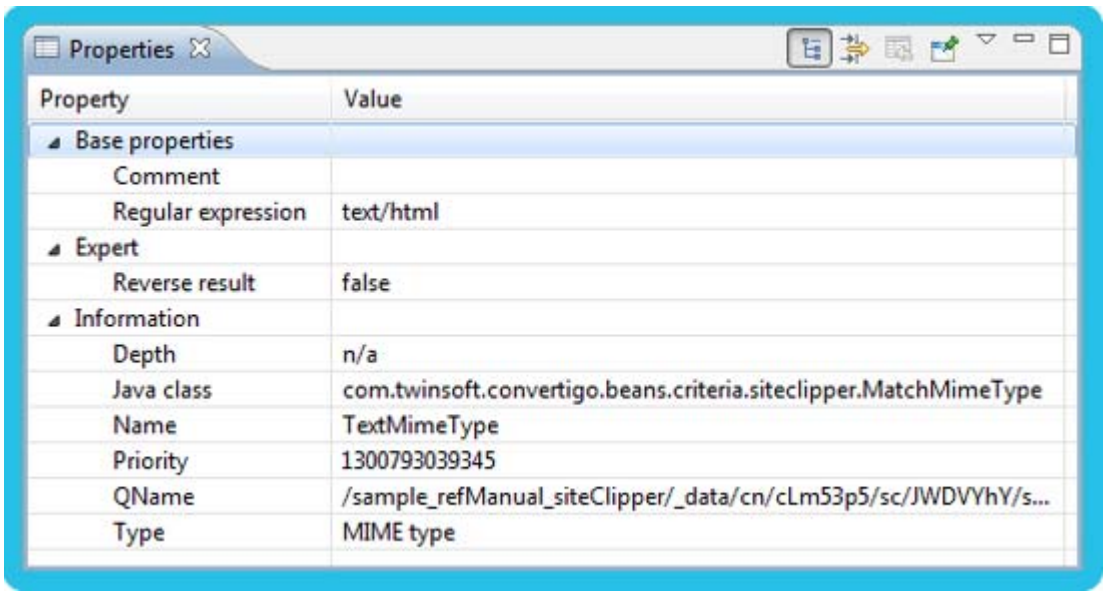


Figure 2 - 597: MIME type criterion - Configuration example

The **Regular expression** property is defined to match every response MIME type containing "`text/html`".

The screen class defined by this criterion is matching every HTML page accessed through the *Site Clipper connector*.

This criterion with the screen class it defines appears as follows in the **Projects** view:

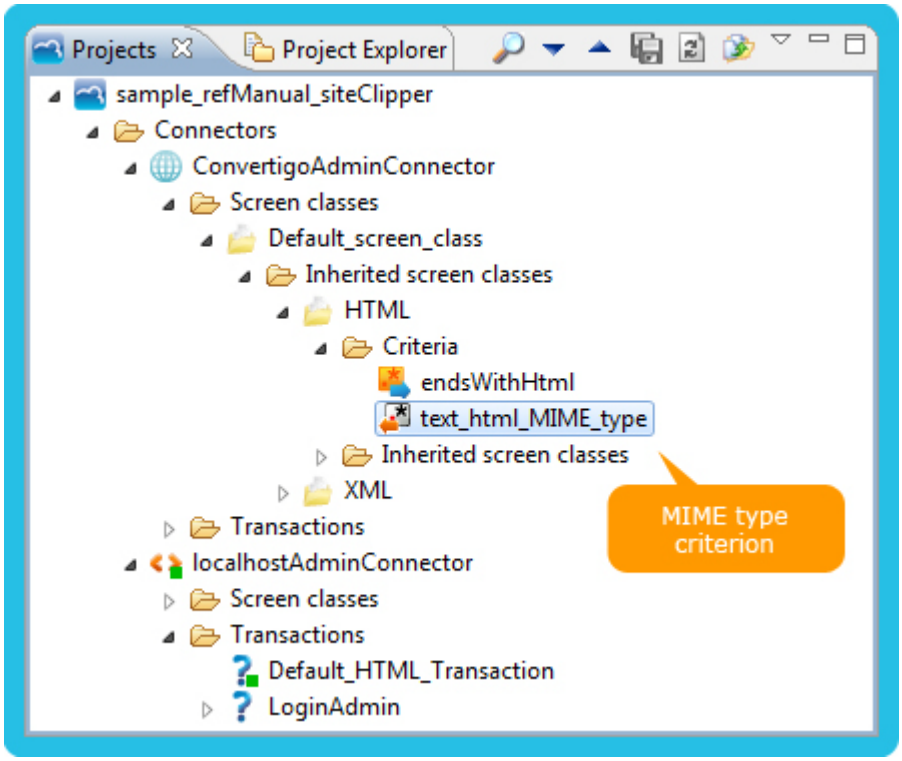


Figure 2 - 598: MIME type criterion - Object in Projects view

Swithching to the test platform of this project in a web browser, when executing the `LoginAdmin` HTML transaction, the user gets the browsing control back on the Configuration page of Convertigo administration website.

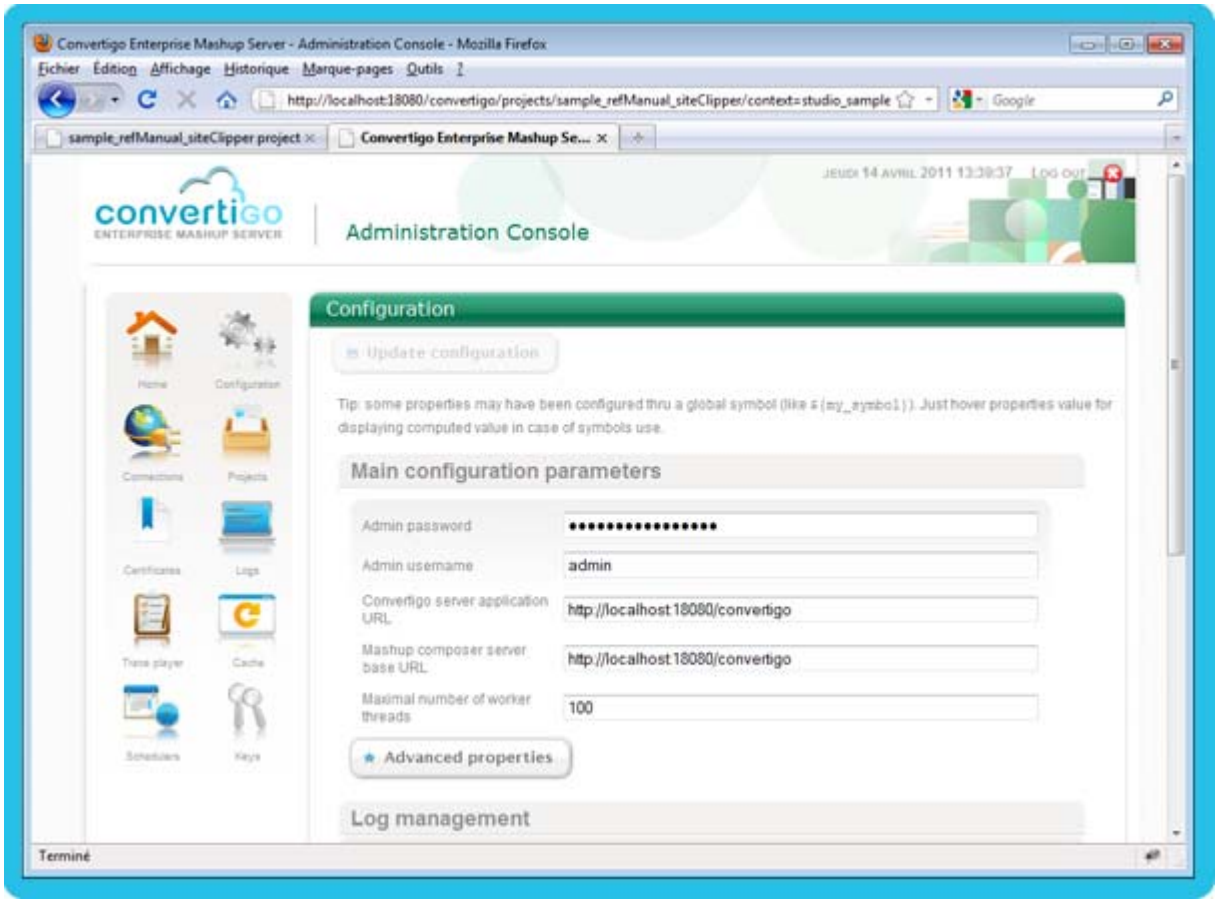


Figure 2 - 599: MIME type criterion - Browser after the execution of the `LoginAdmin` transaction

The *MIME type* criterion matches on this resource.

Regular expression (SiteClipper) 

Object Description

Defines a response criterion based on a regular expression on data content for Site Clipper screen classes.

The *Regular expression* criterion allows defining a regular expression that is applied on response data content.

Matching condition: The *Regular expression* criterion matches when the regular expression defined in **Regular expression** property matches response data content, i.e. if the string pattern described by the regular expression is found in the response data.

Object Properties

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

Examples

Let's consider the Convertigo administration website. In the context of this website clipping, the login phase is automated thanks to an HTML transaction, named `LoginAdmin`, and then the user gets the browsing control back on the Configuration page of Convertigo administration website.

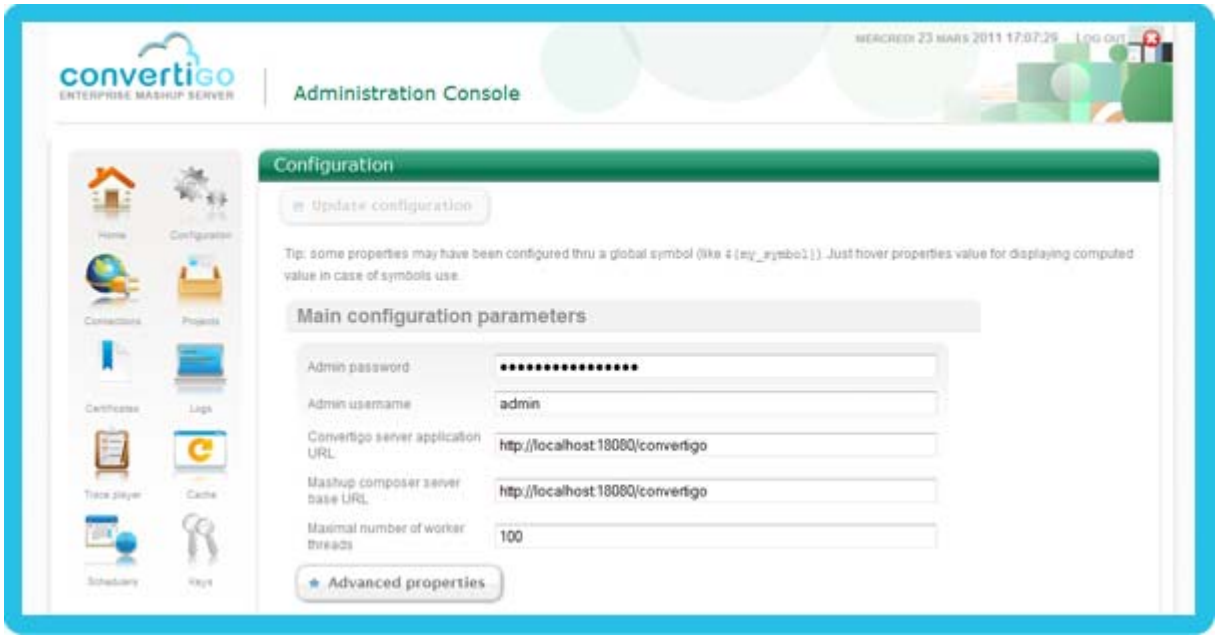


Figure 2 - 600: Regular expression criterion - Convertigo Administration Configuration page

A *Site Clipper connector* is defined in the same project for the administration website to be accessed through Convertigo. It contains a screen classes hierarchy, defined thanks to several criteria.

A root screen class is defined to match HTML pages that are accessed through Convertigo. Inherited screen classes are defined to differentiate localhost named server from local IP calls in the URL.

In "localhost" branch, a child screen class, named `AdministrationConsole`, is defined thanks to a response *Regular expression* criterion, with the following parameters:

```
Regular expression [
   regular expression=Administration Console
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

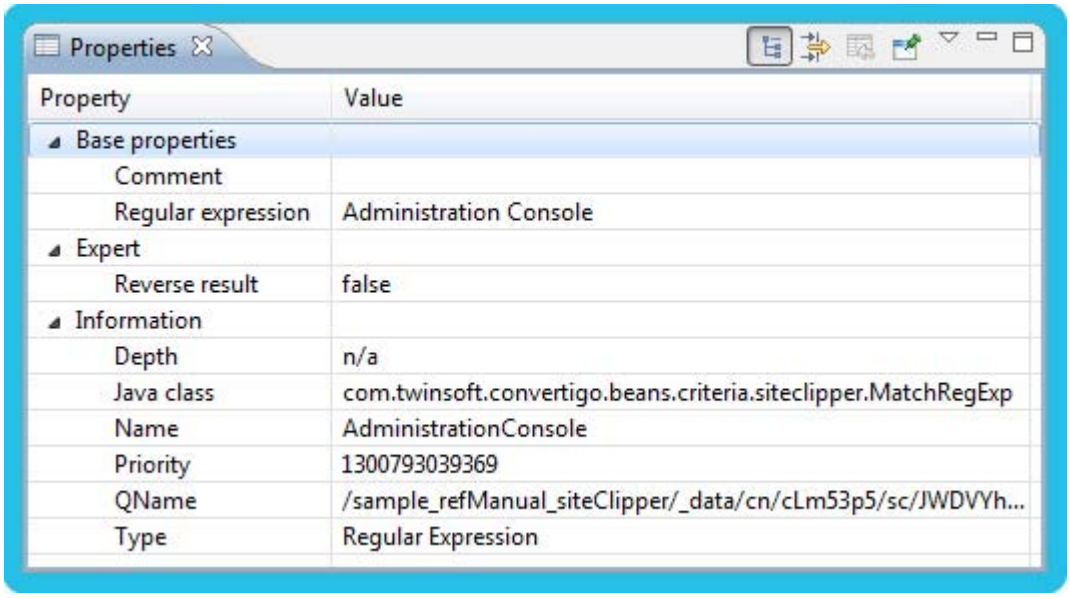


Figure 2 - 601: Regular expression criterion - Configuration example

The **Regular expression** property is defined to match every response page content containing "`Administration Console`".

As "Administration Console" text is appearing as title in every page from the Convertigo server Administration, the screen class defined by this criterion is matching every page from Convertigo Administration website accessed through the *Site Clipper connector*, possibly connected by the `LoginAdmin` HTML transaction.

To sum up, the screen class defined by this criterion is matching every Convertigo Administration (this screen class criterion) HTML web page (criteria inherited from root screen class) from localhost domain (criterion inherited from parent screen class), accessed through the *Site Clipper connector*.

This criterion with the screen class it defines appears as follows in the **Projects** view:

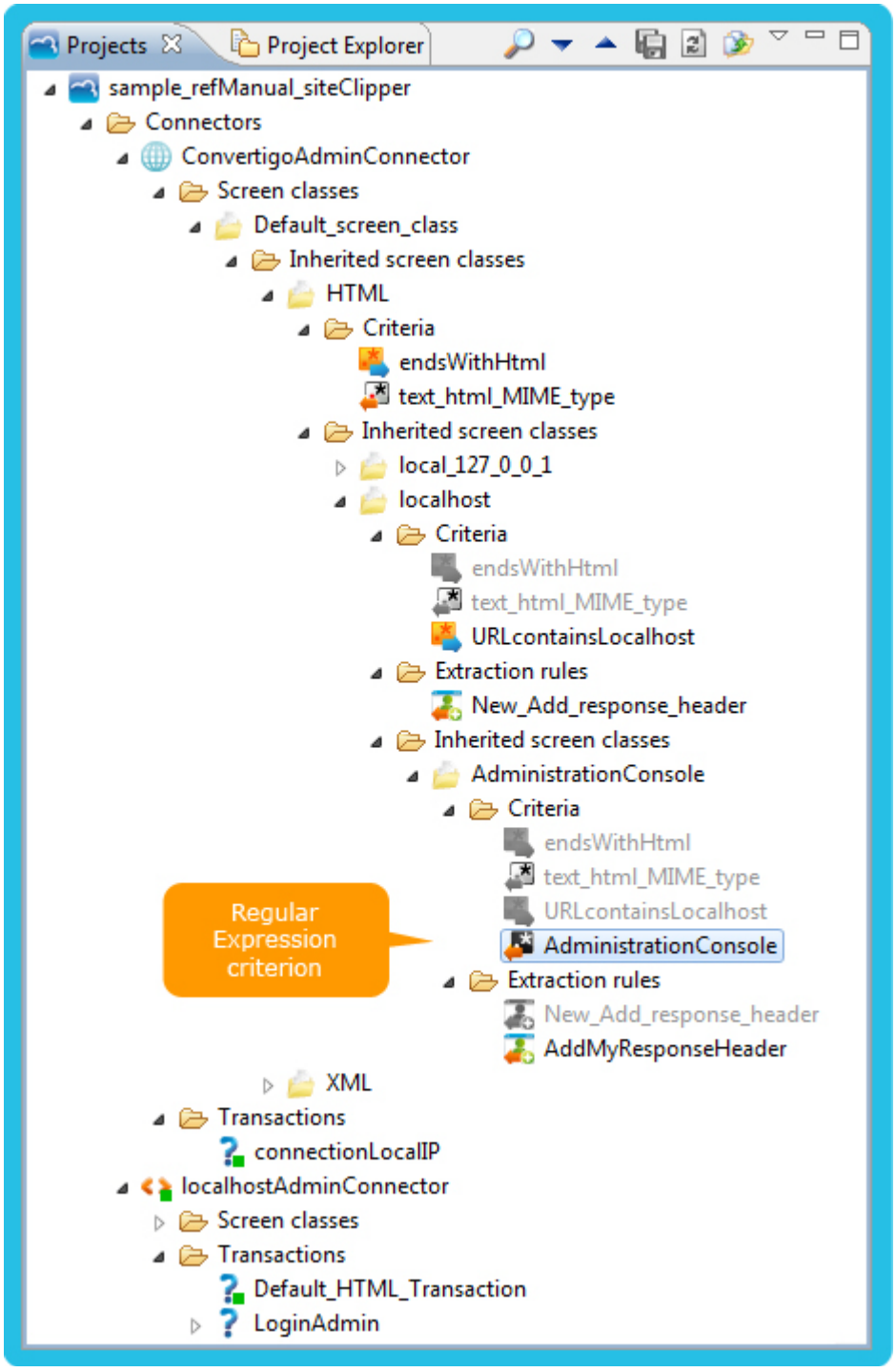


Figure 2 - 602: Regular expression criterion - Object in Projects view

Swith to the test platform of this project in a Firefox web browser, activate Firebug extension. Executing the `LoginAdmin` HTML transaction (in a new tab thanks to **Execute full screen** button), the user gets the browsing control back on the Configuration page of Convertigo administration website.

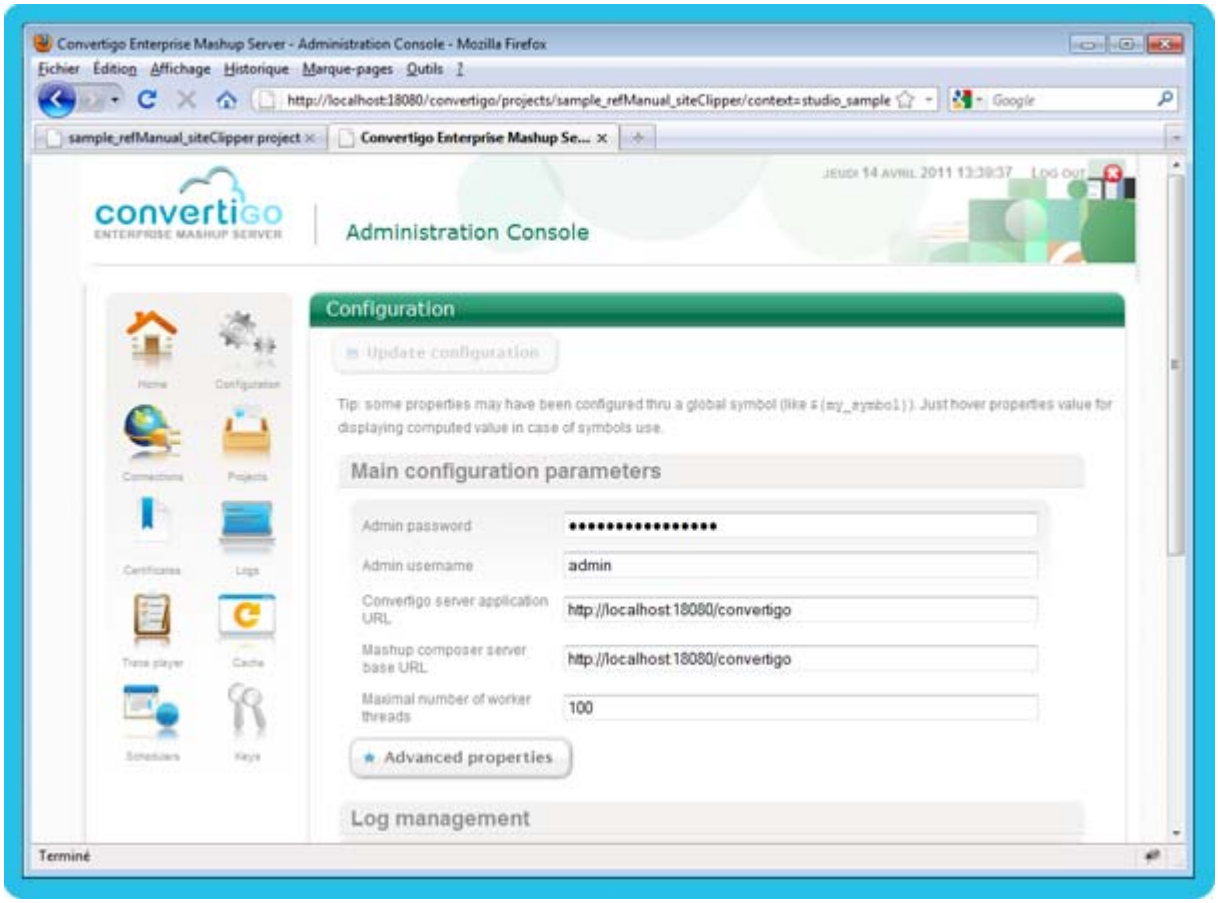


Figure 2 - 603: Regular expression criterion - Browser after the execution of the `LoginAdmin` transaction

In Firebug, we can see that response HTTP headers, named `myConvertigoHeader` and `myConvertigoHeader2`, were added by Convertigo to the HTML page resource. These HTTP headers are added thanks to *Add response header* extraction rules, one is positionned on each screen class. The header values are parametered with the screen classes names, which differ between screen classes. Thus, these headers values reflect the detected screen classes tree.

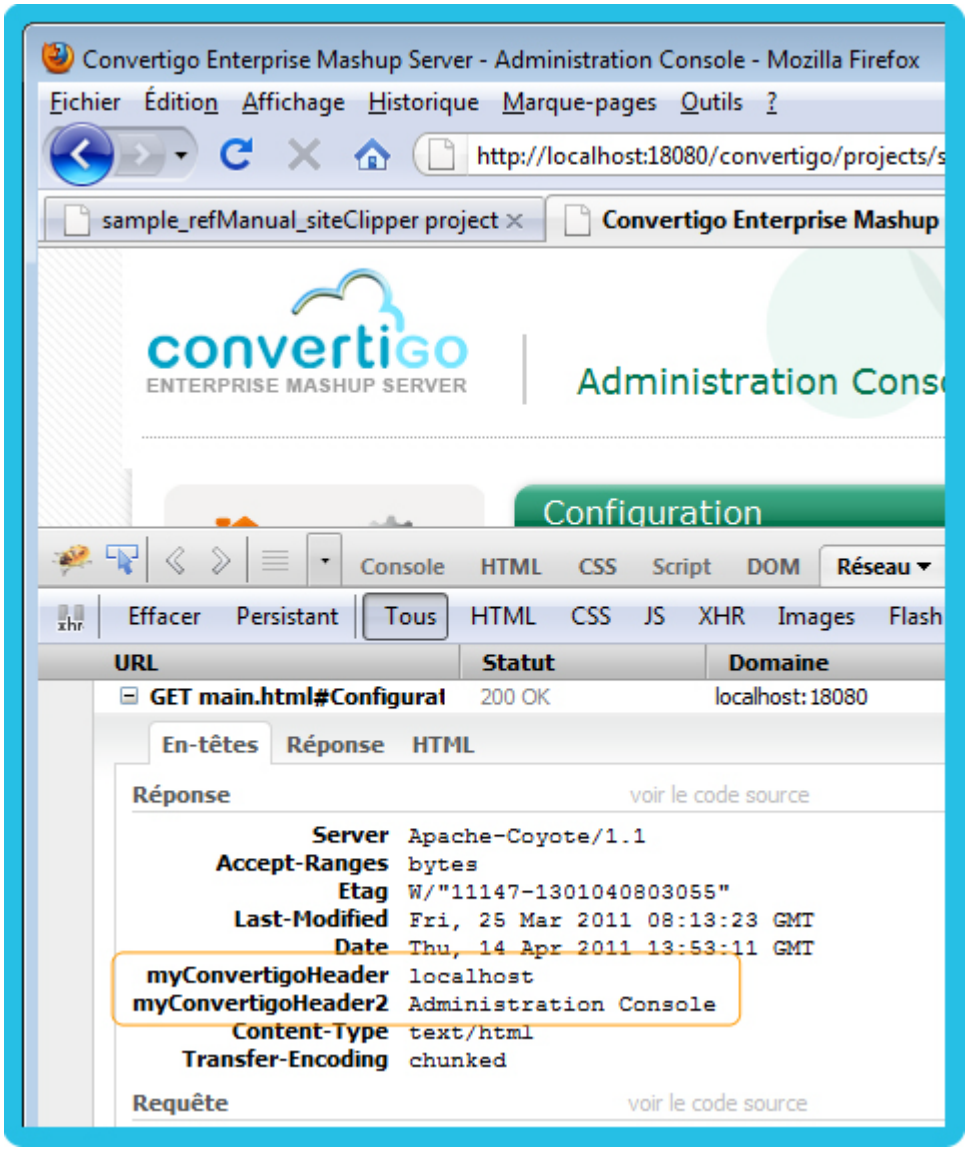


Figure 2 - 604: Regular expression criterion - Added HTTP header containing detected screen class names

In the test browser, the header values are "`localhost`" and "`Administration Console`", meaning that the screen class defined by the *Regular expression* criterion has matched.

For more information about *Add response header* extraction rule, see "*Add response header*" extraction rule documentation and examples.

RESPONSE HEADER 

OBJECT DESCRIPTION

Defines a response criterion based on HTTP headers for Site Clipper screen classes.

The *Response header* criterion allows defining a regular expression that is applied on a response header. Response header to test is defined in **Header name** property.

Matching condition: The *Response header* criterion matches when the regular expression defined in **Regular expression** property matches defined response header, i.e. if the string pattern described by the regular expression is found in the response header value.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	standard	Defines the name of the header which content is tested against the regular expression.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EXAMPLES

Let’s consider the Convertigo test platform Home page, listing available projects and giving the ability to access several applications such as Convertigo Administration website, Developer network website, etc.

Figure 2 - 605: Response header criterion - Convertigo test platformHome page

In the context of a *Site Clipper connector*, a transaction, named `connectionLocalIP`, is defined to connect to this page. A screen classes hierarchy is defined thanks to several criteria to identify accessed data and resources.

A screen class is defined to match XML resources UTF-8 encoded. Such resources are transfered in order to retrieve information about the Convertigo platform. These information are displayed on the top of the page (Convertigo version, Engine version, etc.).

To identify this screen class, a *Response header* criterion is created with the following parameters:

```
Response header [
   header name=Content-Type
   regular expression=text/xml;charset=UTF-8
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

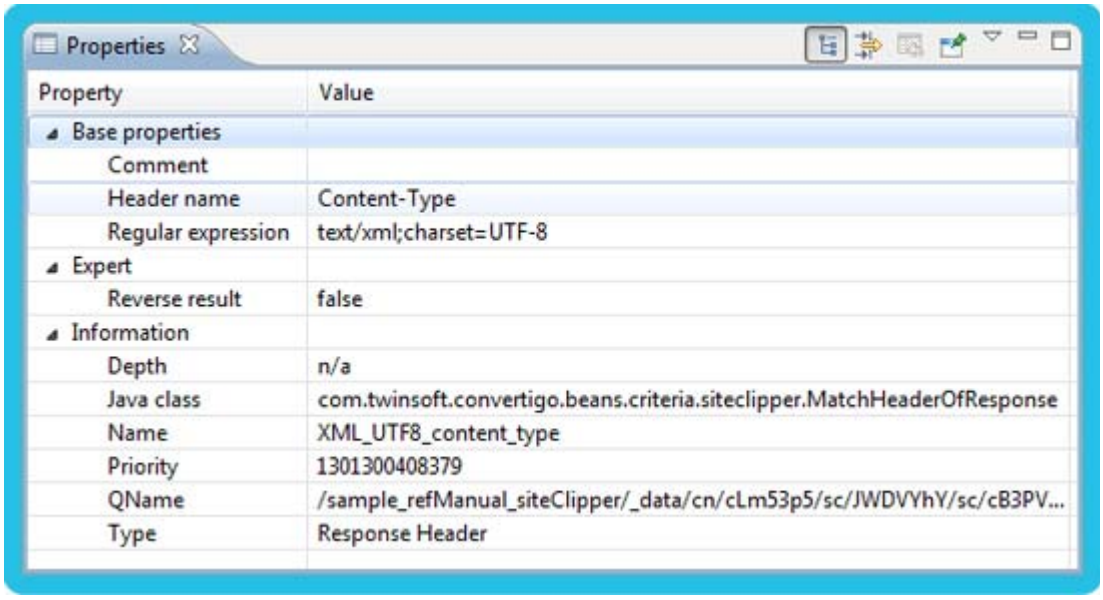


Figure 2 - 606: Response header criterion - Configuration example

The **Header name** property is defined to analyse content of `Content-Type` header, and the **Regular expression** property is set to find "`text/xml;charset=UTF-8`" content in this header, which corresponds to finding the text/XML MIME type, plus an UTF-8 charset encoding.

This criterion with the screen class it defines appears as follows in the **Projects** view:

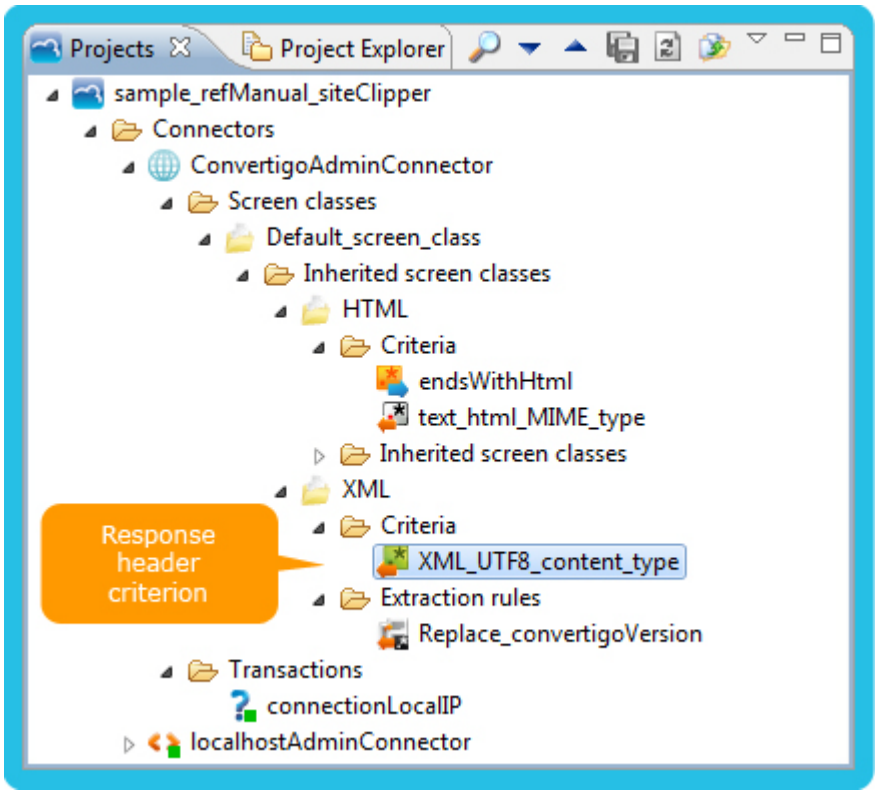


Figure 2 - 607: Response header criterion - Objects in Projects view

Swith to the test platform of this project in a web browser. Executing the `connectionLocalIP` Site Clipper transaction (in a new tab thanks to **Execute full screen**

button), the user gets the browsing control back on the Convertigo Projects page.

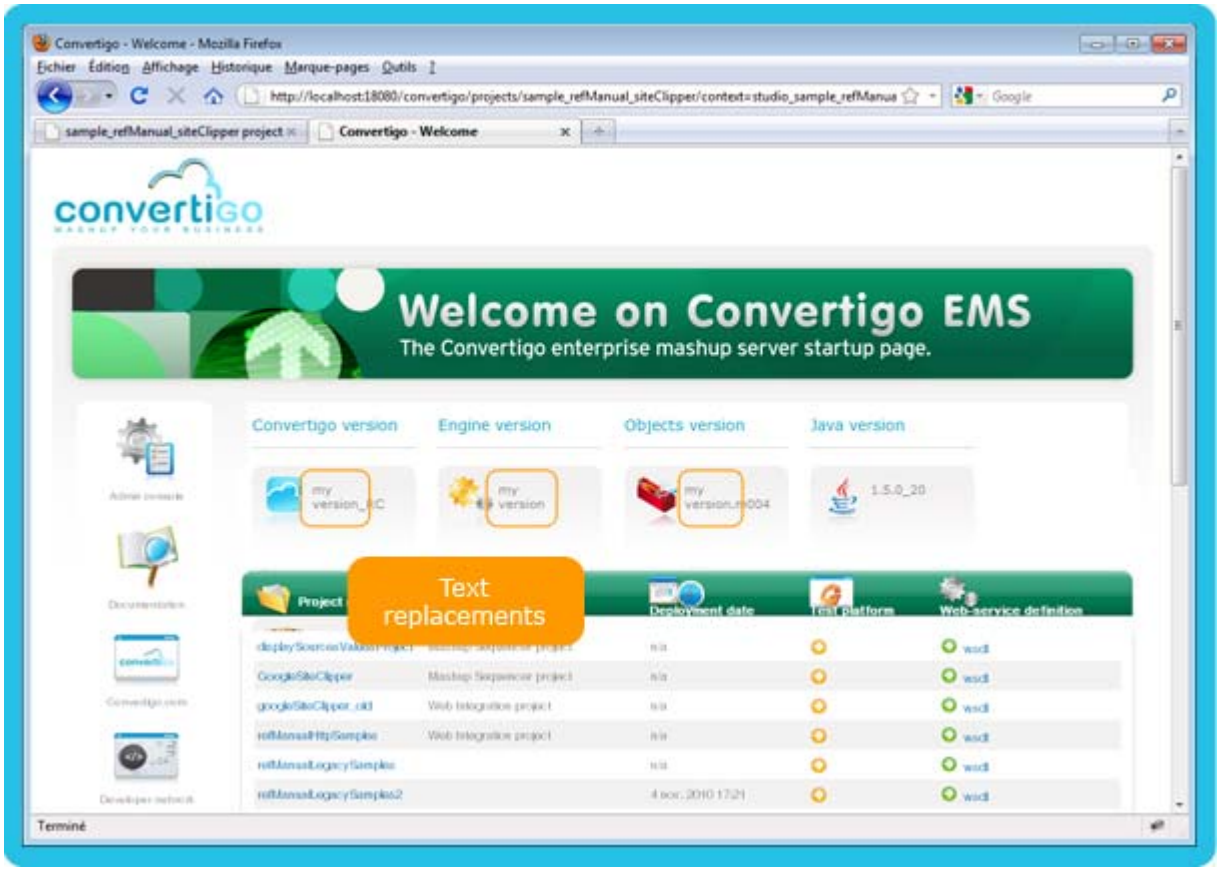


Figure 2 - 608: Response header criterion - Browser after the execution of the `connectionLocalIP` transaction

Using Firefox browser, we can see that replacements have been performed in data contained in the XML resource: Convertigo actual version is replaced by "`my version`" text. These replacements are performed thanks to a *Replace string* extraction rule, positionned on the screen class defined thanks to the *Response header* criterion. This replacements being performed, this means that the screen class defined by the *Response header* criterion has matched.

For more information about *Replace string* extraction rule, see "*Replace string"* extraction rule documentation and examples.

STATUS-CODE 

OBJECT DESCRIPTION

Defines a response criterion based on HTTP Status-Code for Site Clipper screen classes.

The *Status-Code* criterion allows defining a regular expression that is applied on response Status-Code.

Matching condition: The *Status-Code* criterion matches when the regular expression defined in **Regular expression** property matches response Status-Code, i.e. if the string pattern described by the regular expression is found in the Status-Code of the HTTP response.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	standard	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Regular expression	String	standard	Defines the regular expression to match. This property allows defining a regular expression as a string pattern. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Reverse result	boolean	expert	Defines if the criteria's result should be reversed. When a criteria is evaluated, it can sometimes be useful to get the opposite of the standard result (i.e. when the criteria matches, its result is false, and when it doesn't match, its result is true). Use this property to reverse the standard result. For example, you may define a screen class that does not contain the text "Hello" in white on black background. For that, you define a criterion matching on the text "Hello" in white on black background, and you reverse it thanks to this property.

EXAMPLES

Let's consider the Apple website. In the context of this website clipping, a `404 Not Found` Status-Code page can be accessed.

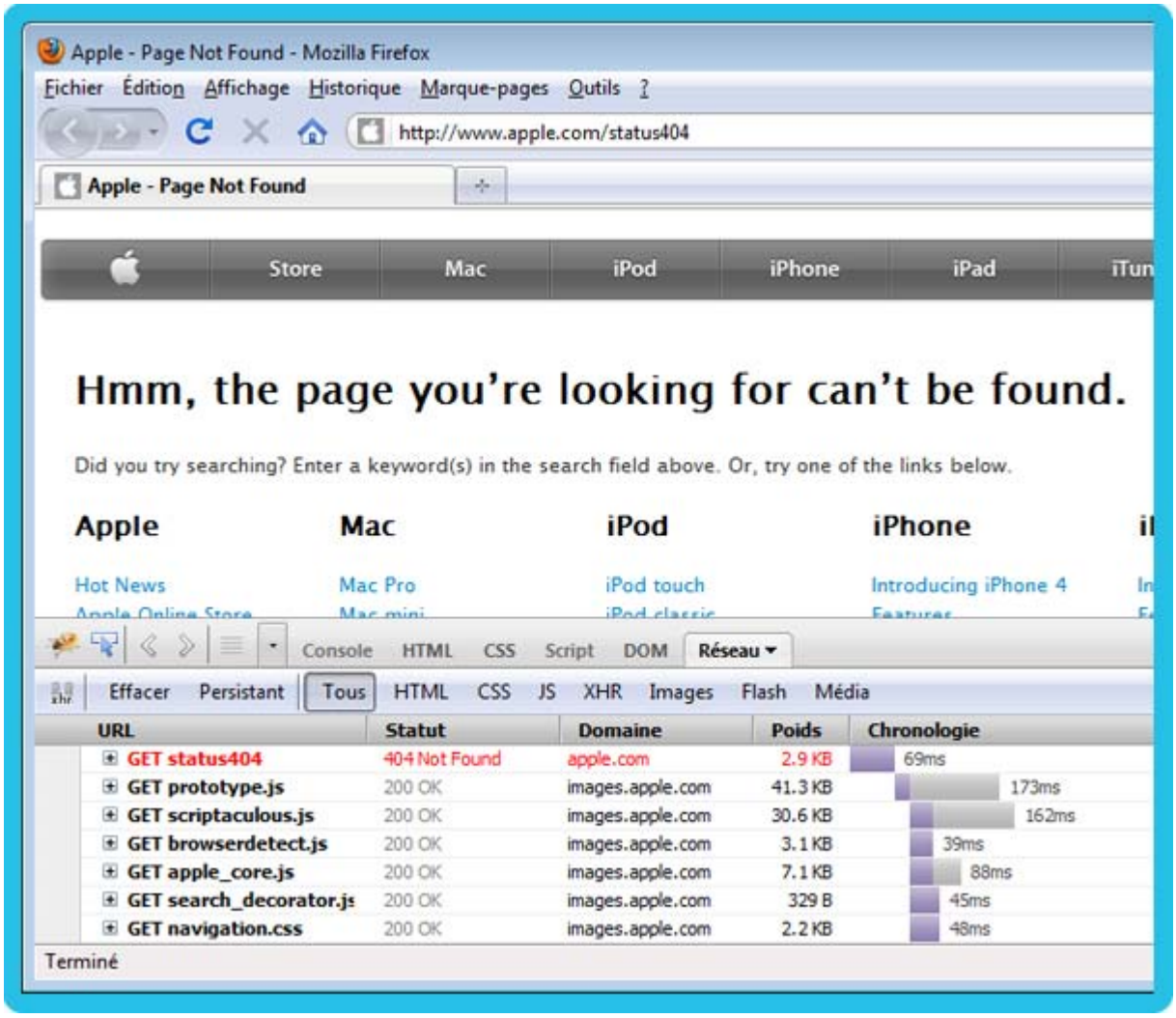


Figure 2 - 609: Status-Code criterion - Apple website 404 Not Found page

In order to demonstrate the Status-Code criterion, the access to this `404 Not Found` page is automated thanks to a *Site Clipper transaction*, named `Access404Page.` Then, the user gets the browsing control back on the Apple website.

This transaction is implemented in a *Site Clipper connector*, which also contains a screen classes hierarchy identifying accessed data and resources. A root screen class is defined to match `apple.com` website pages accessed through Convertigo thanks to an *URL* criterion.

A child screen class, named `Apple_404`, is defined thanks to a response *Status-Code* criterion, with the following parameters:

```
Status-Code [
   regular expression=404
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

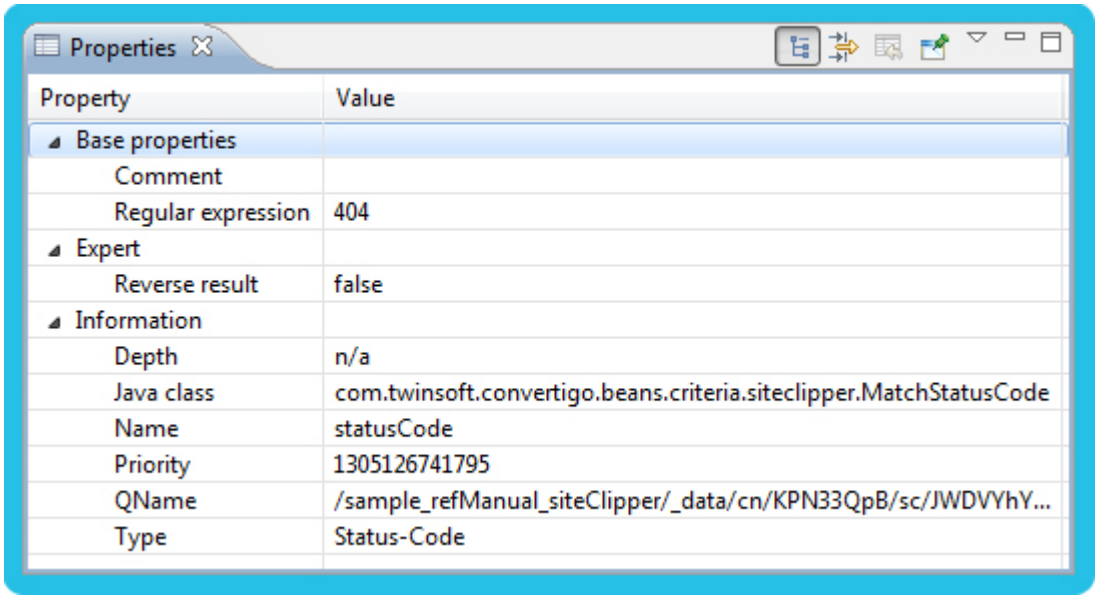


Figure 2 - 610: Status-Code criterion - Configuration example

The **Regular expression** property is defined to a simple string pattern containing the Status-Code number "`404`".

The screen class defined by this criterion is matching every page from Apple website, accessed through the *Site Clipper connector*, with the Status-Code containing "`404`".

This criterion with the screen class it defines appears as follows in the **Projects** view:

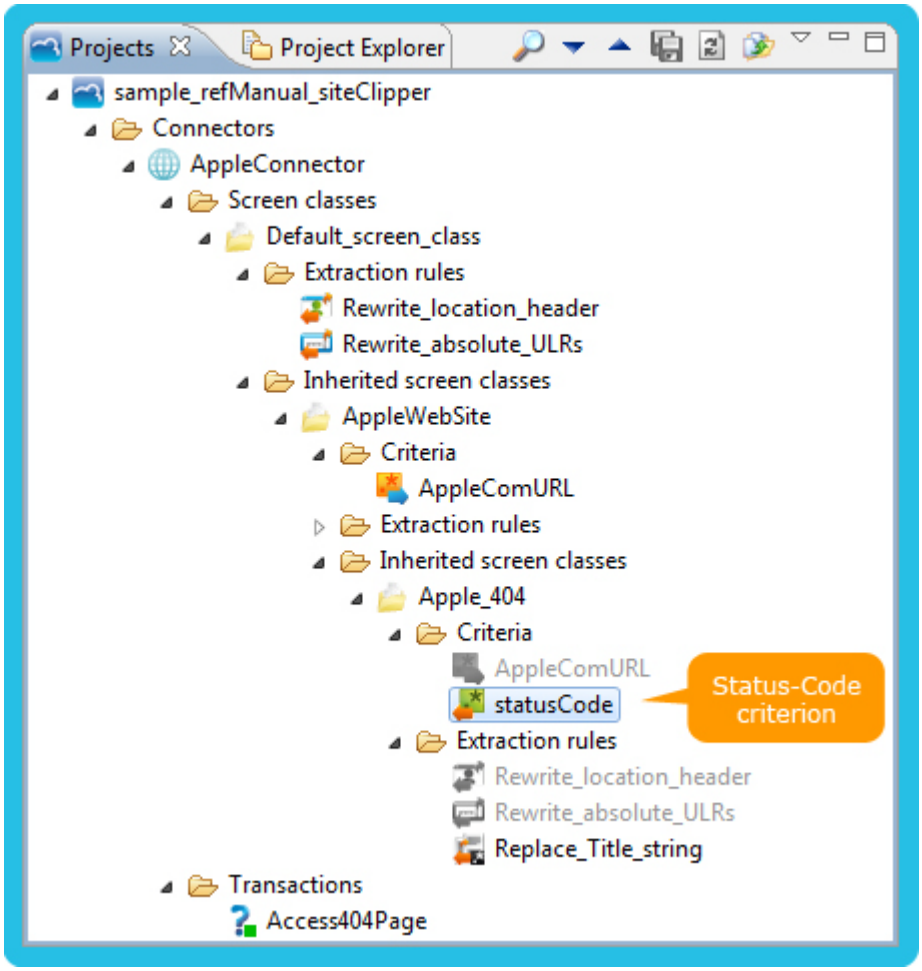


Figure 2 - 611: Status-Code criterion - Object in Projects view

We can see that a *Replace string* extraction rule is positionned on the screen class. It is parametered to replace the title usually displayed on the page by another title string. Thus, this modified title reflects the detection of the screen class.

Swithching to the test platform of this project in a web browser, when executing the `Access404Page` *Site Clipper transaction*, the user gets the browsing control back on the Apple `404 Not Found` web page, modified by Convertigo:

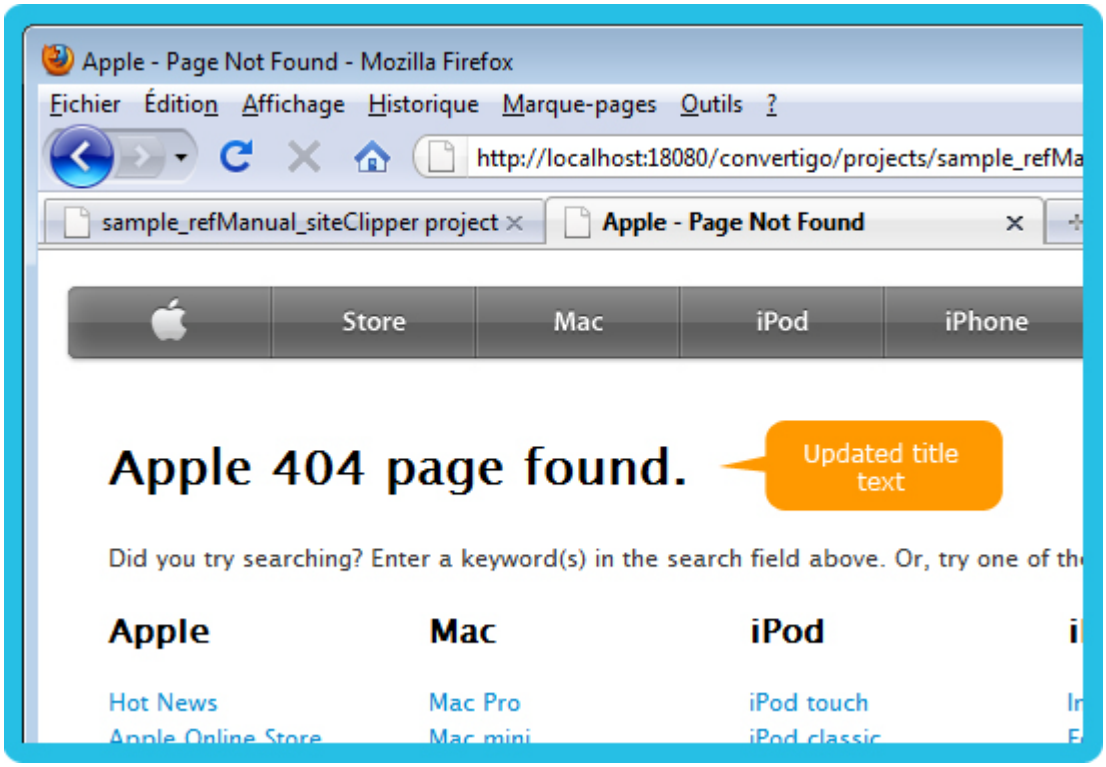


Figure 2 - 612: Status-Code criterion - Browser after the execution of the Access404Page transaction

The title text was modified by Convertigo, the screen class was detected thanks to the *Status-Code* criterion.

2.10.3 Rules

REQUEST RULES

ADD REQUEST HEADER 

OBJECT DESCRIPTION

Adds an HTTP header to a Site Clipper request.

The *Add request header* extraction rule adds the header defined in **Header name** property to the request's headers.

The value set in this header is defined by the **Header value** property.

If the header defined in the **Header name** property exists in the request, the *Add request header* extraction rule has no effect.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	configuration	Defines the header name. This property allows to define the name of the header to add to/modify in/remove from the request or the response which headers are manipulated.
Header value	String	configuration	Defines the header value. This property allows to define the value to set in the header defined by Header name property.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Example 1

Let’s consider the website “request.urih.com” which welcome page displays the headers of the http request that it receives.

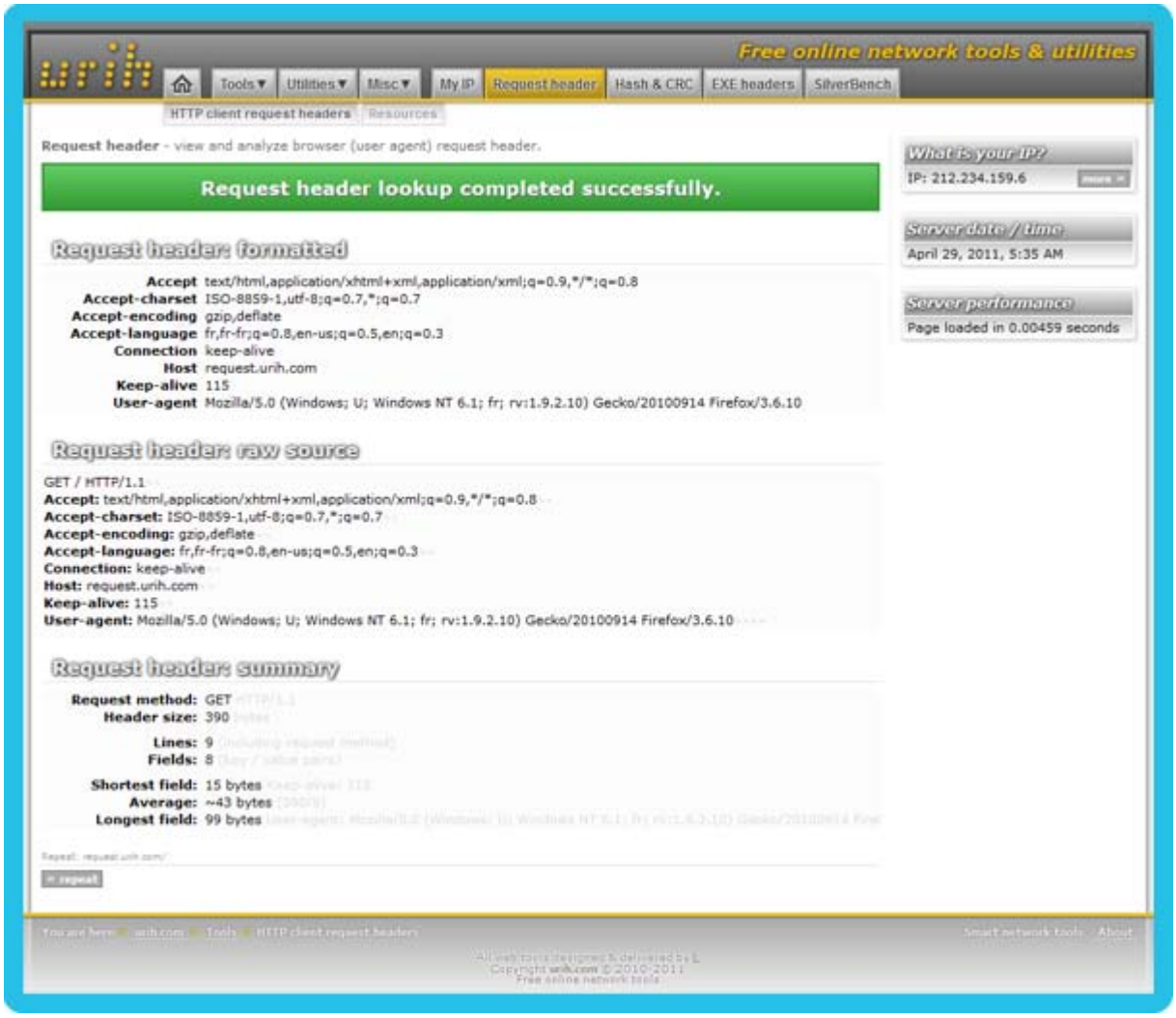


Figure 2 - 613: Add request header extraction rule - request.urih.com website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically add a header to the request to HTML resources that are accessed through Convertigo.

A transaction, named `Uri_headers_transaction`, is defined as default transaction for the *Site Clipper connector* named `RequestHeadersConnector`. It defines the URL `http://request.urih.com` as target URL to connect to the previously described web page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources.

A screen class, named `HTML`, is defined thanks to a response *MIME type* criterion to handle HTML resources. On this screen class, an *Add request header* extraction rule is added in order to add a new request header.

The rule is created with the following parameters:

```
Add request header [
   header name=convertigo
   header value=convertigo-header
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

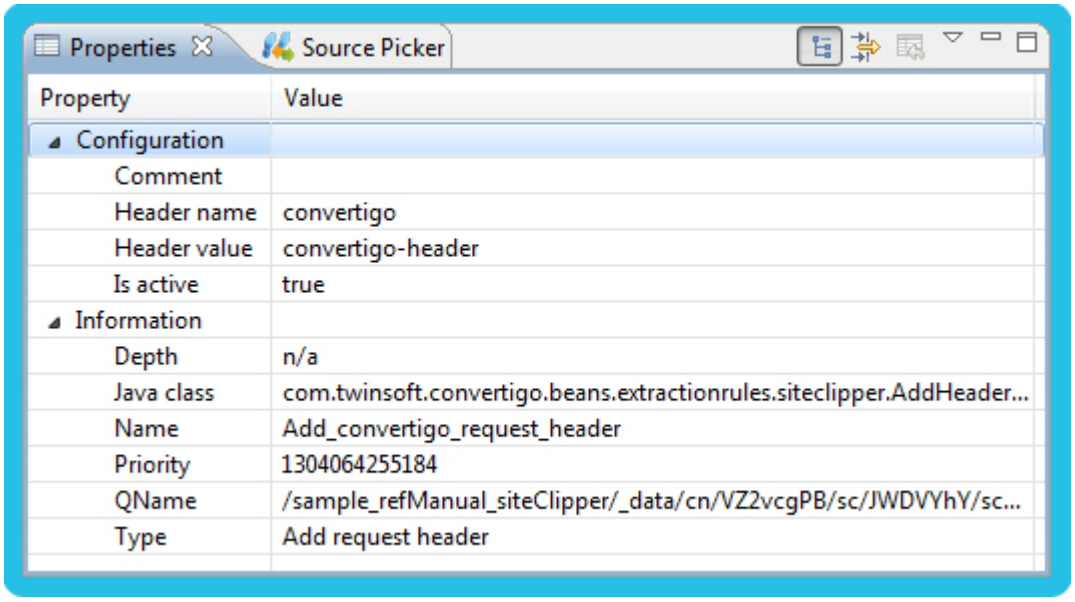


Figure 2 - 614: Add request header extraction rule - Configuration example

The extraction rule appears as follows in the **Projects** view:

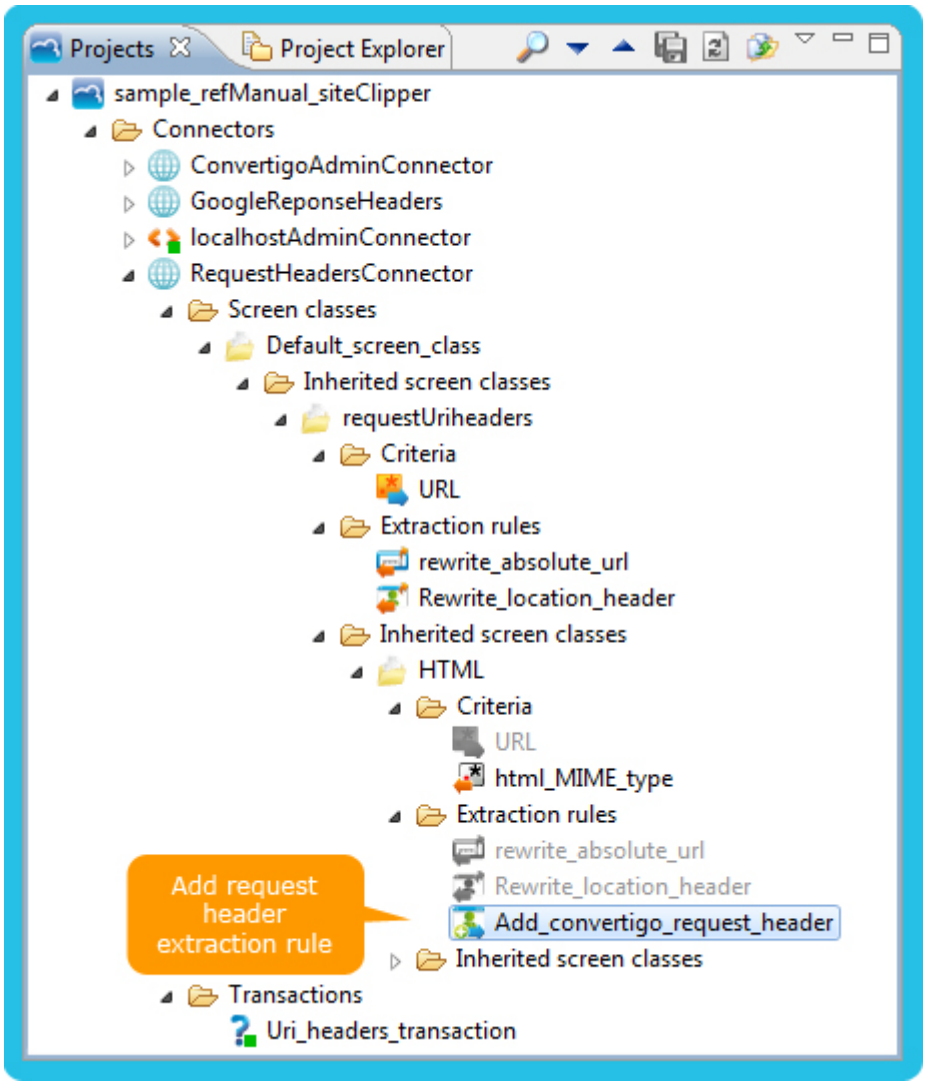


Figure 2 - 615: Add request header extraction rule - Object in Projects view

Switch to a web browser displaying the test platform of this project (for example Firefox). Executing the `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main page.

The website displays the request HTTP headers that it receives. We can see that the request HTTP header named `convertigo` and containing the defined value `convertigo-header` was added to the request to the HTML page:

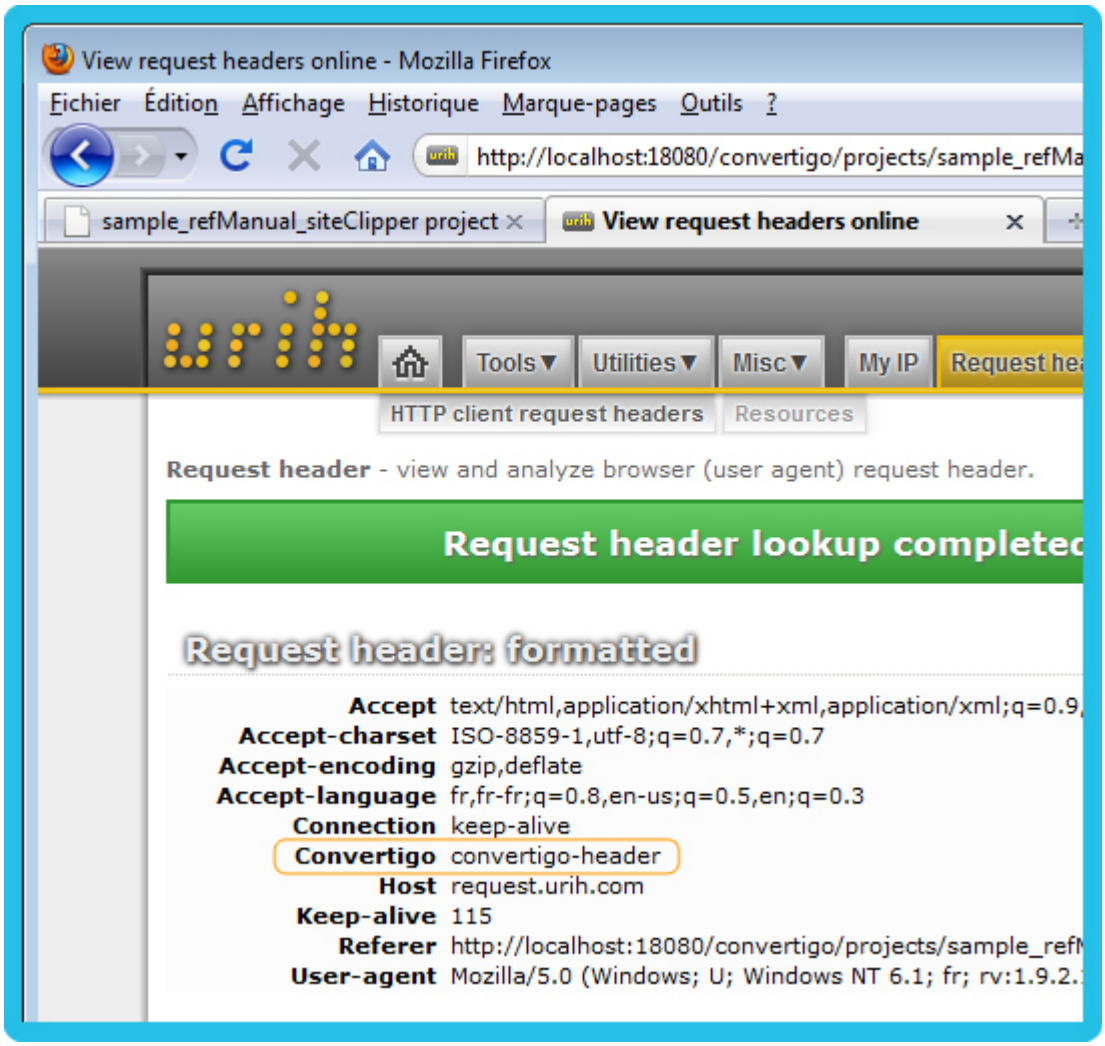


Figure 2 - 616: Add request header extraction rule - Header added to the request to the website page

Example 2

The purpose of this second example is to show that an already existing request header cannot be added again nor changed the value by the *Add request header* extraction rule.

Let's consider the same context and project as the first example.

In the web browser, we can see the existing `accept-encoding` header appearing with the "`gzip,deflate`" value received when accessing the HTML page resource:

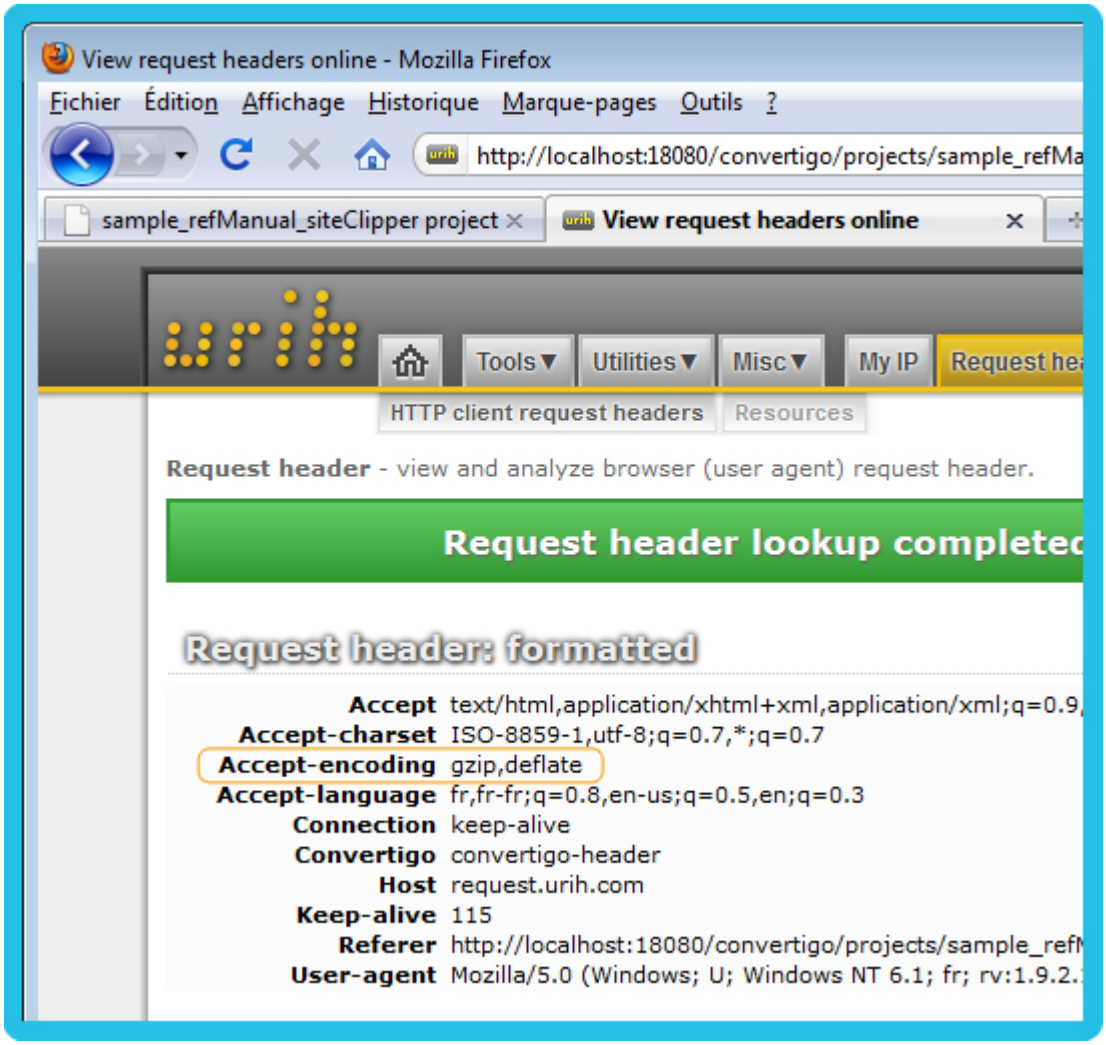


Figure 2 - 617: Add request header extraction rule - accept-encoding header existing on the request to the web page

A screen class, named `hasAcceptEncoding`, is defined as inherited from previous `HTML` screen class. It uses a *Request header* criterion parametered to detect `accept-encoding` header, regardless of its value.

On this screen class, an *Add request header* extraction rule is added in order to add this `accept-encoding` request header again. It is created with the following parameters:

```
Add request header [
  header name=accept-encoding
  header value=my-convertigo-encoding
]
```

A second *Add request header* extraction rule is added in order to add a new request header to be sure the screen class has been detected. It is created with the following parameters:

```
Add request header [
  header name=hasAcceptEncoding
  header value=hasAcceptEncoding
]
```

For both extraction rules, the parameters are edited in the **Properties** view of the Convertigo Studio:

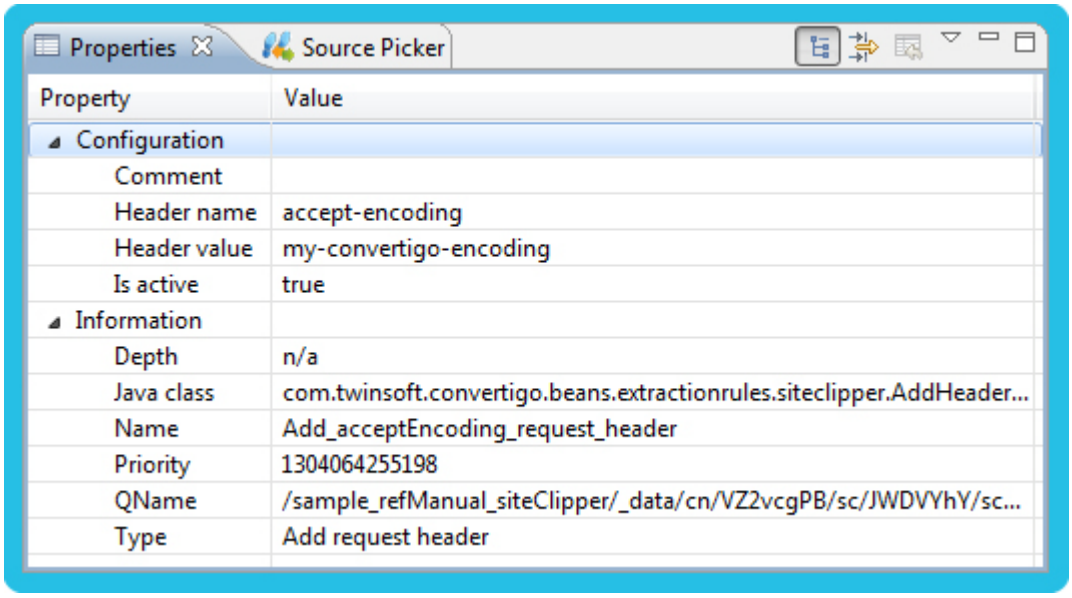


Figure 2 - 618: Add request header extraction rule - Configuration example

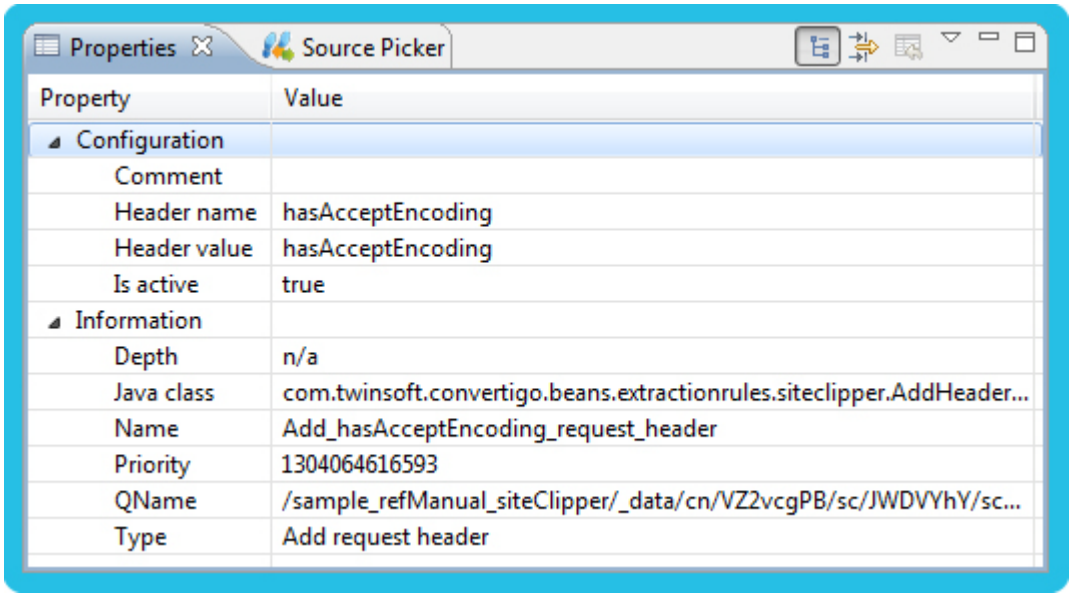


Figure 2 - 619: Add request header extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

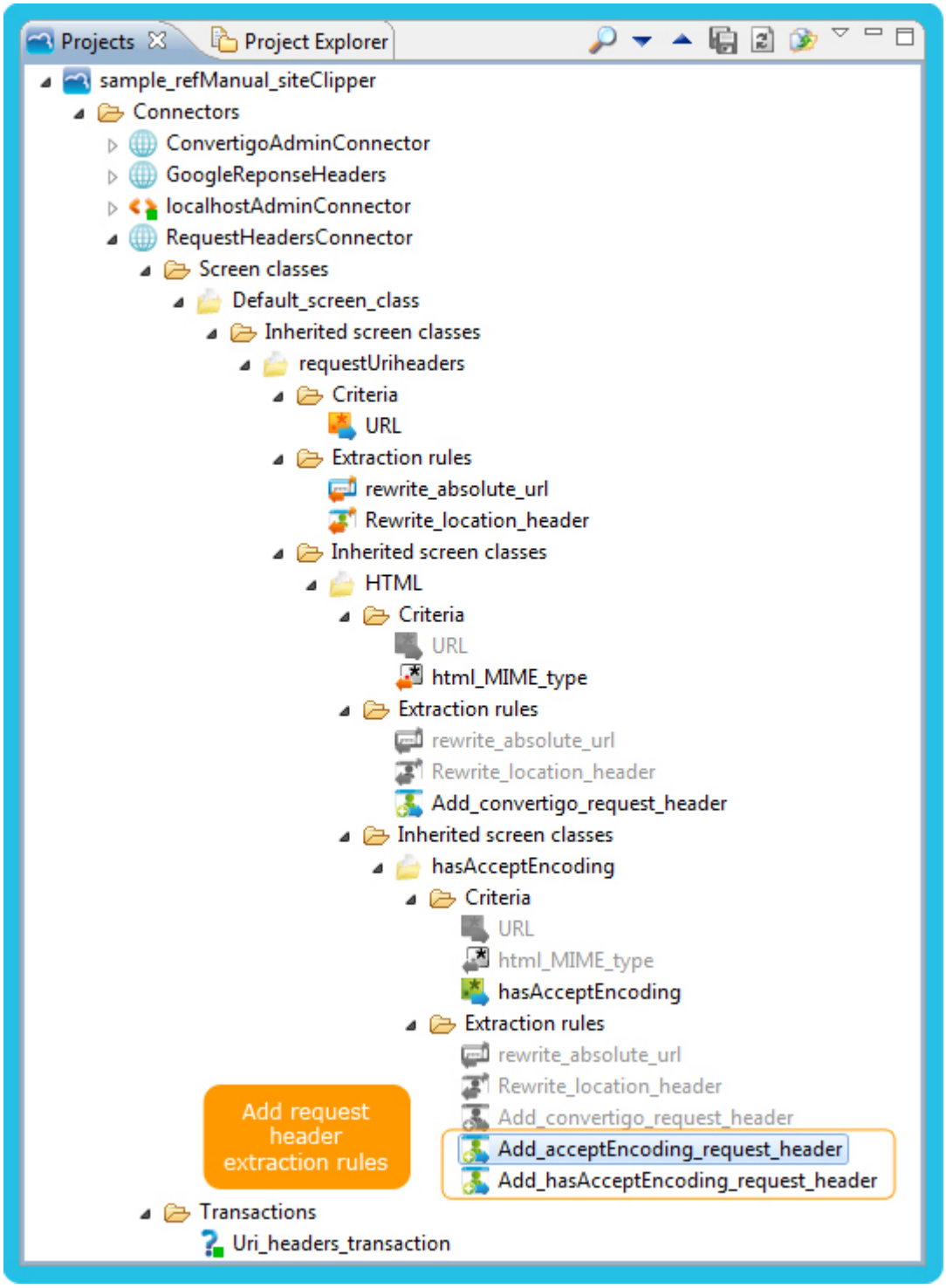


Figure 2 - 620: Add request header extraction rule - Objects in Projects view

Switch back to the web browser displaying the test platform of this project. Executing the `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main page.

The website displays the request HTTP headers that it receives. We can see that the request HTTP header named `hasAcceptEncoding` is added to the request to the HTML page, meaning that the `hasAcceptEncoding` screen class has matched. The already existing `accept-encoding` header is not added again by the rule and is left unchanged:

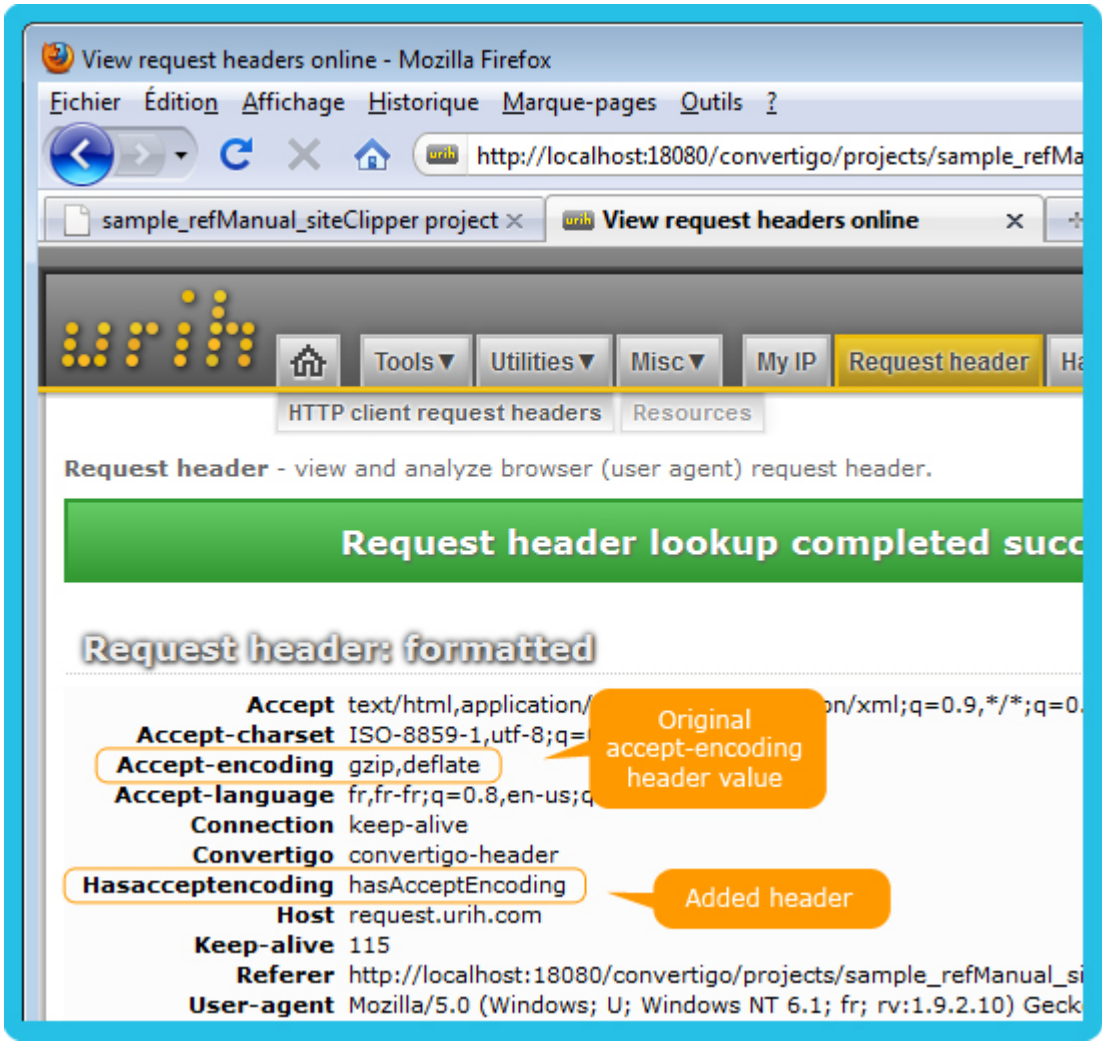


Figure 2 - 621: Add request header extraction rule - Header added and header not modified on the request to the web page

The *Add request header* extraction rule doesn't change the value of existing headers nor add a second header with the same name. If you want to modify an existing header, use a *Modify request header* extraction rule instead. For more information about this extraction rule, see "*Modify request header*" extraction rule documentation and examples.

MODIFY REQUEST HEADER 

OBJECT DESCRIPTION

Modifies an HTTP header in a Site Clipper request.

The *Modify request header* extraction rule modifies an existing header from a request. The name of the header to modify is defined by the **Header name** property.

The new value to set in this header is defined by the **Header value** property.

If the header defined in the **Header name** property doesn't exist in the request, the *Modify request header* extraction rule has no effect.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	configuration	Defines the header name. This property allows to define the name of the header to add to/modify in/remove from the request or the response which headers are manipulated.
Header value	String	configuration	Defines the header value. This property allows to define the value to set in the header defined by Header name property.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Example 1

Let’s consider the website “request.urih.com” which welcome page displays the headers of the http request that it receives.

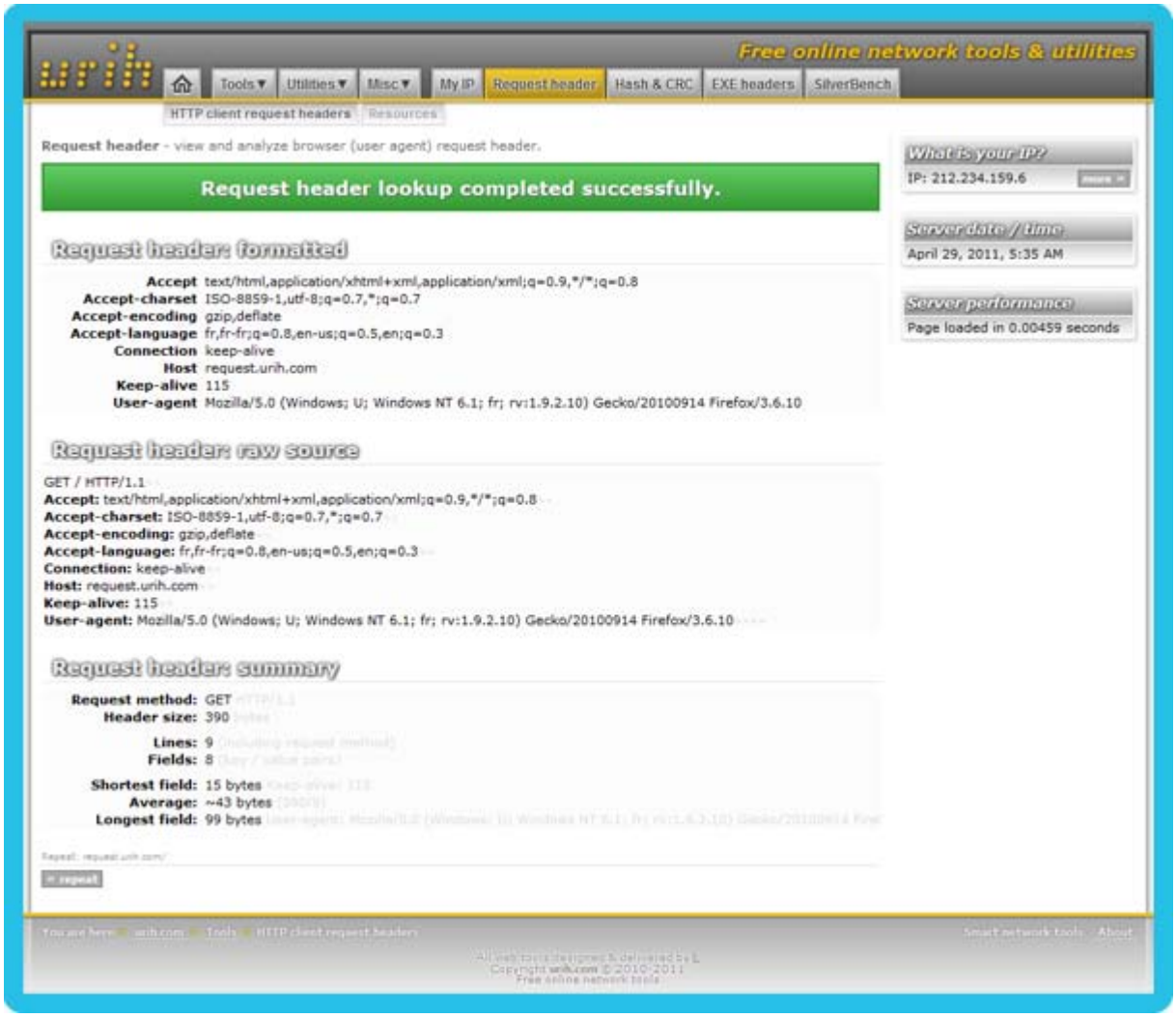


Figure 2 - 622: Modify request header extraction rule - request.urih.com website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically modify a header from the request to HTML resources that are accessed through Convertigo.

A transaction, named `Uri_headers_transaction`, is defined as default transaction for the *Site Clipper connector* named `RequestHeadersConnector`. It defines the URL `http://request.urih.com` as target URL to connect to the previously described web page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A screen class, named `HTML`, is defined thanks to a response *MIME type* criterion to handle HTML resources.

Finally, a screen class, named `hasAcceptEncoding`, is defined as inherited from previous one, using a *Request header* criterion parametered to detect `accept-encoding` header, regardless of its value. On this screen class, an *Add request header* extraction rule is added in order to add a new request header, named `hasAcceptEncoding`, to be sure the screen class has been detected.

Before adding the new extraction rule, let's switch to a web browser displaying the test platform of this project (for example Firefox). Executing the `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main page.

We can see that the request HTTP header named `hasAcceptEncoding` is added to the request to the HTML page, meaning that the `hasAcceptEncoding` screen class has matched. The existing `accept-encoding` header is appearing with the "`gzip,deflate`" value:

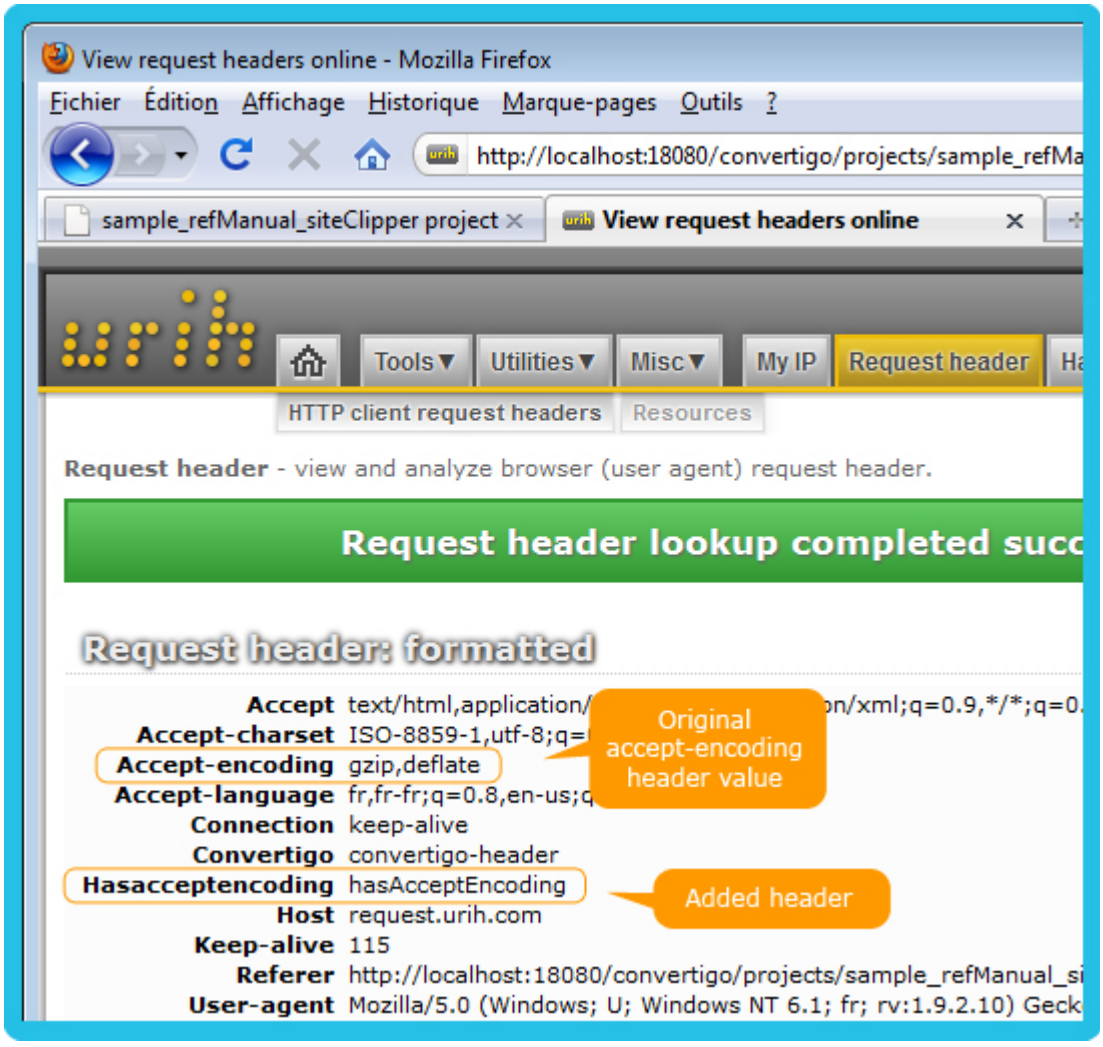


Figure 2 - 623: Modify request header extraction rule - Added header and original accept-encoding header on the request to the web page

On the `hasAcceptEncoding` screen class, a *Modify request header* extraction rule is added in order to modify the value of this already present `accept-encoding` request header.

The rule is created with the following parameters:

```
Modify request header [
   header name=accept-encoding
   header value=convertigo-encoding
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

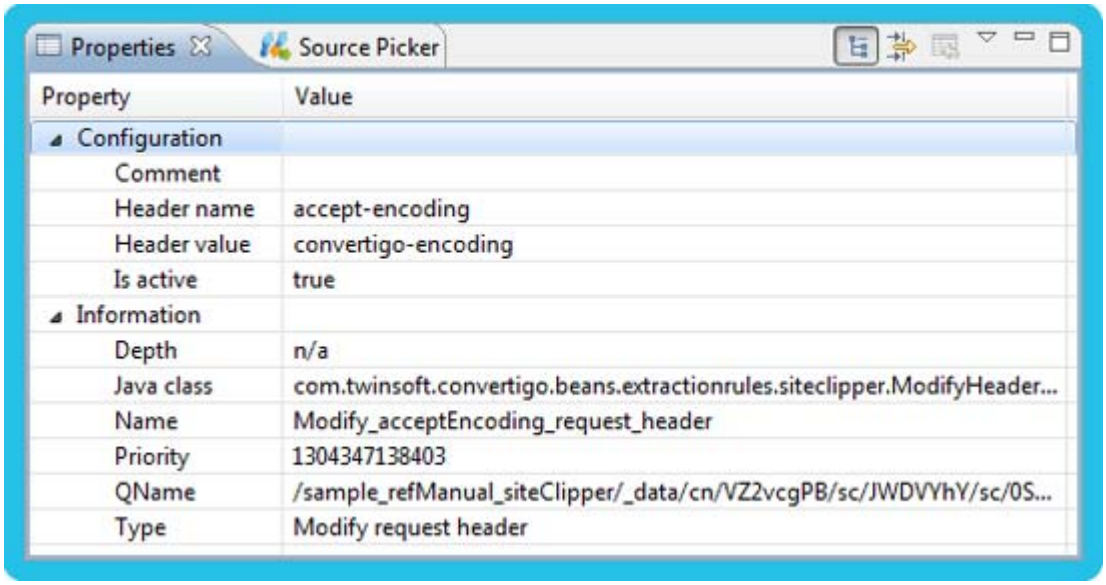


Figure 2 - 624: Modify request header extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

Figure 2 - 625: Modify request header extraction rule - Objects in Projects view

Switch back to the web browser displaying the test platform of this project. Executing the `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main page again.

The website displays the request HTTP headers that it receives. We can see that the request HTTP header named `hasAcceptEncoding` is still added to the request to the HTML page, meaning that the `hasAcceptEncoding` screen class has matched. The already existing `accept-encoding` header value is modified by the *Modify request header* extraction rule:

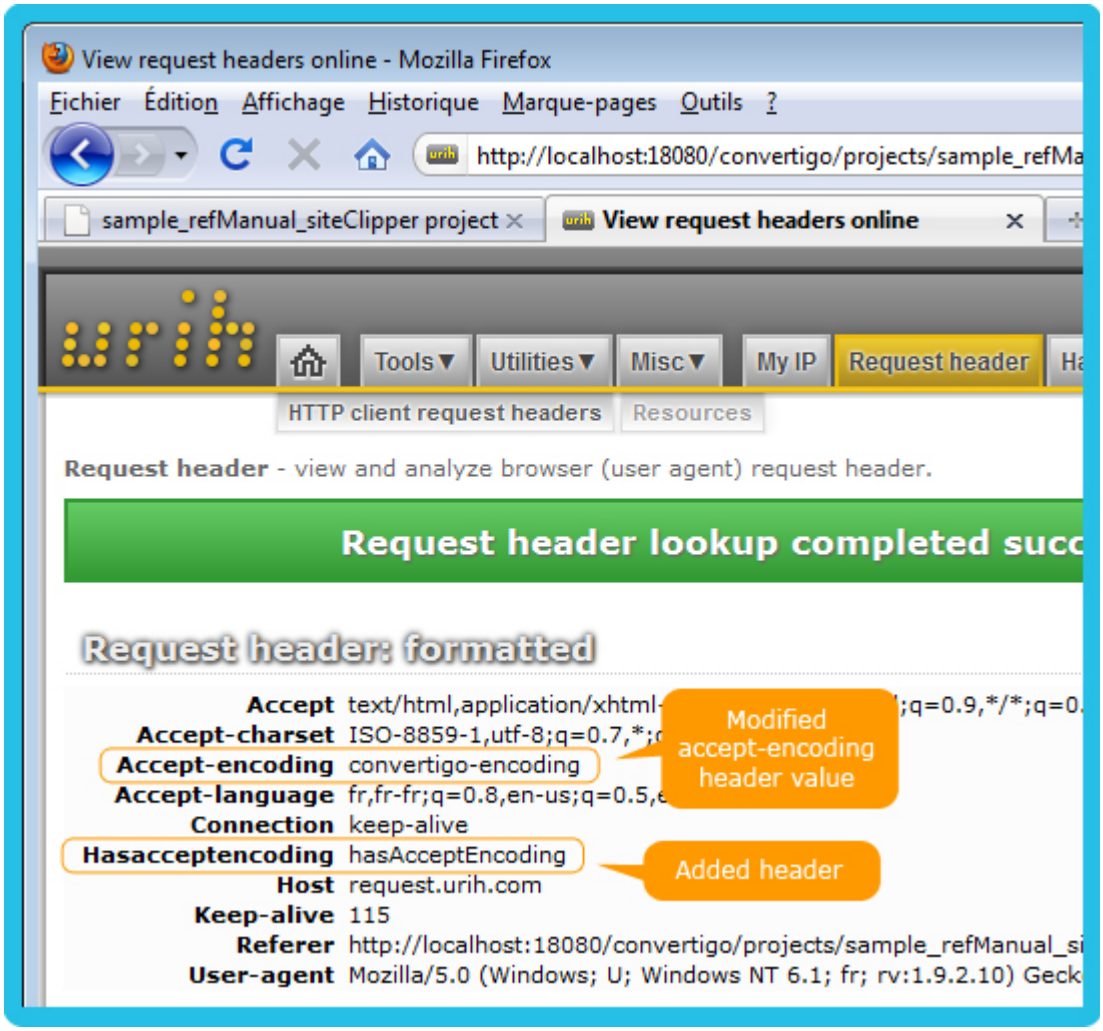


Figure 2 - 626: Modify request header extraction rule - Added and modified headers on the request to the web page

Example 2

The purpose of this second example is to show that a non-existing request header cannot be modified nor added by the *Modify request header* extraction rule.

Let's consider the same context and project as the first example.

On the `hasAcceptEncoding` screen class, a *Modify request header* extraction rule is added in order to modify a non-existing request header. It is created with the following parameters:

```
Modify request header [
   header name=undefined-header
   header value=convertigo
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

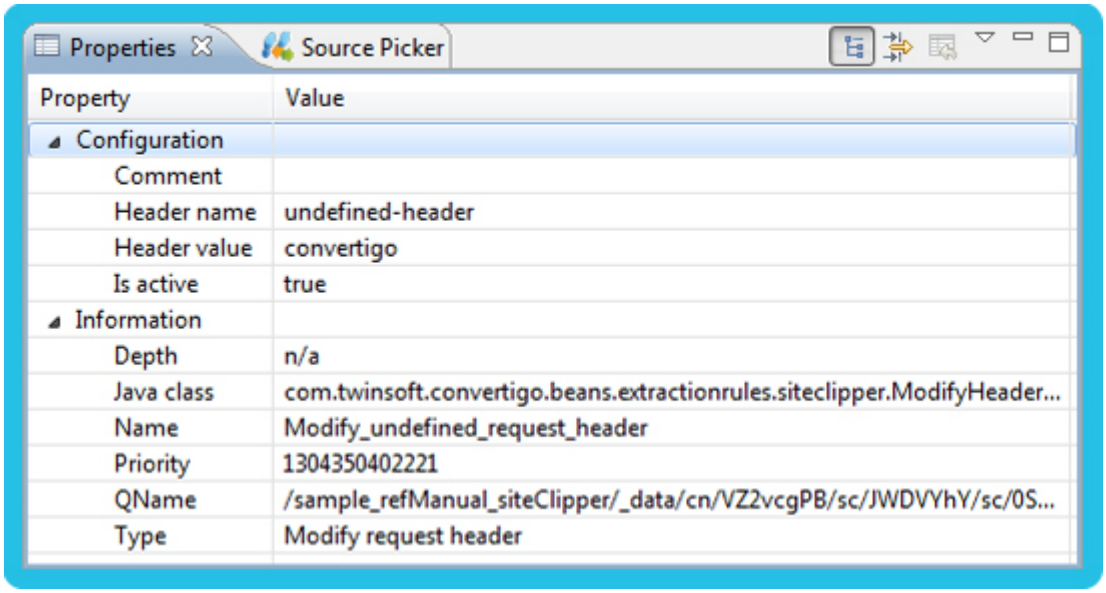


Figure 2 - 627: Modify request header extraction rule - Configuration example

The extraction rule appears as follows in the **Projects** view:

Figure 2 - 628: Modify request header extraction rule - Object in Projects view

Switch to a web browser displaying the test platform of this project (for example Firefox). Executing `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main page.

The website displays the request HTTP headers that it receives. We can see that the request HTTP header named `hasAcceptEncoding` is added to the request to the HTML page, meaning that the `hasAcceptEncoding` screen class has matched. The non-existing `undefined-header` header has not been modified nor added by the *Modify request header* extraction rule:

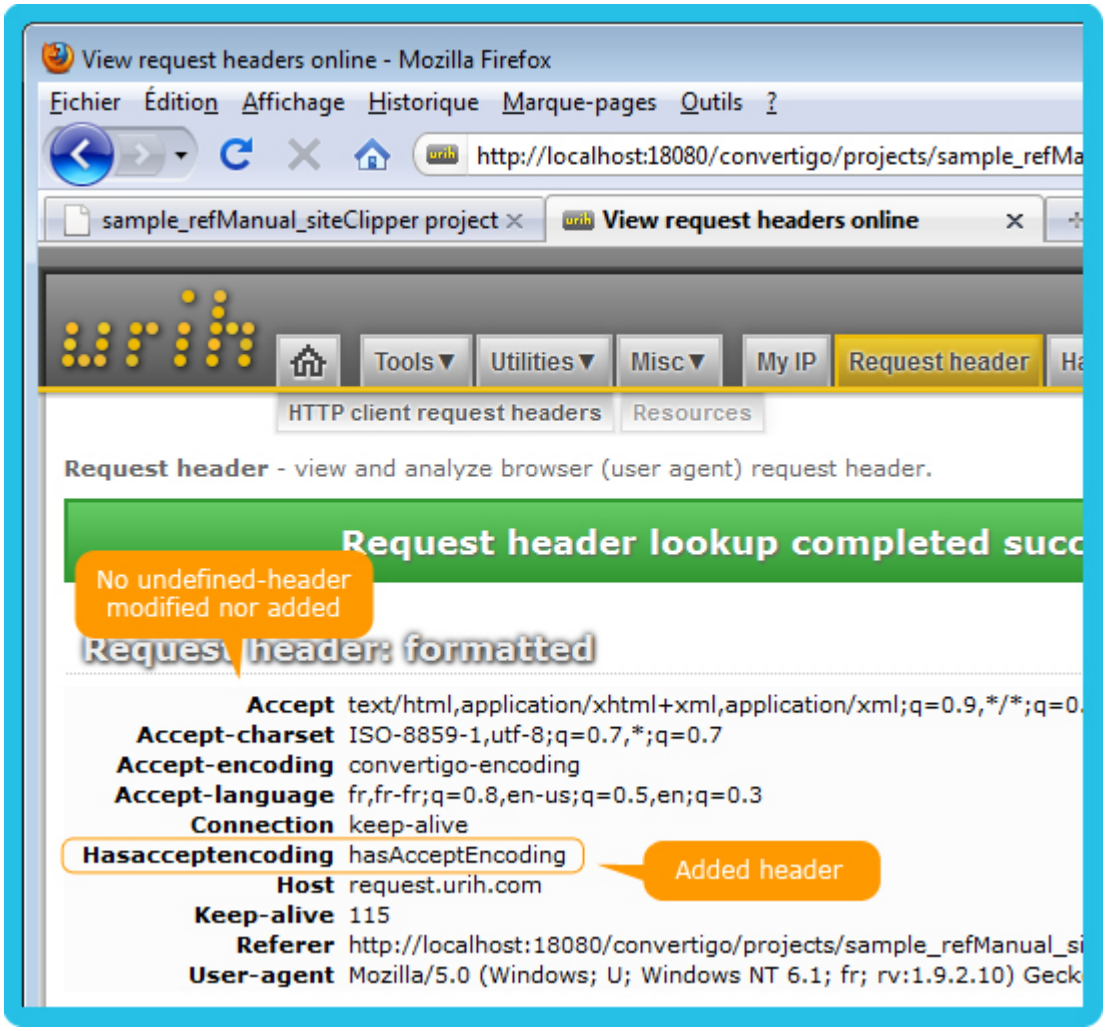


Figure 2 - 629: Modify request header extraction rule - Header added on Google HTML page

The *Modify request header* extraction rule doesn't change the value nor add non-existing headers. If you want to add a non-existing header to a request to a resource, use an *Add request header* extraction rule instead. For more information about this extraction rule, see "*Add request header*" extraction rule documentation and examples.

REMOVE REQUEST HEADER 

OBJECT DESCRIPTION

Removes an HTTP header from a Site Clipper request.

The *Remove request header* extraction rule removes an existing header from a request. The name of the header to remove is defined by the **Header name** property.

If the header defined in the **Header name** property doesn't exist in the request, the *Remove request header* extraction rule has no effect.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	configuration	Defines the header name. This property allows to define the name of the header to add to/modify in/remove from the request or the response which headers are manipulated.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let’s consider the website “request.urih.com” which welcome page displays the headers of the http request that it receives.

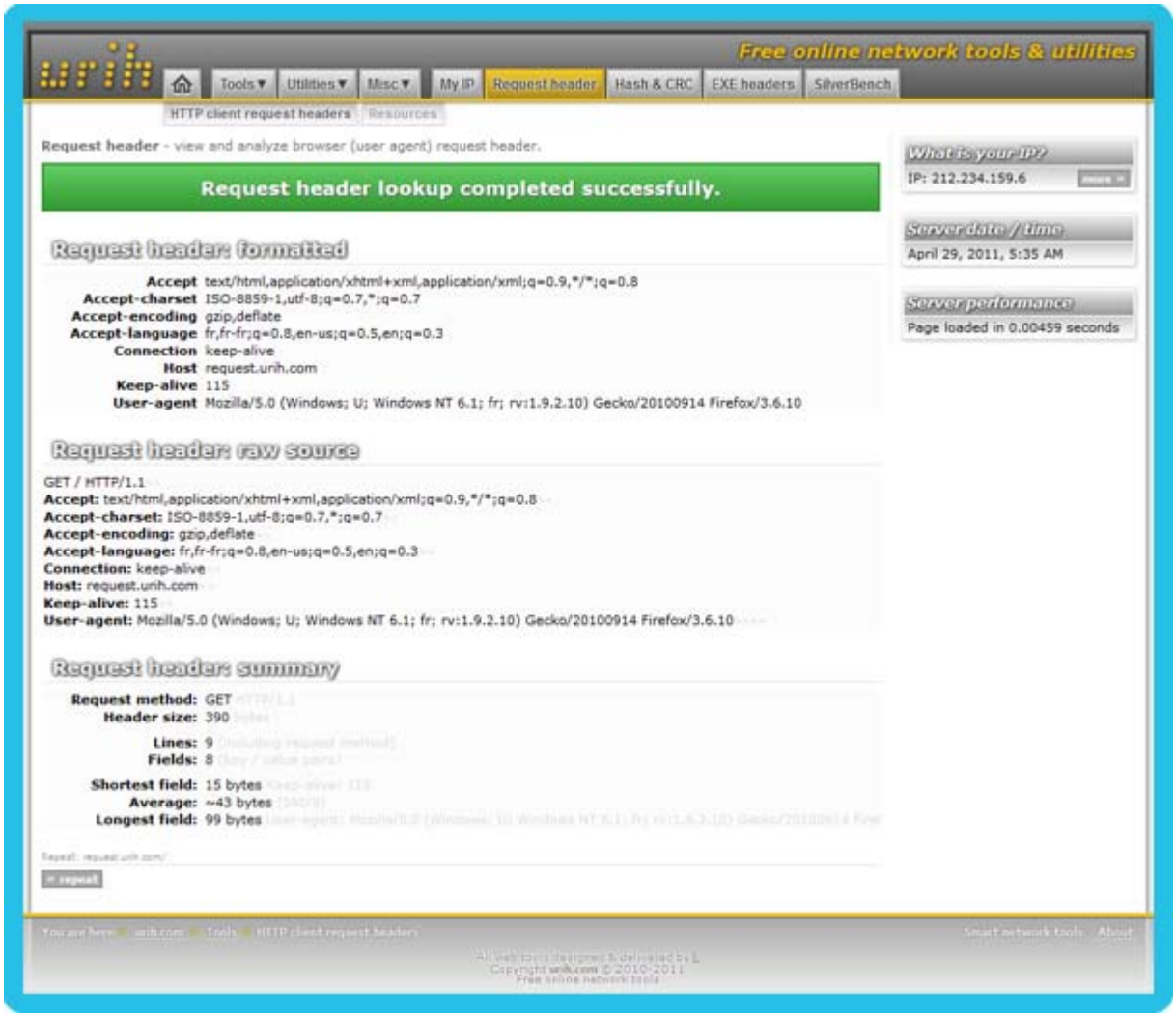


Figure 2 - 630: Remove request header extraction rule - request.urih.com website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically remove a request header from the request to HTML resources that are accessed through Convertigo.

A transaction, named `Uri_headers_transaction`, is defined as default transaction for the *Site Clipper connector* named `RequestHeadersConnector`. It defines the URL `http://request.urih.com` as target URL to connect to the previously described web page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A screen class, named `HTML`, is defined thanks to a response *MIME type* criterion to handle HTML resources.

Finally, a screen class, named `hasAcceptEncoding`, is defined as inherited from previous one, using a *Request header* criterion parametered to detect `accept-encoding` header, regardless of its value. On this screen class, an *Add request header* extraction rule is added in order to add a new request header, named `hasAcceptEncoding`, to be sure the screen class has been detected.

Before adding the new extraction rule, let's switch to a web browser displaying the test platform of this project (for example Firefox). Executing the `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main

page.

We can see that the request HTTP header named `hasAcceptEncoding` is added to the request to the HTML page, meaning that the `hasAcceptEncoding` screen class has matched. The existing `accept-encoding` header is appearing with the "`gzip,deflate`" value:

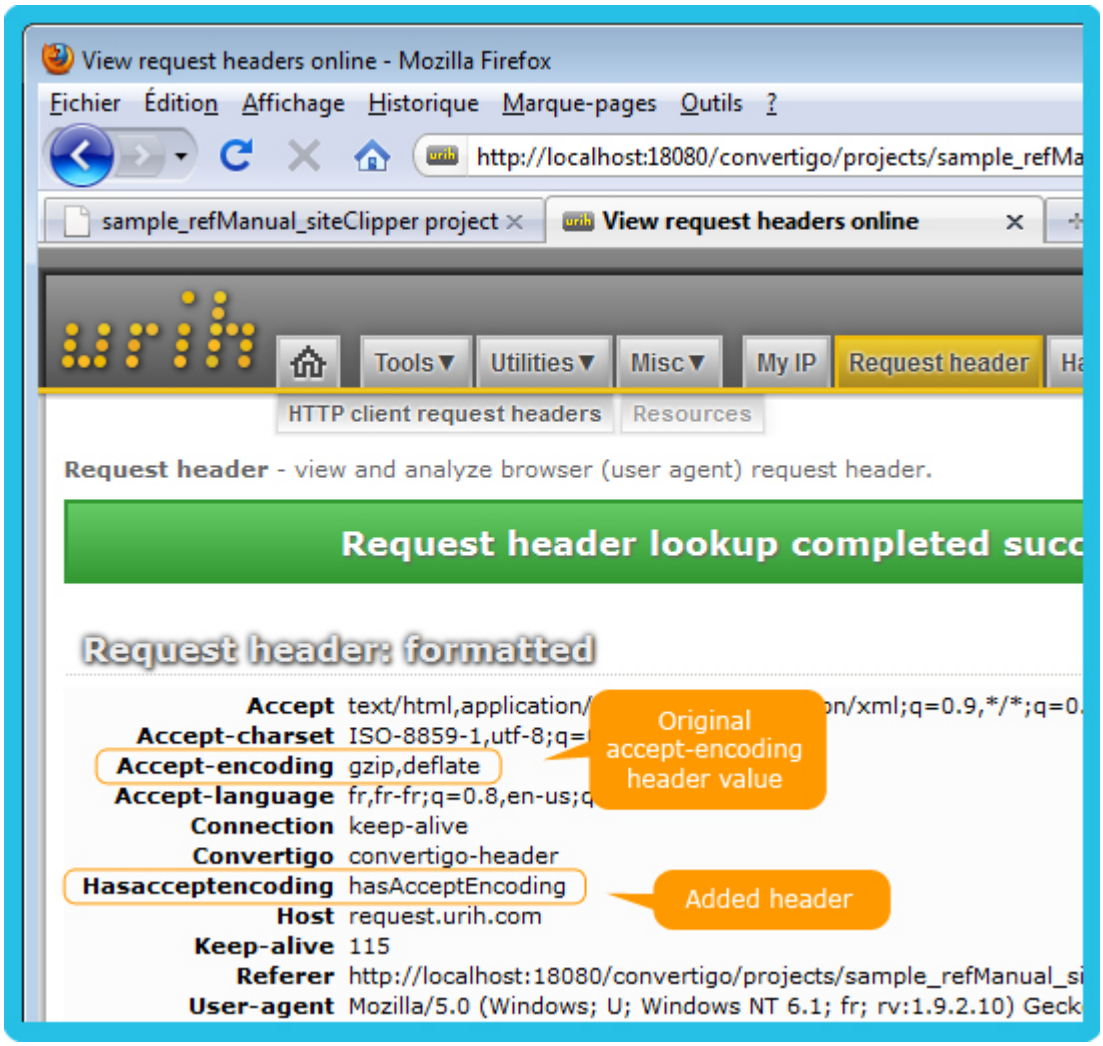


Figure 2 - 631: Remove request header extraction rule - Added header and original accept-encoding header on the request to the web page

On the `hasAcceptEncoding` screen class, a *Remove request header* extraction rule is added in order to remove the `accept-encoding` request header.

The rule is created with the following parameters:

```
Remove request header [
   header name=accept-encoding
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

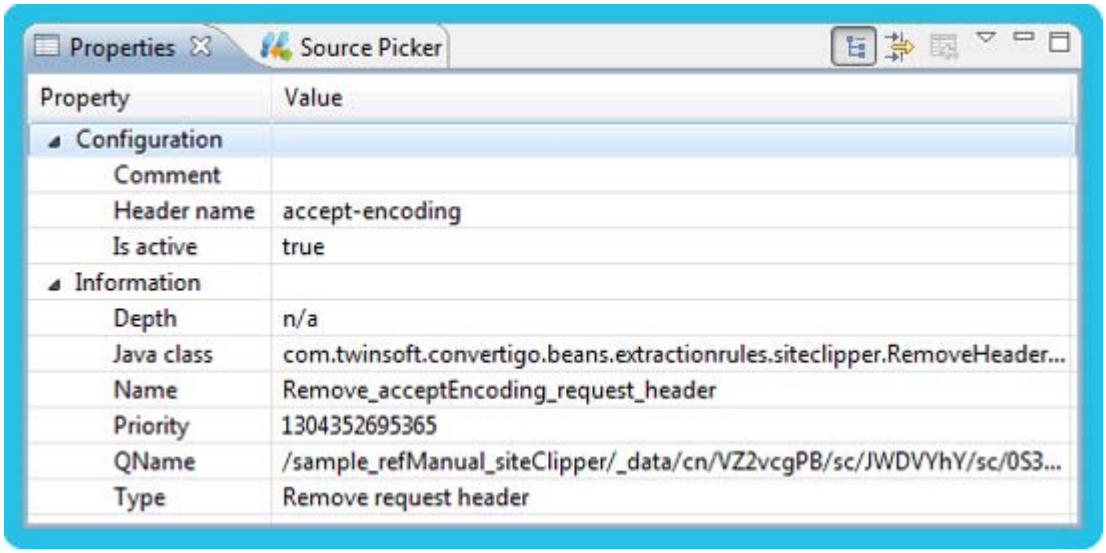


Figure 2 - 632: Remove request header extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

Figure 2 - 633: Remove request header extraction rule - Objects in Projects view

Switch back to the web browser displaying the test platform of this project. Executing the `Uri_headers_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the website main page again.

The website displays the request HTTP headers that it receives. We can see that the request HTTP header named `hasAcceptEncoding` is still added to the request to the HTML page, meaning that the `hasAcceptEncoding` screen class has matched. The `accept-encoding` header is not appearing anymore, it has been removed by the *Remove request header*

extraction rule:

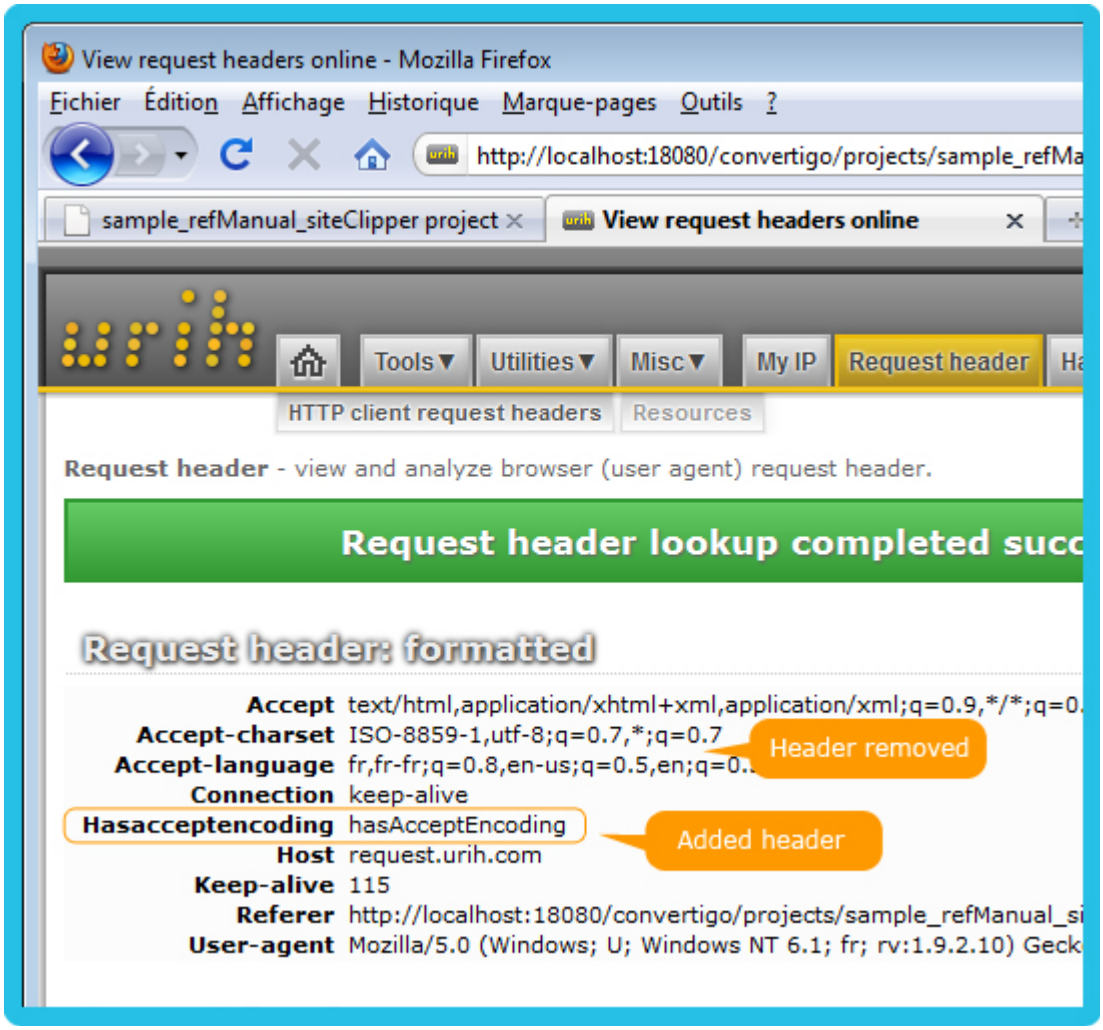


Figure 2 - 634: Remove request header extraction rule - Added and removed headers on the request to the web page

REMOVE REQUEST CACHE HEADERS 

OBJECT DESCRIPTION

Removes HTTP cache-related headers from a Site Clipper request.

The *Remove request cache headers* extraction rule removes all existing cache-related headers from a request. The following headers are removed:

- `Cache-Control`,
- `If-Match`,
- `If-Modified-Since`,
- `If-Range`,
- `If-None-Match`,
- `If-Unmodified-Since`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

REQUEST JS 

OBJECT DESCRIPTION

Executes JavaScript code in a Site Clipper request.

The *Request JS* extraction rule executes JavaScript code defined in the **Expression** property in a Site Clipper request.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	configuration	Defines the JavaScript expression to execute. This property is a JavaScript expression that is executed in the Site Clipper scope.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Response rules

RESPONSE JS 

OBJECT DESCRIPTION

Executes JavaScript code in a Site Clipper response.

The *Response JS* extraction rule executes JavaScript code defined in the **Expression** property in a Site Clipper response.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Expression	JS expression	configuration	Defines the JavaScript expression to execute. This property is a JavaScript expression that is executed in the Site Clipper scope.
Is active	boolean	configuration	Defines whether the extraction rule is active.

ADD RESPONSE HEADER 

OBJECT DESCRIPTION

Adds an HTTP header to a Site Clipper response.

The *Add response header* extraction rule adds the header defined in **Header name** property to the response's headers.

The value set in this header is defined by the **Header value** property.

If the header defined in the **Header name** property exists in the response, the *Add response header* extraction rule has no effect.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	configuration	Defines the header name. This property allows to define the name of the header to add to/modify in/remove from the request or the response which headers are manipulated.
Header value	String	configuration	Defines the header value. This property allows to define the value to set in the header defined by Header name property.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Example 1

Let’s consider the french version of the Google website.

Figure 2 - 635: Add response header extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically add a response header on HTML resources that are accessed through Convertigo.

A transaction, named `Google_fr_transaction`, is defined as default transaction for the *Site Clipper connector* named `GoogleResponseHeaders`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources.

A screen class, named `Google_Html_Page`, is defined thanks to a response *MIME type* criterion to handle HTML resources. On this screen class, an *Add response header* extraction rule is added in order to add a new response header.

The rule is created with the following parameters:

```
Add response header [
  header name=convertigo
  header value=googleHTMLpage
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

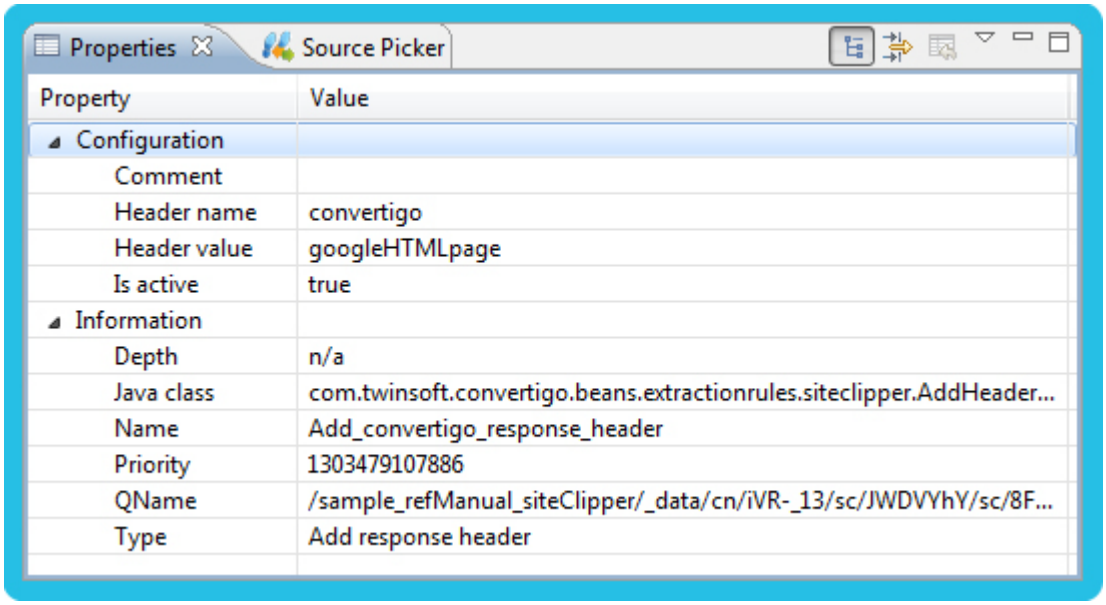


Figure 2 - 636: Add response header extraction rule - Configuration example

The extraction rule appears as follows in the **Projects** view:

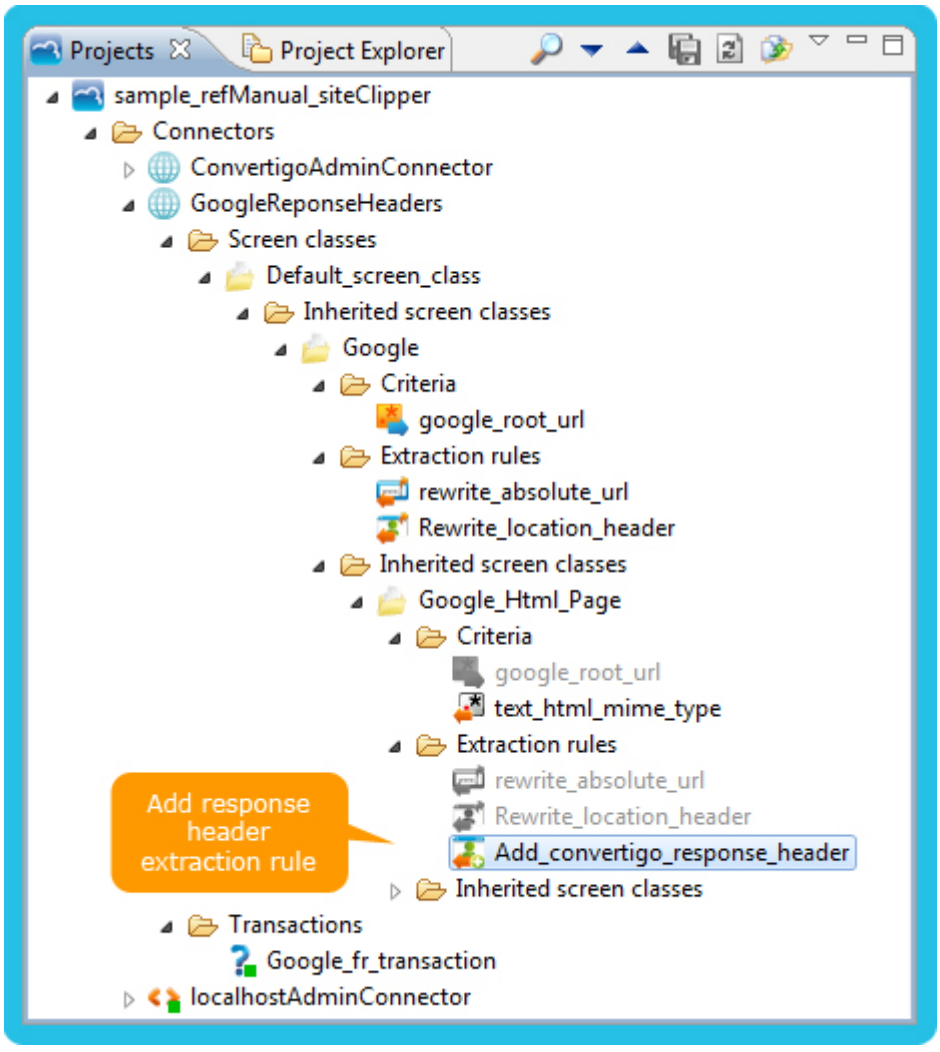


Figure 2 - 637: Add response header extraction rule - Object in Projects view

Switch to a Firefox browser displaying the test platform of this project, activate Firebug extension. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page.

In Firebug, we can see that the response HTTP header named `convertigo` and containing the defined value `googleHTMLpage` was added to the HTML page resource:

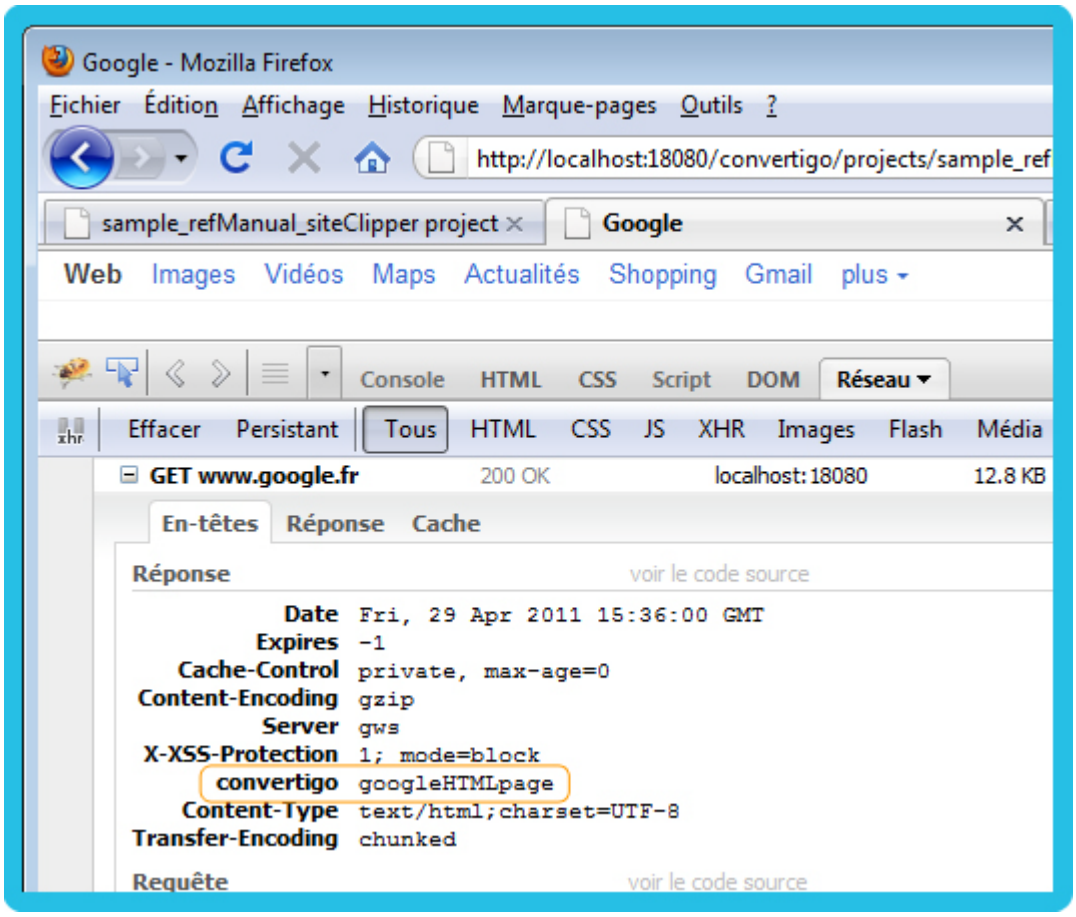


Figure 2 - 638: Add response header extraction rule - Header added on Google HTML page

Example 2

The purpose of this second example is to show that an already existing response header cannot be added again nor changed the value by the *Add response header* extraction rule.

Let's consider the same context and project as the first example.

In Firebug, we can see the existing `content-type` header appearing with the "`text/html;charset=UTF-8`" value received when accessing the HTML page resource:

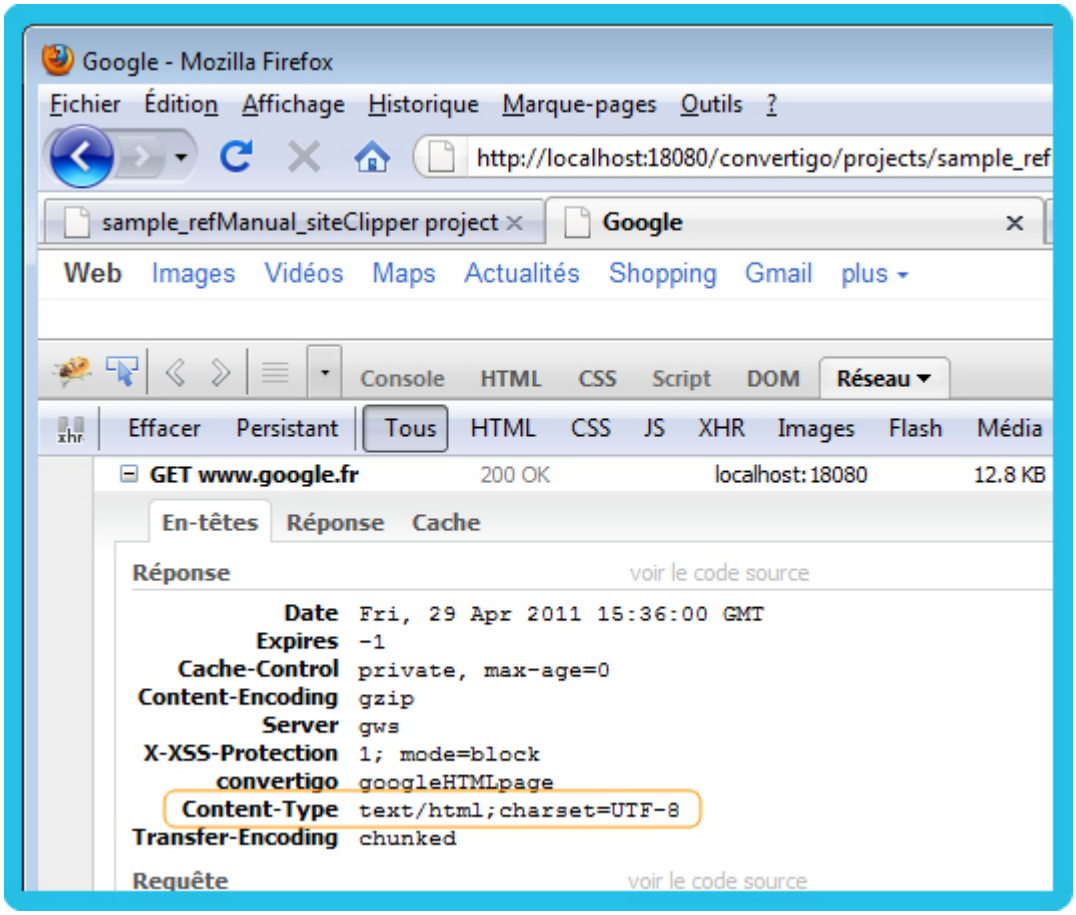


Figure 2 - 639: Add response header extraction rule - content-type header existing on Google HTML page

A screen class, named `hasContentType`, is defined as inherited from previous `Google_Html_Page` screen class. It uses a *Response header* criterion parametered to detect `content-type` header, regardless of its value.

On this screen class, an *Add response header* extraction rule is added in order to add this `content-type` response header again. It is created with the following parameters:

```
Add response header [
  header name=content-type
  header value=convertigo
]
```

A second *Add response header* extraction rule is added in order to add a new response header to be sure the screen class has been detected. It is created with the following parameters:

```
Add response header [
  header name=hasContentType
  header value=hasContentType
]
```

For both extraction rules, the parameters are edited in the **Properties** view of the Convertigo Studio:

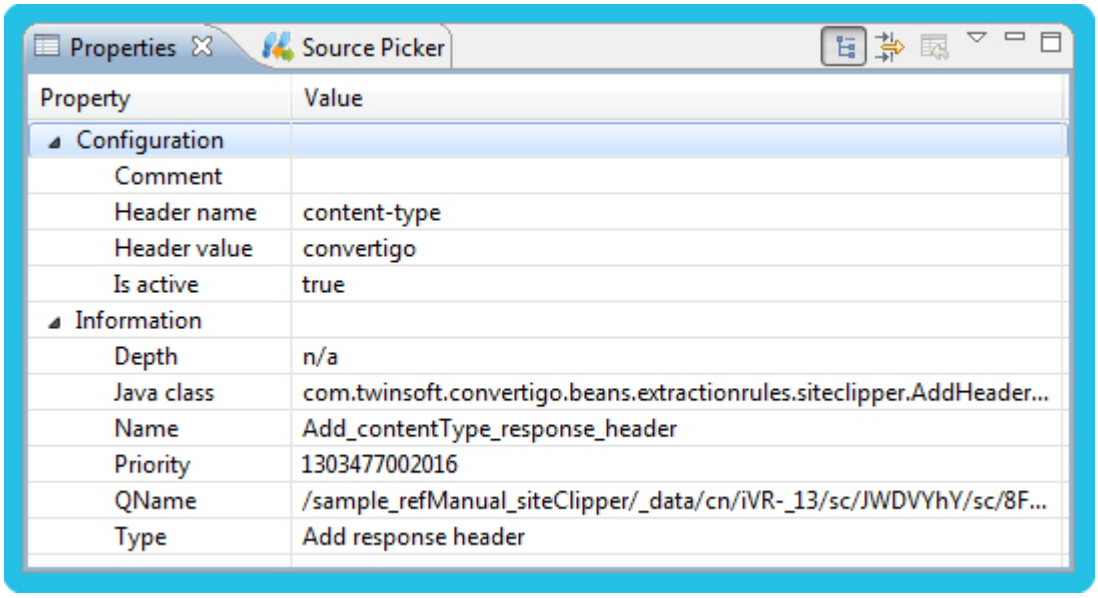


Figure 2 - 640: Add response header extraction rule - Configuration example

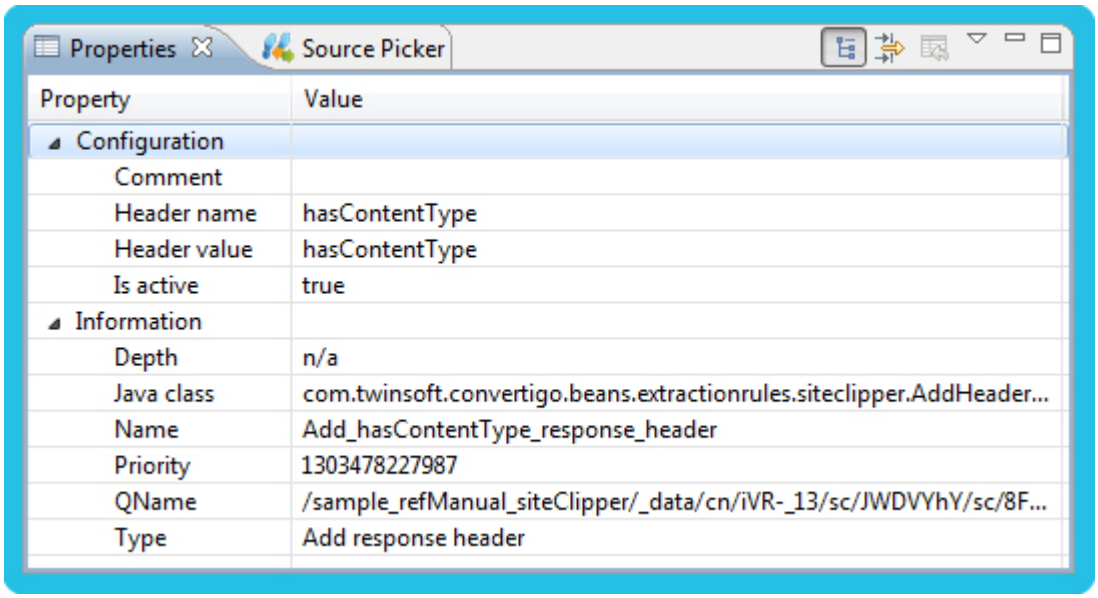


Figure 2 - 641: Add response header extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

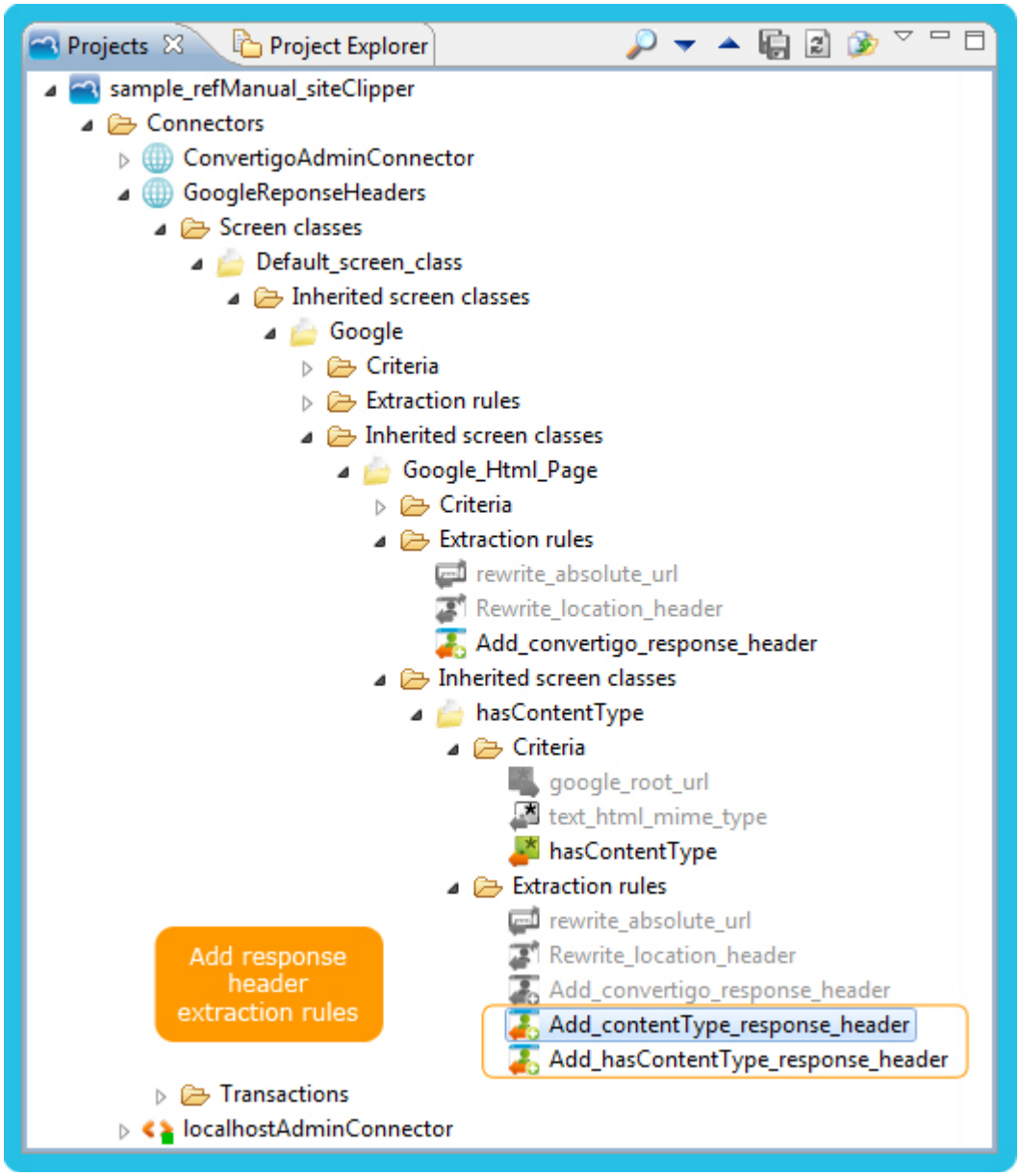


Figure 2 - 642: Add response header extraction rule - Objects in Projects view

Switch to a Firefox browser displaying the test platform of this project, activate Firebug extension. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page.

In Firebug, we can see that the response HTTP header named `hasContentType` is added to the HTML page resource, meaning that the `hasContentType` screen class has matched. The already existing `content-type` header is not added again by the rule and is left unchanged:

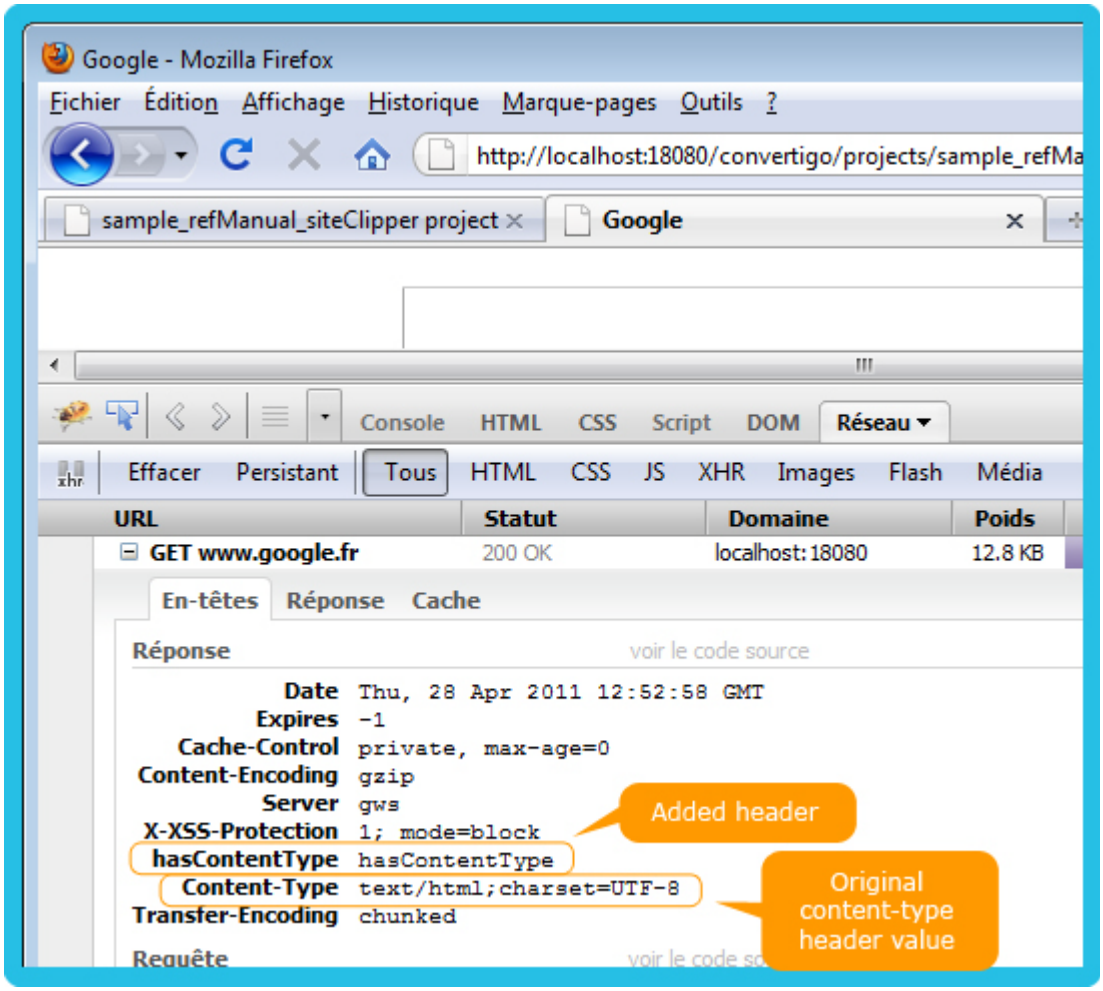


Figure 2 - 643: Add response header extraction rule - Header added and header not modified on Google HTML page

The *Add response header* extraction rule doesn't change the value of existing headers nor add a second header with the same name. If you want to modify an existing header, use a *Modify response header* extraction rule instead. For more information about this extraction rule, see "*Modify response header*" extraction rule documentation and examples.

MODIFY RESPONSE HEADER 

OBJECT DESCRIPTION

Modifies an HTTP header in a Site Clipper response.

The *Modify response header* extraction rule modifies an existing header from a response. The name of the header to modify is defined by the **Header name** property.

The new value to set in this header is defined by the **Header value** property.

If the header defined in the **Header name** property doesn't exist in the response, the *Modify response header* extraction rule has no effect.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	configuration	Defines the header name. This property allows to define the name of the header to add to/modify in/remove from the request or the response which headers are manipulated.
Header value	String	configuration	Defines the header value. This property allows to define the value to set in the header defined by Header name property.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Example 1

Let’s consider the french version of the Google website.

Figure 2 - 644: Modify response header extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically modify a response header on HTML resources that are accessed through Convertigo.

A transaction, named `Google_fr_transaction`, is defined as default transaction for the *Site Clipper connector* named `GoogleResponseHeaders`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A screen class, named `Google_Html_Page`, is defined thanks to a response *MIME type* criterion to handle HTML resources.

Finally, a screen class, named `hasContentType`, is defined as inherited from previous one, using a *Response header* criterion parametered to detect `content-type` header, regardless of its value. On this screen class, an *Add response header* extraction rule is added in order to add a new response header, named `hasContentType`, to be sure the screen class has been detected.

Before adding the new extraction rule, let's switch to a Firefox browser displaying the test platform of this project, with the Firebug extension activated. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page.

In Firebug, we can see that the response HTTP header named `hasContentType` is added to the HTML page resource, meaning that the `hasContentType` screen class has matched. The existing `content-type` header is appearing with the "`text/html;charset=UTF-8`" value:

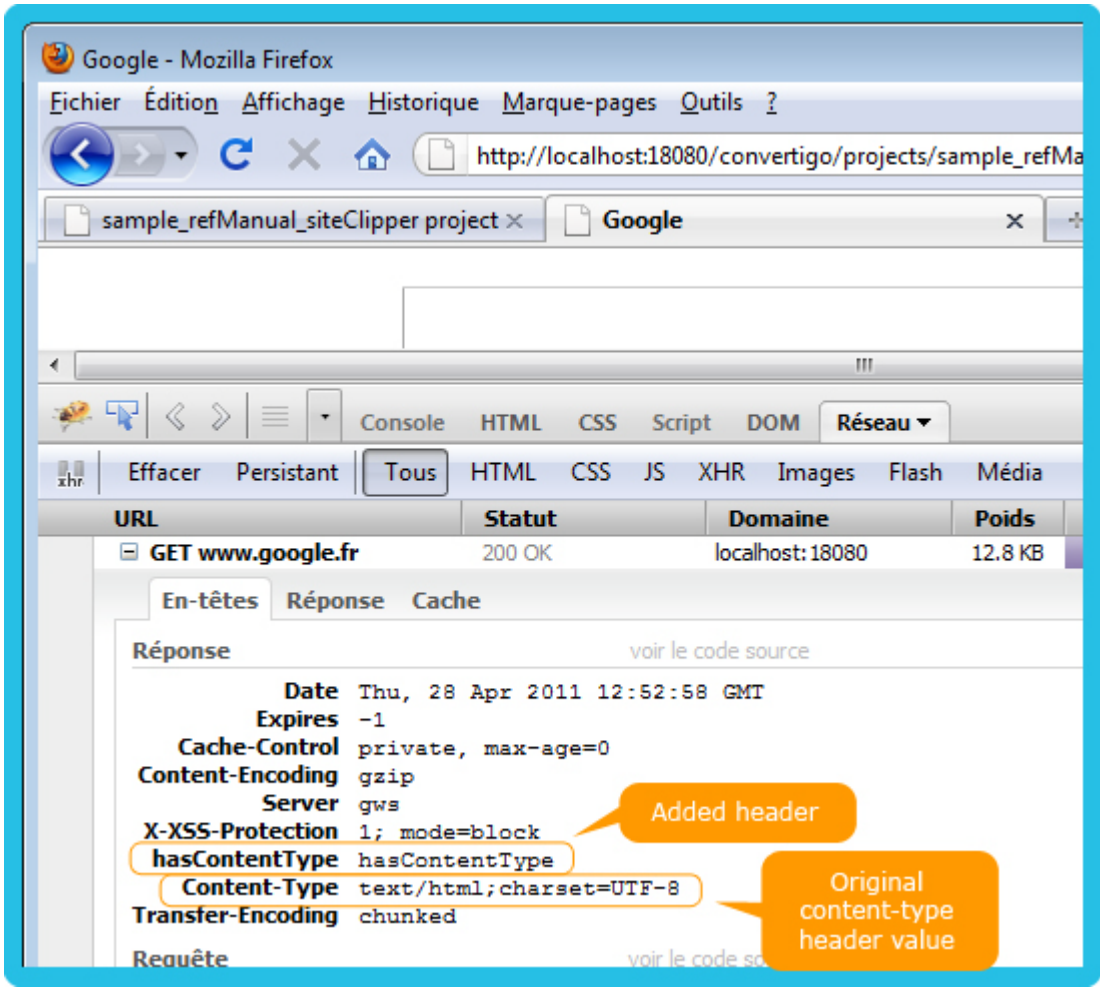


Figure 2 - 645: Modify response header extraction rule - Added header and original content-type header on Google HTML page

On the `hasContentType` screen class, a *Modify response header* extraction rule is added in order to modify the value of this already present `content-type` response header.

The rule is created with the following parameters:

```
Modify response header [
  header name=content-type
  header value=text/plain
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

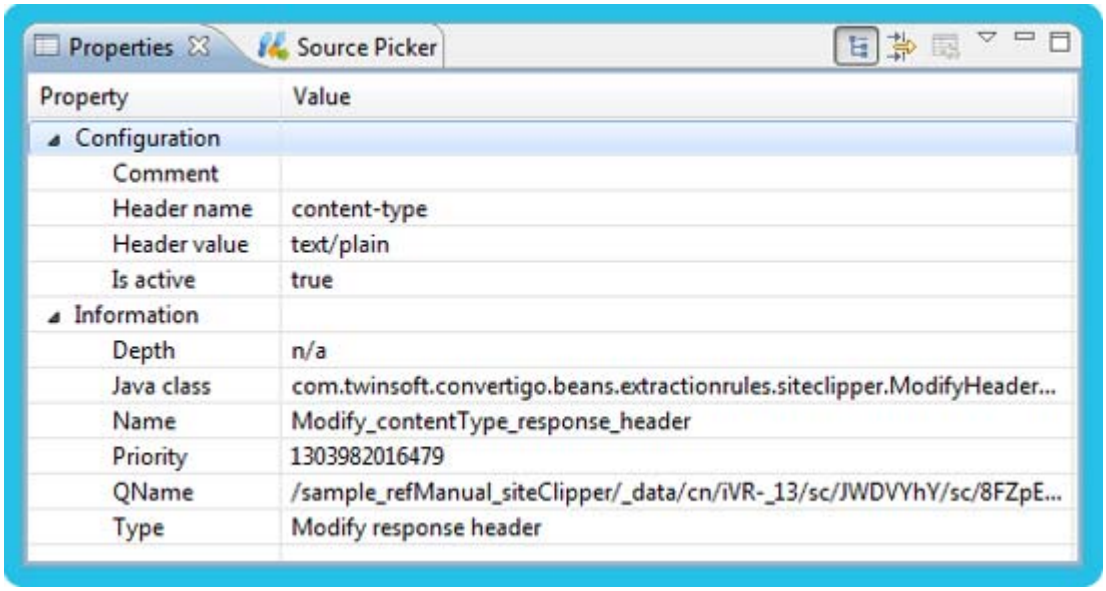


Figure 2 - 646: Modify response header extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

Figure 2 - 647: Modify response header extraction rule - Objects in Projects view

Switch back to Firefox browser displaying the test platform of this project, with Firebug extension activated. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page again.

In Firebug, we can see that the response HTTP header named `hasContentType` is still added to the HTML page resource, meaning that the `hasContentType` screen class has matched. The already existing `content-type` header value is modified by the *Modify response header* extraction rule:

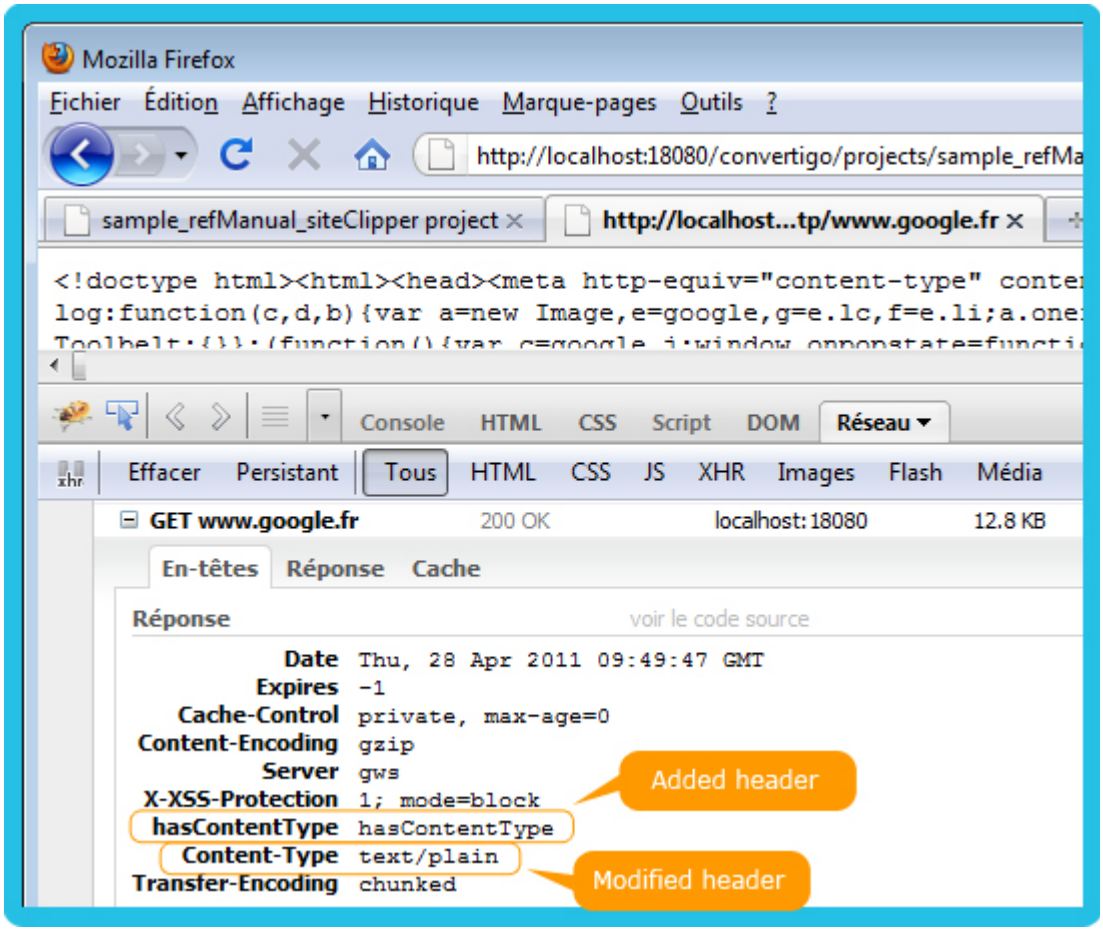


Figure 2 - 648: Modify response header extraction rule - Added and modified headers on Google HTML page

Example 2

The purpose of this second example is to show that a non-existing response header cannot be modified nor added by the *Modify response header* extraction rule.

Let's consider the same context and project as the first example.

On the `hasContentType` screen class, a *Modify response header* extraction rule is added in order to modify a non-existing response header. It is created with the following parameters:

```
Modify response header [
  header name=undefined-header
  header value=convertigo
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

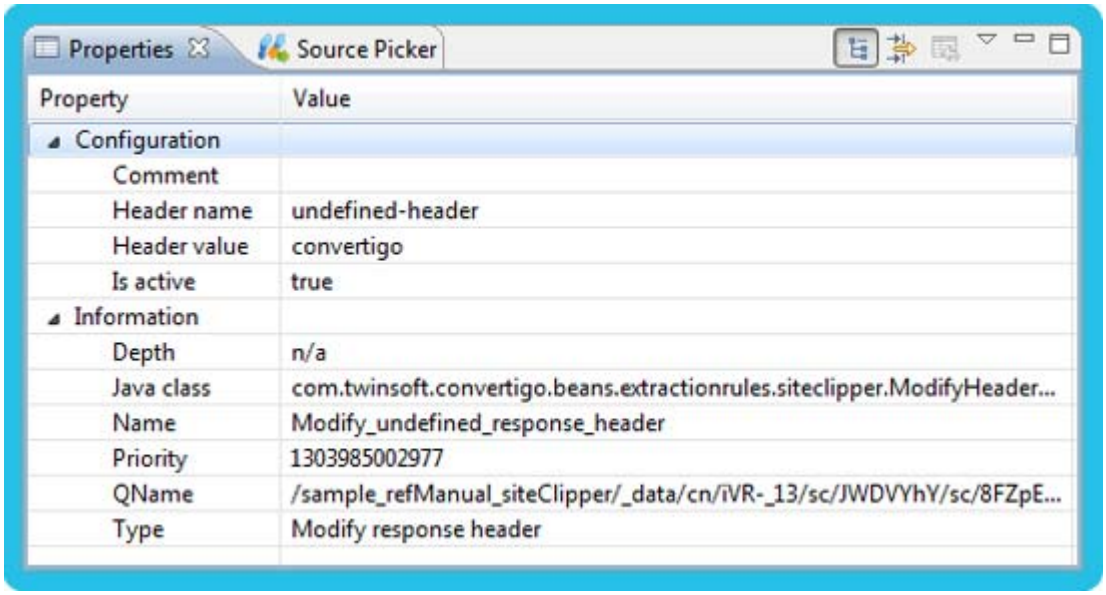


Figure 2 - 649: Modify response header extraction rule - Configuration example

The extraction rule appears as follows in the **Projects** view:

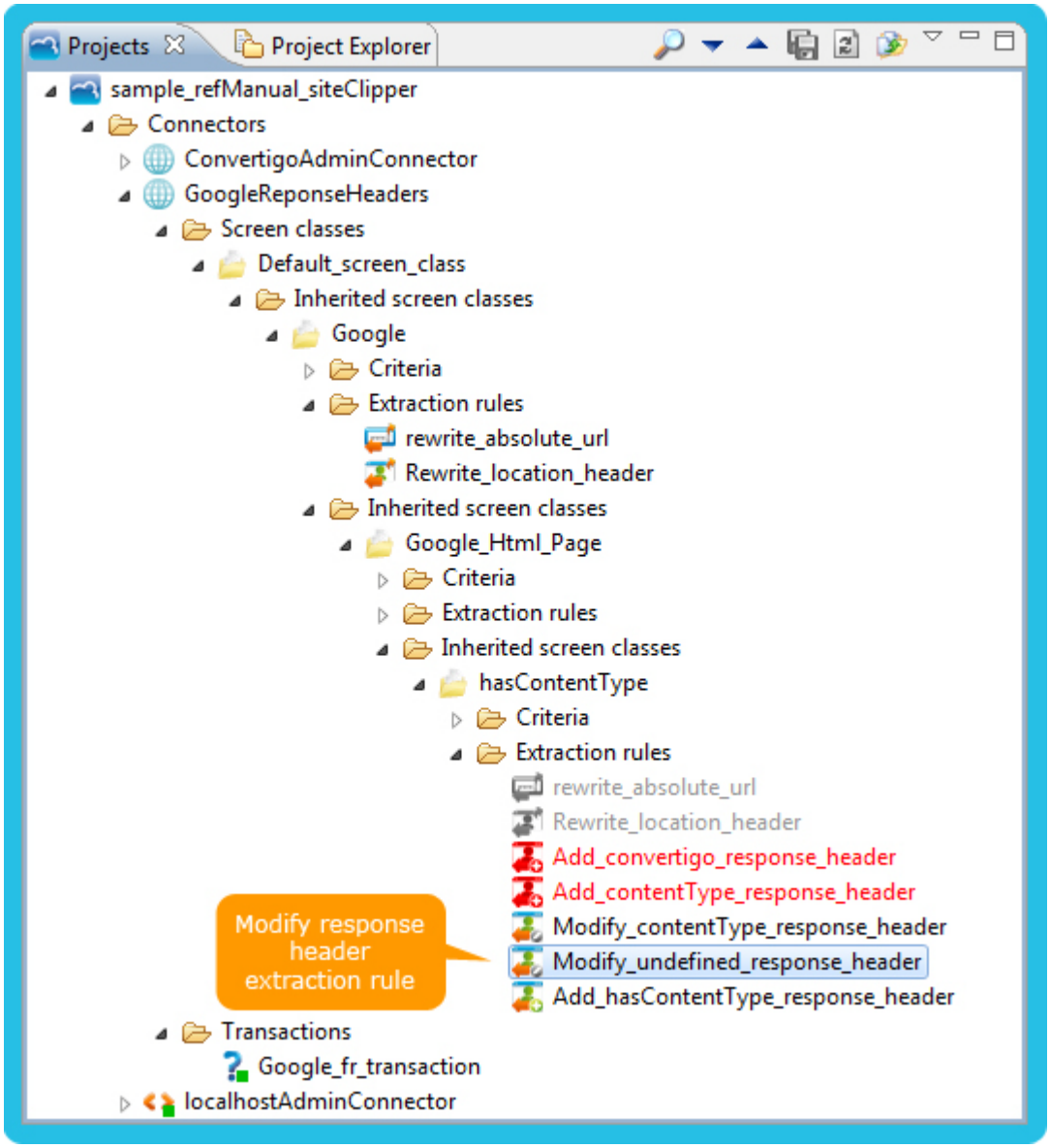


Figure 2 - 650: Modify response header extraction rule - Object in Projects view

Switch to a Firefox browser displaying the test platform of this project, activate Firebug extension. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page.

In Firebug, we can see that the response HTTP header named `hasContentType` is added to the HTML page resource, meaning that the `hasContentType` screen class has matched. The non-existing `undefined-header` header has not been modified nor added by the *Modify response header* extraction rule:

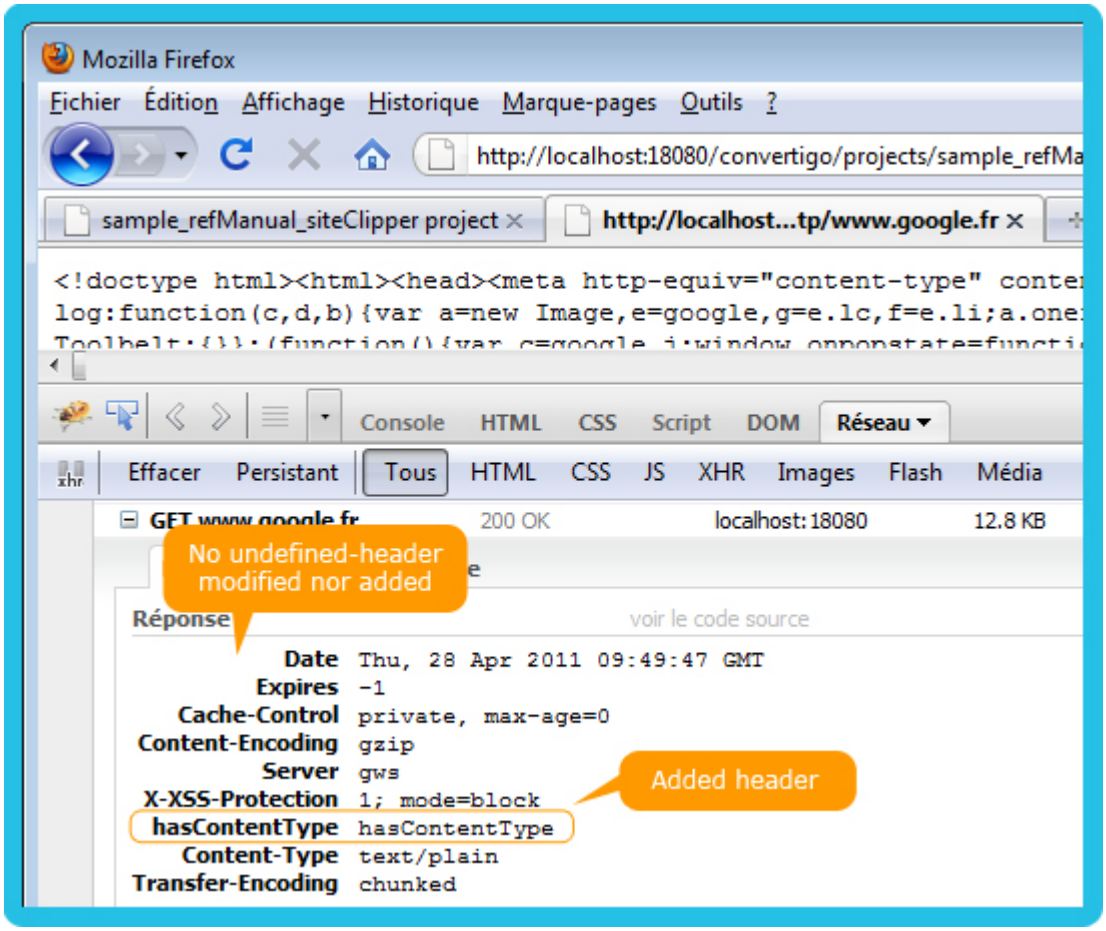


Figure 2 - 651: Modify response header extraction rule - Header added on Google HTML page

The *Modify response header* extraction rule doesn't change the value nor add non-existing headers. If you want to add a non-existing header to a resource, use an *Add response header* extraction rule instead. For more information about this extraction rule, see "*Add response header*" extraction rule documentation and examples.

REMOVE RESPONSE HEADER 

OBJECT DESCRIPTION

Removes an HTTP header from a Site Clipper response.

The *Remove response header* extraction rule removes an existing header from a response. The name of the header to remove is defined by the **Header name** property.

If the header defined in the **Header name** property doesn't exist in the response, the *Remove response header* extraction rule has no effect.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Header name	String	configuration	Defines the header name. This property allows to define the name of the header to add to/modify in/remove from the request or the response which headers are manipulated.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let's consider the french version of the Google website.

Figure 2 - 652: Remove response header extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically remove a response header on HTML resources that are accessed through Convertigo.

A transaction, named `Google_fr_transaction`, is defined as default transaction for the *Site Clipper connector* named `GoogleResponseHeaders`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A screen class, named `Google_Html_Page`, is defined thanks to a response *MIME type* criterion to handle HTML resources.

Finally, a screen class, named `hasContentType`, is defined as inherited from previous one, using a *Response header* criterion parametered to detect `content-type` header, regardless of its value. On this screen class, an *Add response header* extraction rule is added in order to add a new response header named `hasContentType` to be sure the screen class has been detected.

Before adding the new extraction rule, let's switch to a Firefox browser displaying the test platform of this project, with the Firebug extension activated. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page.

In Firebug, we can see that the response HTTP header named `hasContentType` is added to the HTML page resource, meaning that the `hasContentType` screen class has matched. The existing `content-type` header is appearing with the "`text/html;charset=UTF-8`" value:

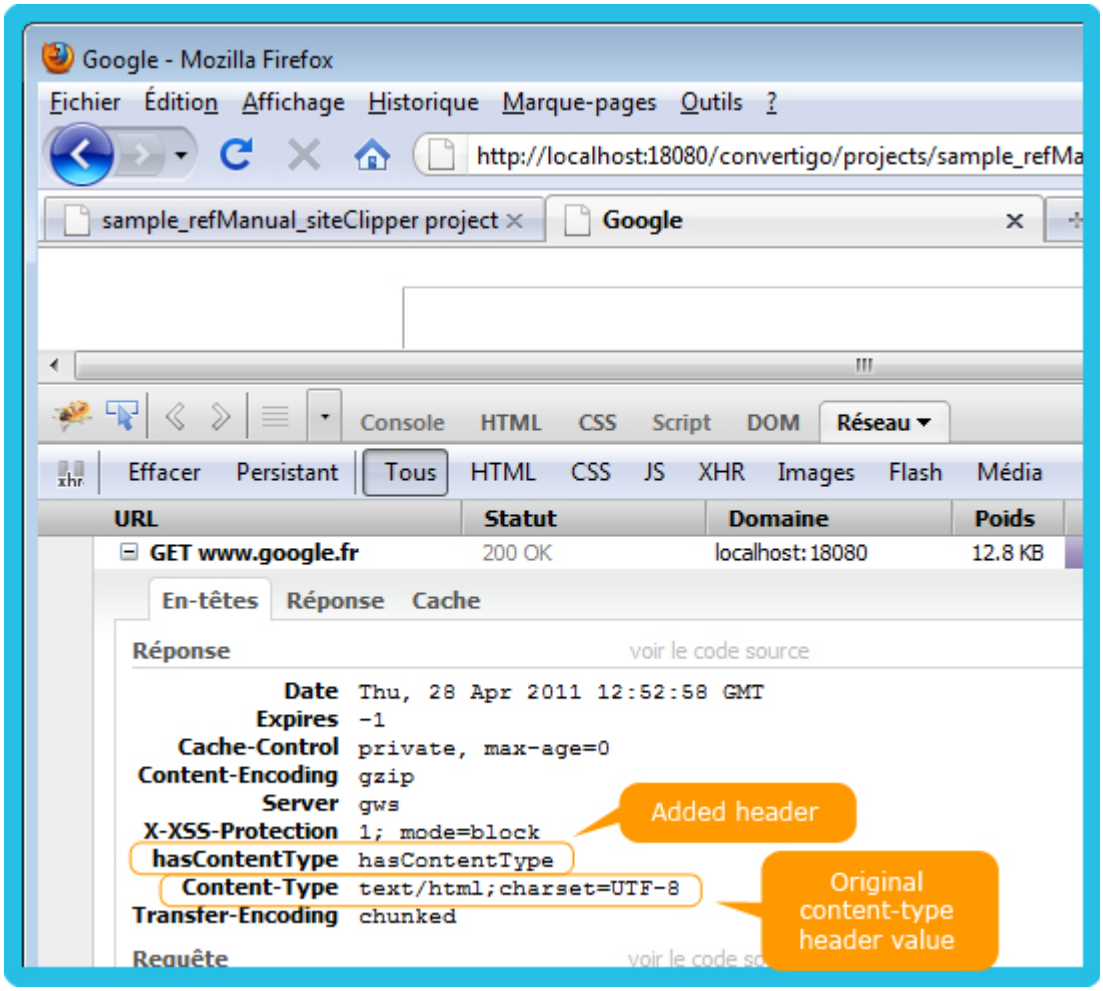


Figure 2 - 653: Remove response header extraction rule - Added header and original content-type header on Google HTML page

On the `hasContentType` screen class, a *Remove response header* extraction rule is added in order to remove the `content-type` response header.

The rule is created with the following parameters:

```
Remove response header [
  header name=content-type
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

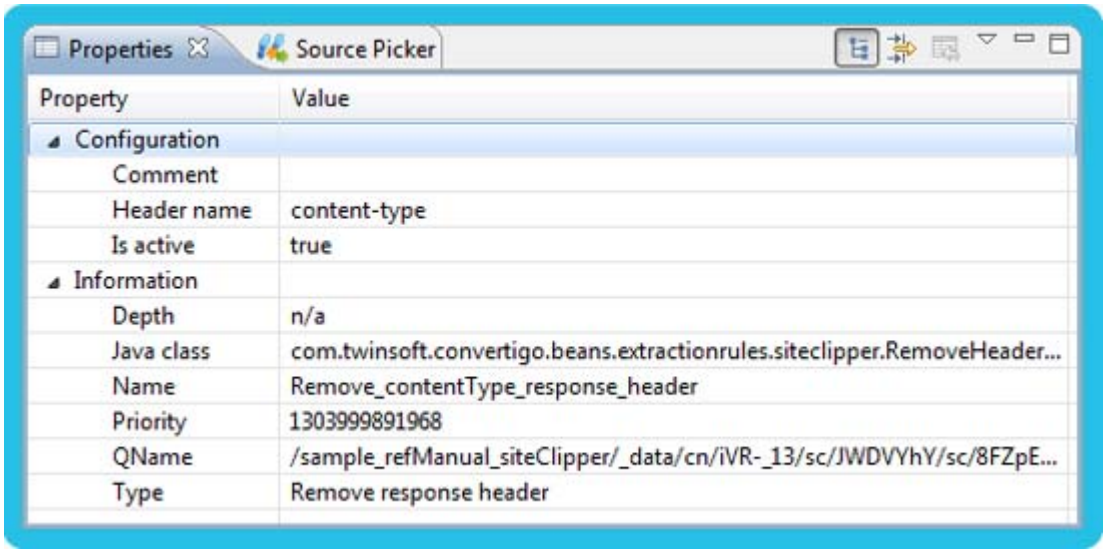


Figure 2 - 654: Remove response header extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

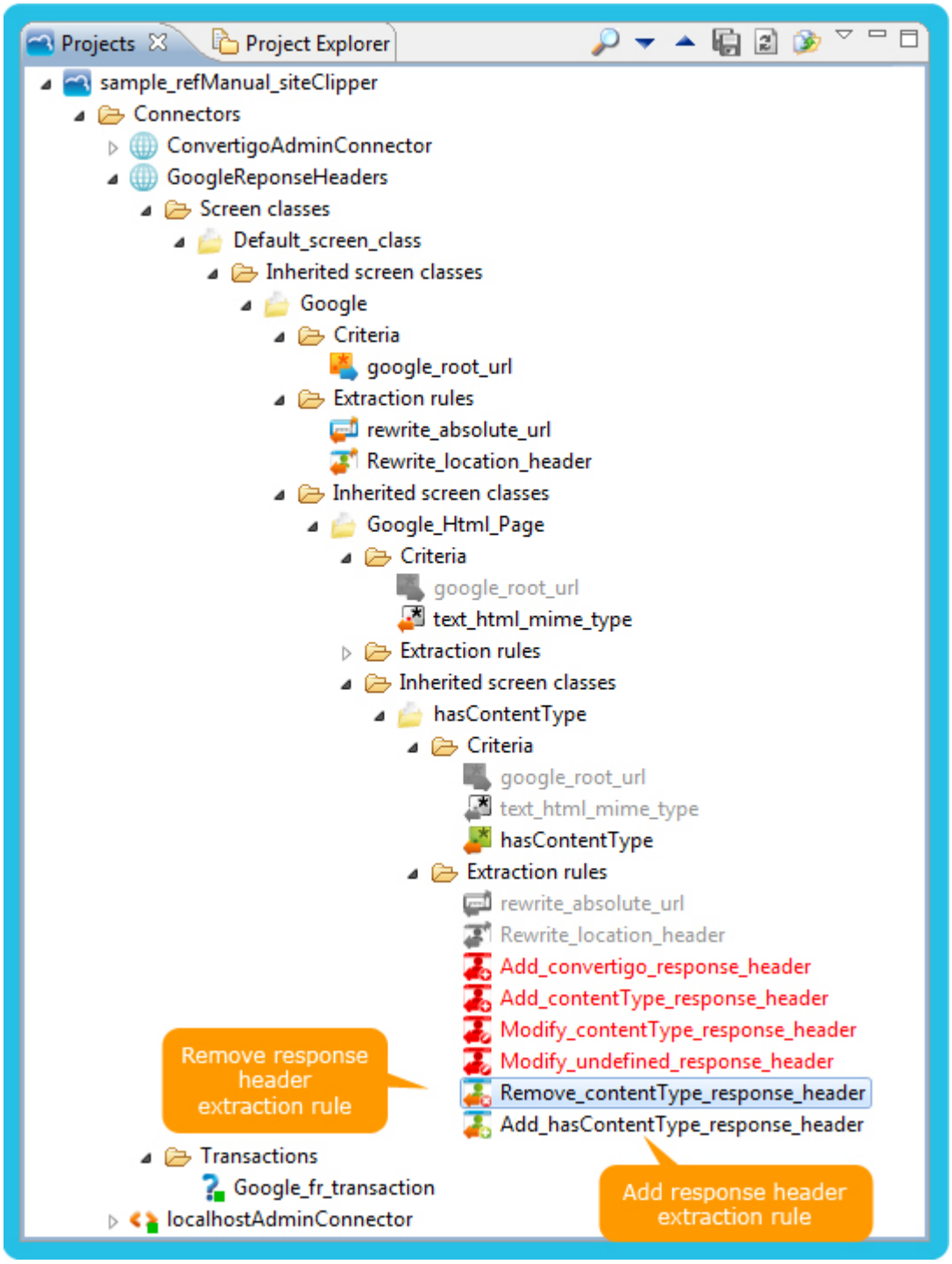


Figure 2 - 655: Remove response header extraction rule - Objects in Projects view

Switch back to Firefox browser displaying the test platform of this project, with Firebug extension activated. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page again.

In Firebug, we can see that the response HTTP header named `hasContentType` is still added to the HTML page resource, meaning that the `hasContentType` screen class has matched. The `content-type` header is not appearing anymore, it has been removed by the *Remove response header* extraction rule:

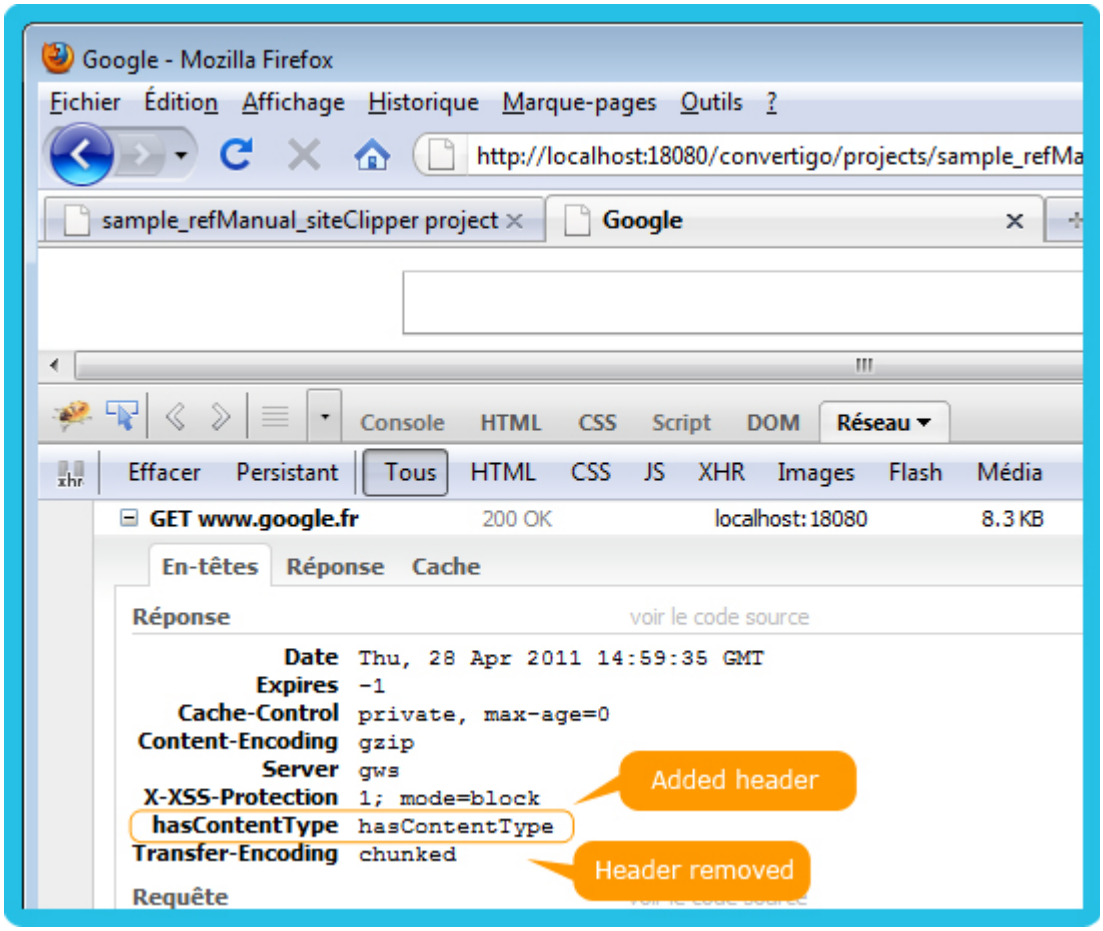


Figure 2 - 656: Remove response header extraction rule - Added and removed headers on Google HTML page

REPLACE STRING 

OBJECT DESCRIPTION

Makes string replacements in a Site Clipper response.

The *Replace string* extraction rule replaces, in a Site Clipper response, string occurrences matching the regular expression defined in **Regular expression** property.

The corresponding strings from the response are replaced by the value defined in the **Replacement** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Regular expression	String	configuration	Defines the regular expression defining text to be replaced. This property allows defining a regular expression as a string pattern to find in the Site Clipper response. Notes: • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
Replacement	String	configuration	Defines the replacement string to apply. This property defines the string that will be set in the place of each string occurrence matching the regular expression in the Site Clipper response. Note: • This property can refer to **groups**, which is a standard notion of regular expressions. For more information about regular expressions groups and symbols to use in replacement strings, see the following pages: `http://www.regular-expressions.info/refadv.html` and `http://www.regular-expressions.info/refreplace.html`. • This property can use available variables automatically containing Convertigo server paths. For more information about Convertigo paths variables, see Appendix "*Convertigo paths variables - Usable symbols*".

EXAMPLES

Example 1

Let's consider the french version of the Google website.

Figure 2 - 657: Replace string extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically change the "France" text under the Google logo on the Google main page by "Convertigo".

A transaction, named `Google_fr_transaction`, is defined as default transaction for the *Site Clipper connector* named `GoogleConnector`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources.

A screen class, named `Google_Html_Page`, is defined thanks to a response *MIME type* criterion to handle HTML resources. On this screen class, a *Replace string* extraction rule is added in order to replace the text.

The rule is created with the following parameters:

```
Replace string [
   regular expression=(<div.*>)(France)(</div>)
   replacement=$1Convertigo$3
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

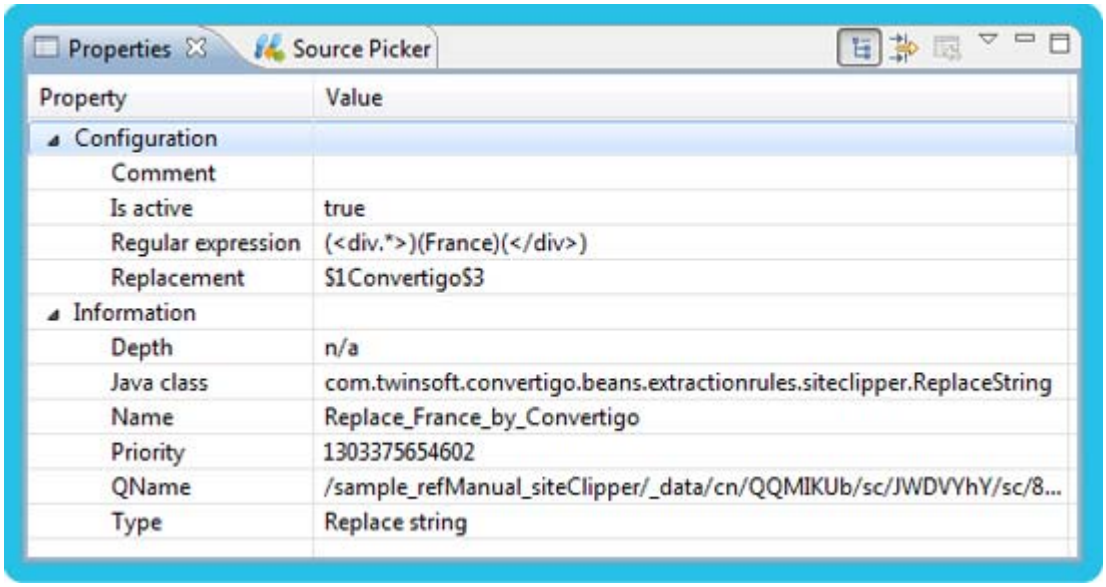


Figure 2 - 658: Replace string extraction rule - Configuration example

The **Regular expression** property is set to a regular expression using groups searching for:

- a first group containing a `DIV` opening tag possibly with attributes,
- a second group containing the word "France",
- a third group containing the `DIV` closing tag.

The groups can then be referenced in the **Replacement** string.This allows recopying the matching strings, i.e. the `DIV` opening tag with its unknown attributes and the `DIV` closing tag, replacing only the second group by the new "Convertigo" string.

The extraction rule appears as follows in the **Projects** view:

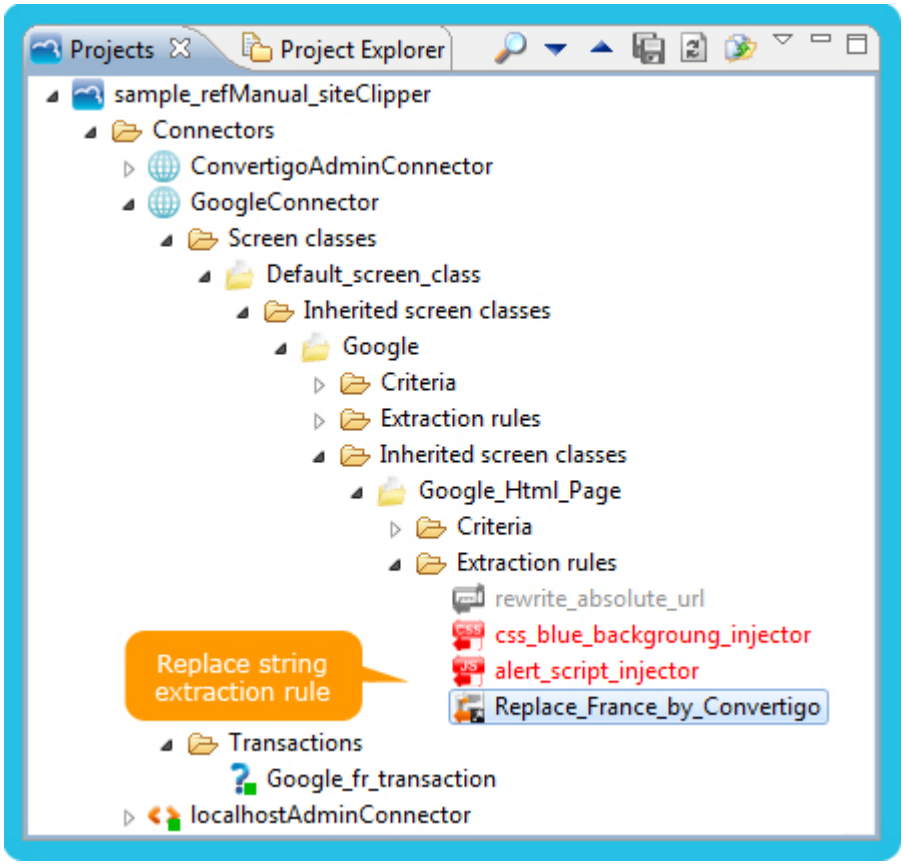


Figure 2 - 659: Replace string extraction rule - Object in Projects view

Switch to a browser displaying the test platform of this project. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page. When this page is loaded, the page's display is changed thanks to the added rule:

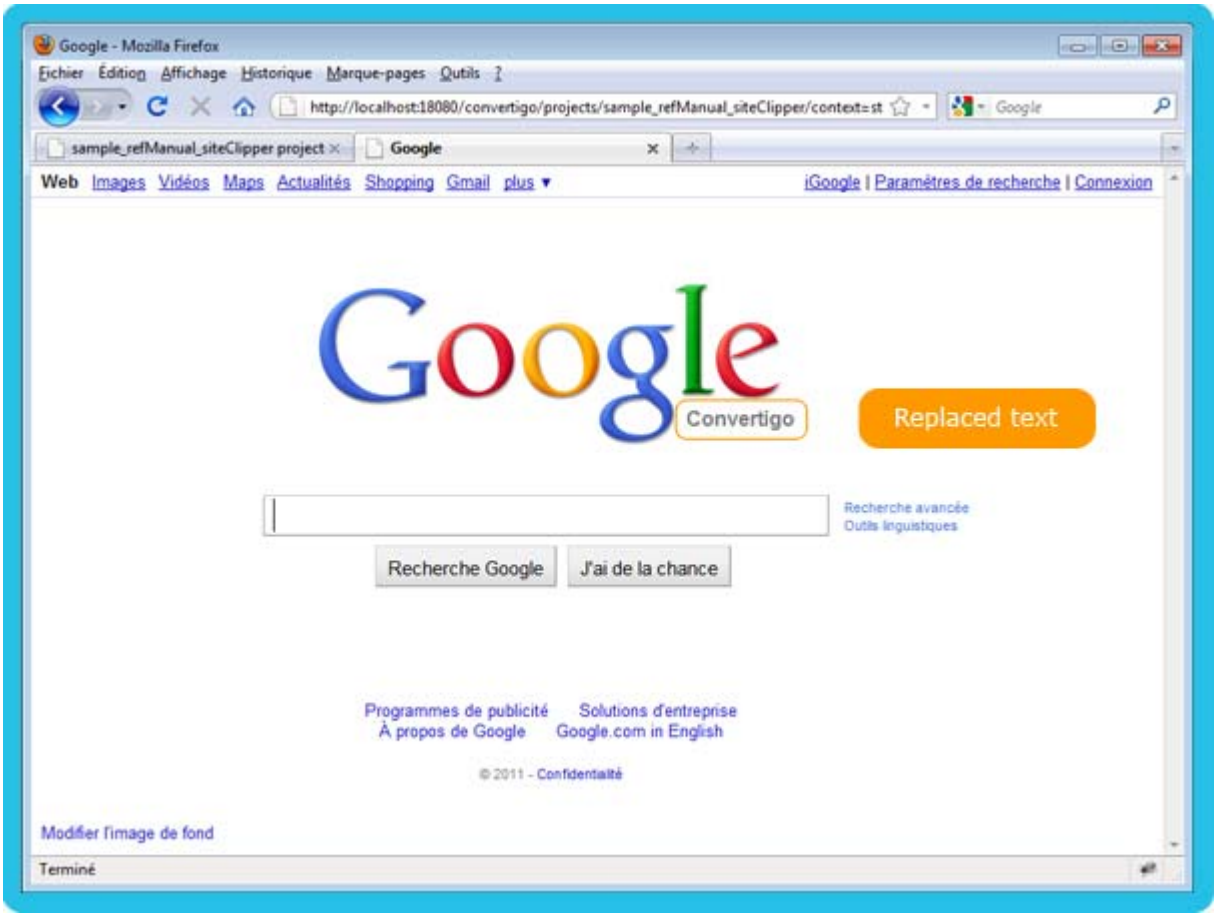


Figure 2 - 660: Replace string extraction rule - "France" text replaced by "Convertigo" on Google main page

Example 2

When you create a new Site Clipper project in Convertigo studio, its default connector is created with:

- a default transaction, named `Default_transaction`, for which you may modify the **Target URL** property value in the creation wizard,
- a default root screen class, named `Default_screen_class`, including predefined inherited screen classes to start using and developing a Site Clipper project.

This architecture should be suitable in most cases.

The root screen class defines a *Rewrite location header* rule to process any request of redirection and an additional *Rewrite absolute url* rule to rewrite URLs found in the HTTP response returned by the server. For more information, see the "*Rewrite location header*" and "*Rewrite absolute URL*" documentations and examples.

One of the screen classes inherited from the `Default_screen_class` screen class, named `Javascript`, includes an additional *Replace string* extraction rule in order to process every necessary string replacements in URLs found in JavaScript resources. Indeed, the *Rewrite absolute url* extraction rule only processes these replacements in HTML and CSS resources.

The rule is created with the following parameters:

```
Replace string [
   regular expression=(["']http[s]?)://
```

```
    replacement=$siteclipper_host$/$1/
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

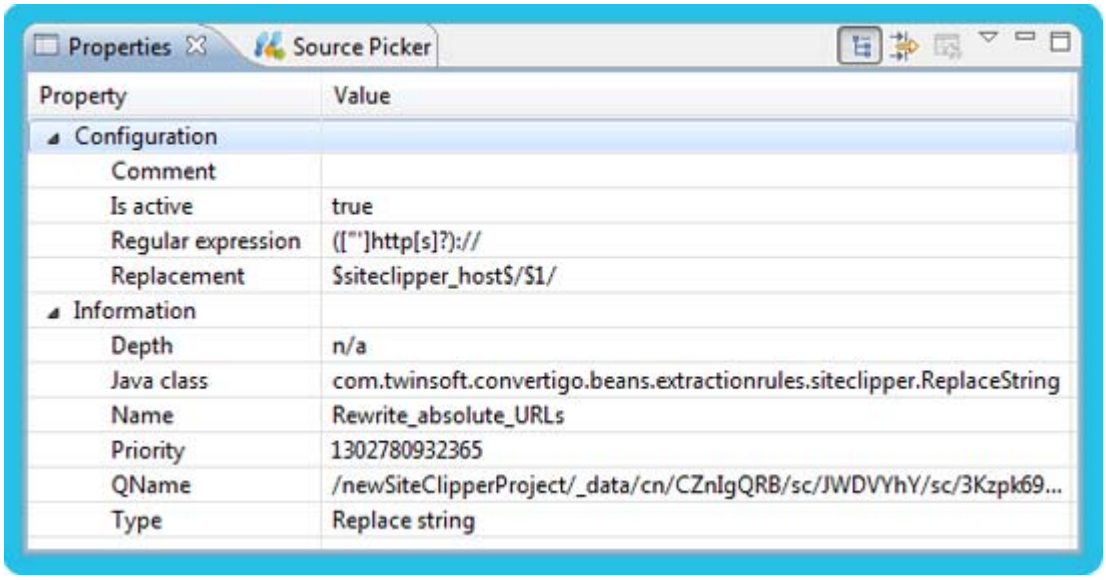


Figure 2 - 661: Replace string extraction rule - Configuration example

The **Regular expression** property is set to a regular expression set in two parts:

- a first group searchs for the beginning of an URL ("`http`" or "`https`"), starting with a quote or double-quote,
- then, the "`://`" characters are described by the second part of the regular expression, not part of a group.

The group can then be referenced in the **Replacement** string, using the `$1` syntax.This allows recopying the matching string, i.e. the `http` or `https` URL beginning with the quote or double-quote first character.

The **Replacement** string also includes a *Convertigo path variable*, identified by the `$siteclipper_host$` syntax. This adds the Site Clipper path in Convertigo server to the rewritten URL. For more information about *Convertigo paths variables*, see Appendix "*Convertigo paths variables - Usable Symbols*".

The extraction rule appears as follows in the **Projects** view:

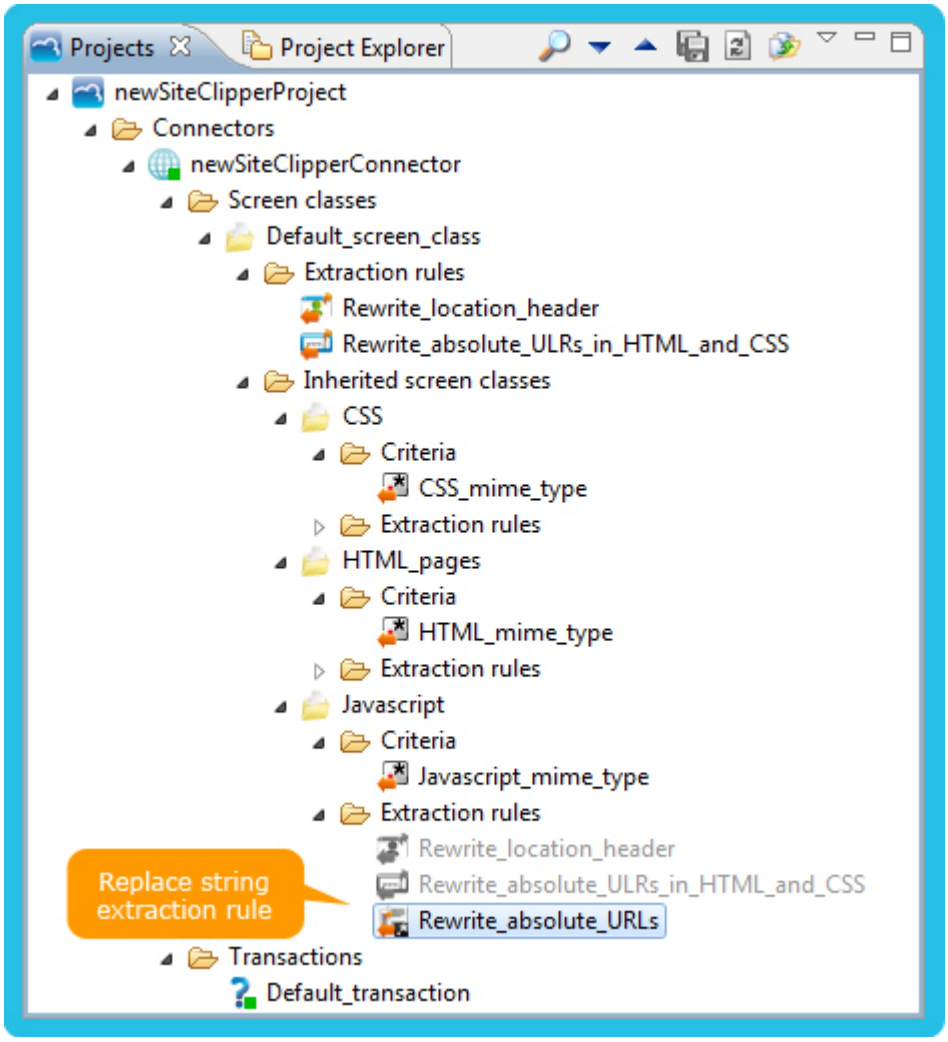


Figure 2 - 662: Replace string extraction rule - Object in Projects view

SCRIPT INJECTOR 

OBJECT DESCRIPTION

Injects script code into a Site Clipper response.

The *Script injector* extraction rule inserts a `<script></script>` element into the received response.

The location where to insert the element is defined by the **Injection location** property which may be customized using the **Custom regexp** property.

The script's source code is contained in an external file which path is defined in the **File path** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Custom regexp	String	selection	Defines the regular expression to use for a custom injection location. When **Injection location** property value is set to `custom`, the **Custom regexp** property allows defining a regular expression to specify the custom location where to insert the HTML element, rather than using the predefined values. Notes: • This expression must contains () to specify where to insert the code. • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
File path	String	configuration	Defines the external file path used for the source attribute of the inserted HTML tag. This property allows specifying the path to an external file which contains the source code. The path value must be relative to the current project folder.

Property	Type	Category	Description
Injection location	HtmlLocation	configuration	Defines the location where to insert the element. This property allows choosing where the new HTML element has to be inserted into the received response. Predefined values are: • `head_top`: inserts after the opening <HEAD> tag, • `head_bottom`: inserts before the closing </HEAD> tag, • `body_top`: inserts after the opening <BODY> tag, • `body_bottom`: inserts before the closing </BODY> tag, • `custom`: inserts at a custom location defined by the **Custom regexp** property.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let's consider the french version of the Google website.

Figure 2 - 663: Script injector extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically pop-up an alert when the Google main page is loaded.

A transaction, named `Google_fr_transaction`, is defined as default transaction for the *Site Clipper connector* named `GoogleConnector`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources.

A screen class, named `Google_Html_Page`, is defined thanks to a response *MIME type* criterion to handle HTML resources. On this screen class, a *Script injector* extraction rule is

added in order to pop-up an alert box.

The rule is created with the following parameters:

```
Script injector [
  file path=js/alert.js
  injection location=head_top
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

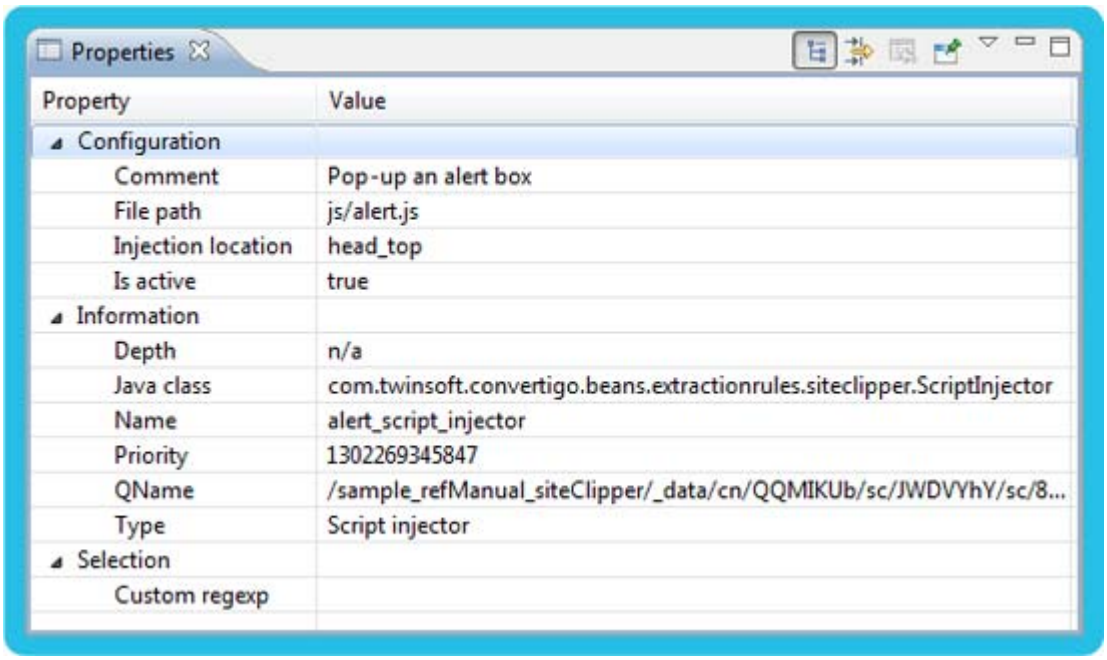


Figure 2 - 664: Script injector extraction rule - Configuration example

The **Injection location** property is set to `head_top` to add the following code after the begining of the HEAD tag in the HTML response:

```
<script src="js/alert.js"></script>
```

The `scr` attribute is filled with the value of the **File path** property.

The extraction rule appears as follows in the **Projects** view:

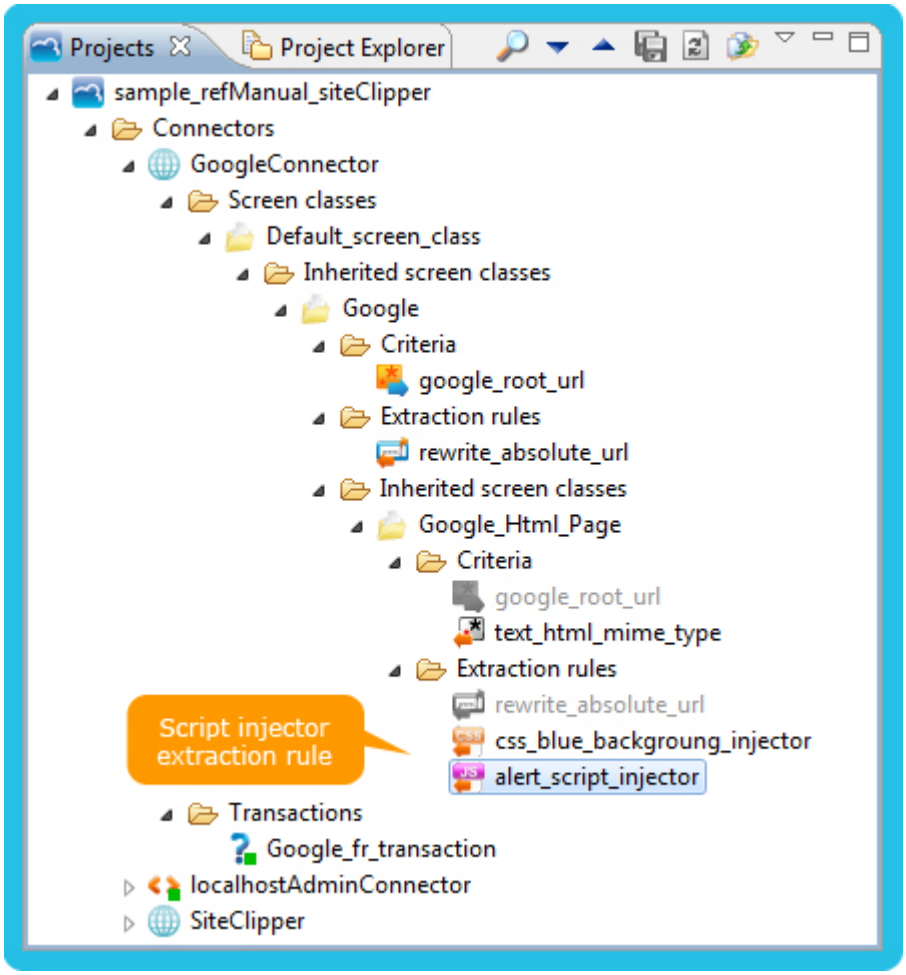


Figure 2 - 665: Script injector extraction rule - Object in Projects view

For this example to be functionnal, we have to create the JavasScript source file that is injected by the rule.

Creating the JavaScript file

The `js/alert.js` file is defined by the *Script injector* rule to be the source of the code to inject. An `alert.js` file is created under the `js` directory of the project:

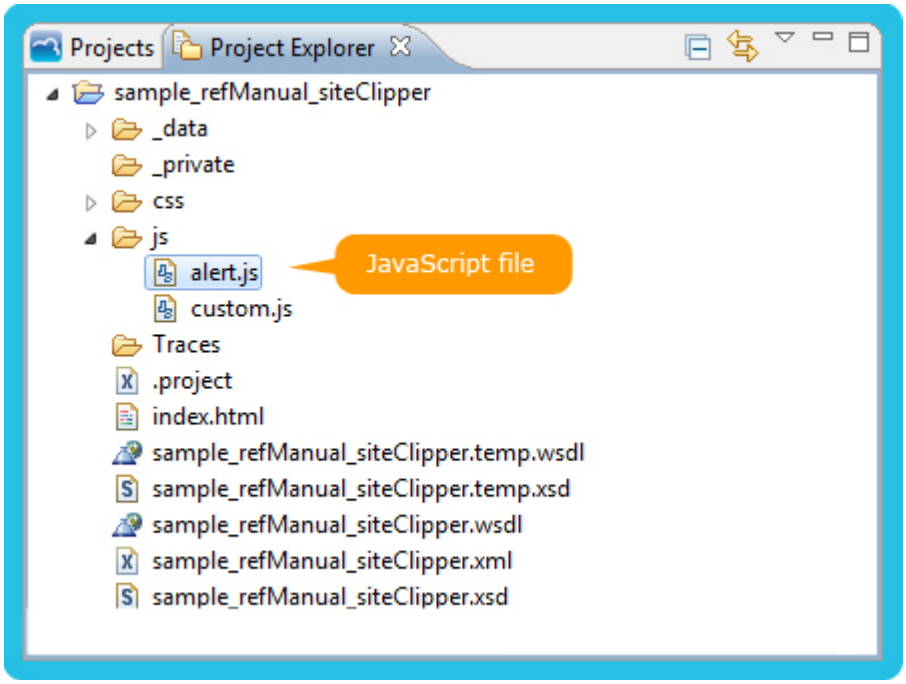


Figure 2 - 666: Script injector extraction rule - Creating alert.js file on project's js folder

As we want to pop-up an alert when the Google main page is loaded, thus we add the following code: `alert("Test a script injector");`

This code is edited in the Convertigo Studio:

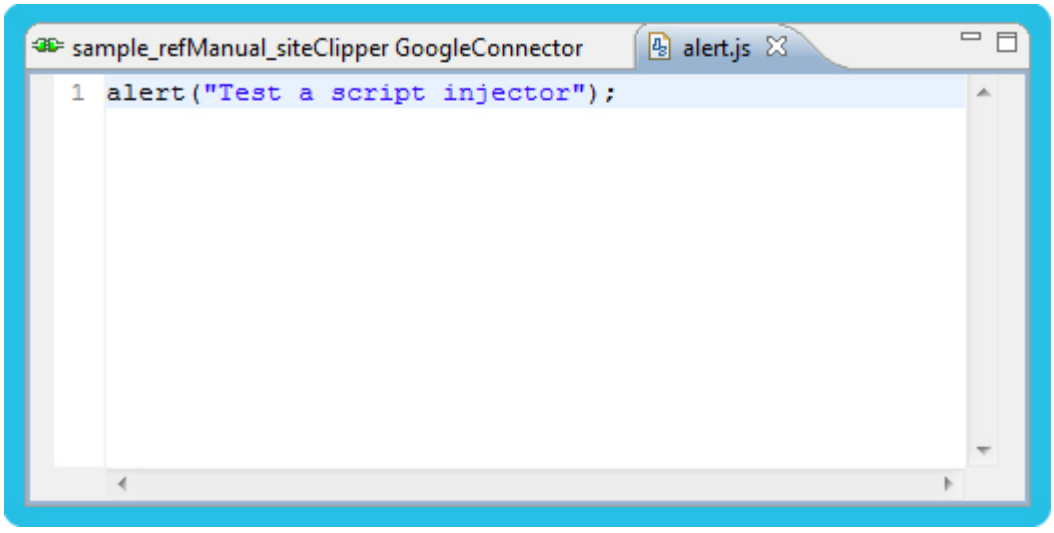


Figure 2 - 667: Script injector extraction rule - Editing alert.js file

Now all configuration is finished, switch to the browser displaying the test platform of this project. Executing the Site Clipper `Google_fr_transaction` transaction reaches the Google main page. When this page is loaded, the page's display is changed thanks to the added rule:

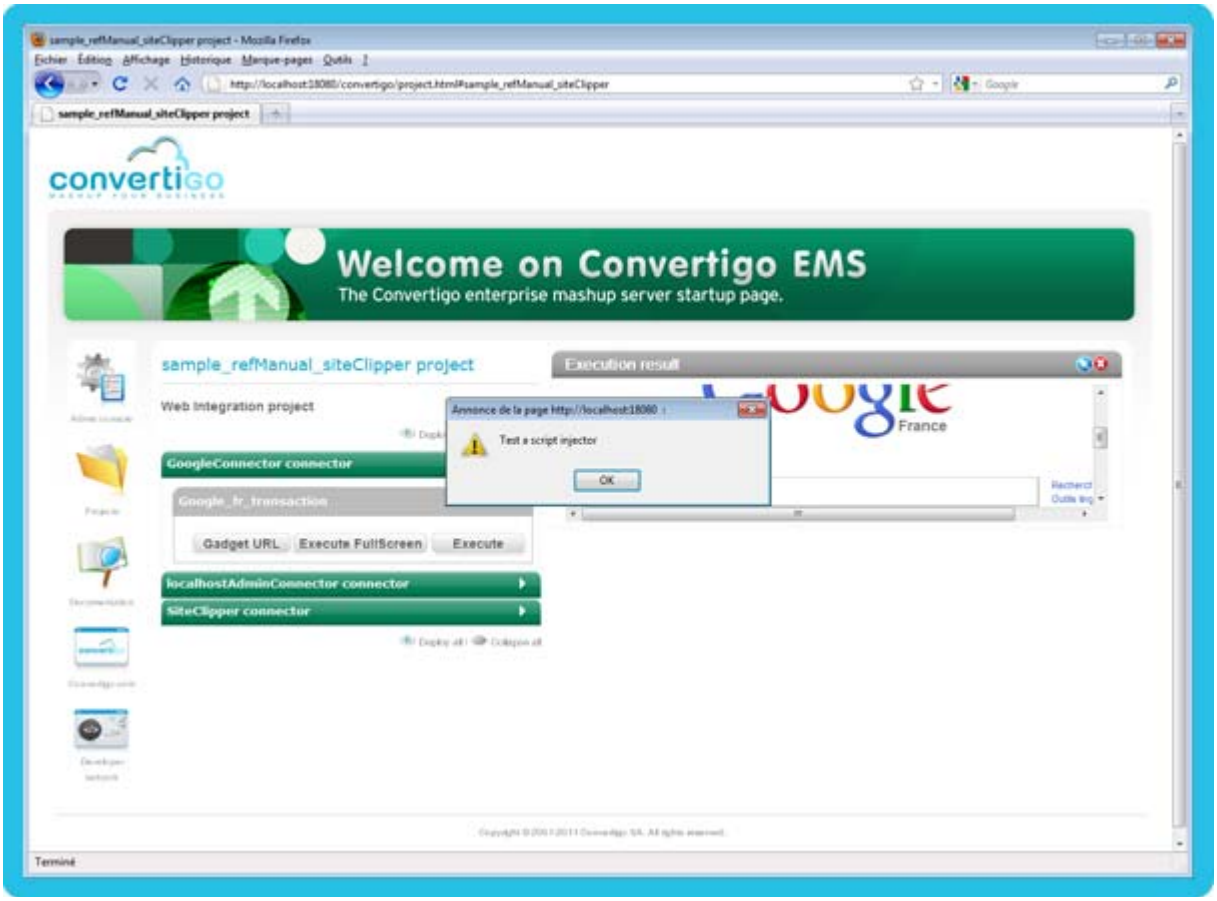


Figure 2 - 668: Script injector extraction rule - Pop-up displayed thanks to JavaScript code injected

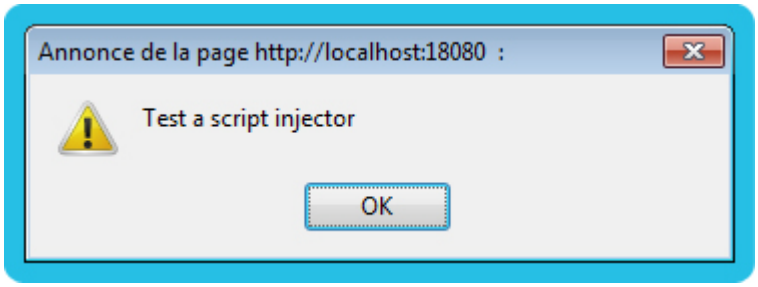


Figure 2 - 669: Script injector extraction rule - Zoom on the pop-up

CSS INJECTOR 

OBJECT DESCRIPTION

Injects style sheet code into a Site Clipper response.

The *CSS injector* extraction rule inserts a `<link />` element into the received response.

The location where to insert the element is defined by the **Injection location** property which may be customized using the **Custom regexp** property.

The style sheet's source code is contained in an external file which path is defined in the **File path** property.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Custom regexp	String	selection	Defines the regular expression to use for a custom injection location. When **Injection location** property value is set to `custom`, the **Custom regexp** property allows defining a regular expression to specify the custom location where to insert the HTML element, rather than using the predefined values. Notes: • This expression must contains () to specify where to insert the code. • For more information about regular expression patterns, see the following page: `http://www.regular-expressions.info/reference.html`. • To test regular expressions, you can use the regular expression tester at the following URL: `http://www.regular-expressions.info/javascriptexample.html`.
File path	String	configuration	Defines the external file path used for the source attribute of the inserted HTML tag. This property allows specifying the path to an external file which contains the source code. The path value must be relative to the current project folder.

Property	Type	Category	Description
Injection location	HtmlLocation	configuration	Defines the location where to insert the element. This property allows choosing where the new HTML element has to be inserted into the received response. Predefined values are: • `head_top`: inserts after the opening <HEAD> tag, • `head_bottom`: inserts before the closing </HEAD> tag, • `body_top`: inserts after the opening <BODY> tag, • `body_bottom`: inserts before the closing </BODY> tag, • `custom`: inserts at a custom location defined by the **Custom regexp** property.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let's consider the french version of the Google website.

Figure 2 - 670: CSS injector extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically change the background color of the center area when the Google main page is loaded.

A transaction, named `Google_fr_transaction`, is defined as default transaction for the *Site Clipper connector* named `GoogleConnector`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources.

A screen class, named `Google_Html_Page`, is defined thanks to a response *MIME type* criterion to handle HTML resources. On this screen class, a *CSS injector* extraction rule is

added in order to change the main area background color.

The rule is created with the following parameters:

```
CSS injector [
  file path=css/bluebg.css
  injection location=head_bottom
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

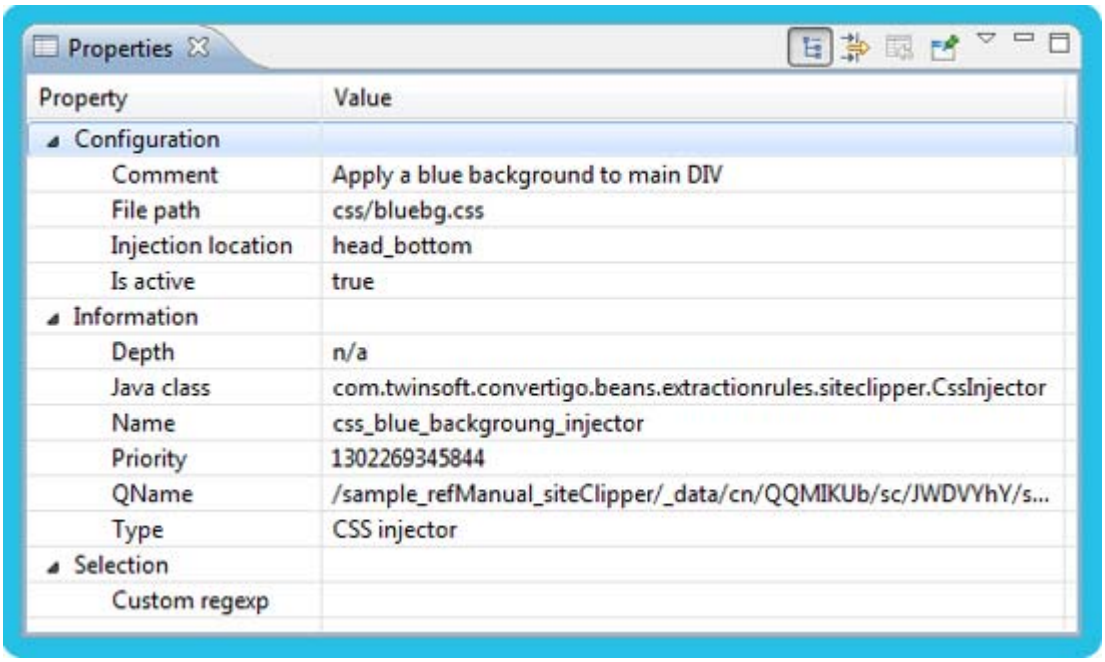


Figure 2 - 671: CSS injector extraction rule - Configuration example

The **Injection location** property is set to `head_bottom` to add the following code before the end of the HEAD tag in the HTML response:

```
<link type="text/css" rel="stylesheet" href="css/bluebg.css"/>
```

The `href` attribute is filled with the value of the **File path** property.

The extraction rule appears as follows in the **Projects** view:

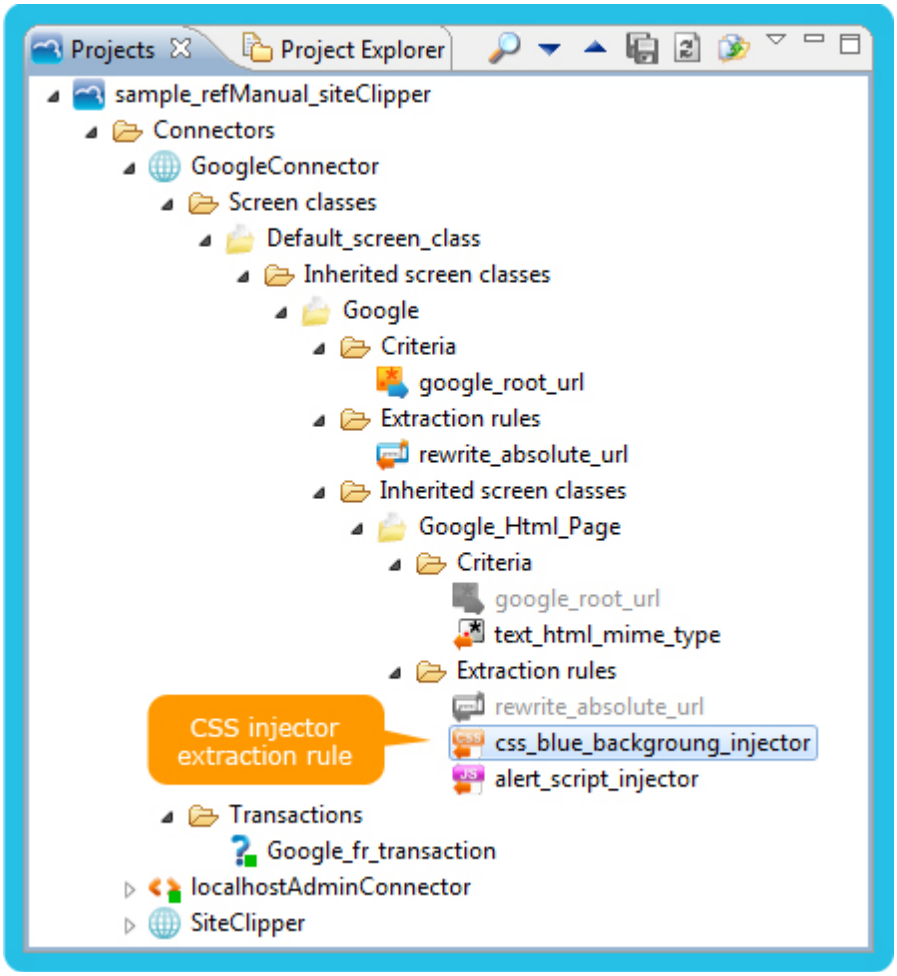


Figure 2 - 672: CSS injector extraction rule - Object in Projects view

For this example to be functionnal, we have to create the CSS source file that is injected by the rule.

Creating the CSS file

The `css/bluebg.css` file is defined by the *CSS injector* rule to be the source of the code to inject. A `bluebg.css` file is created under the `css` directory of the project:

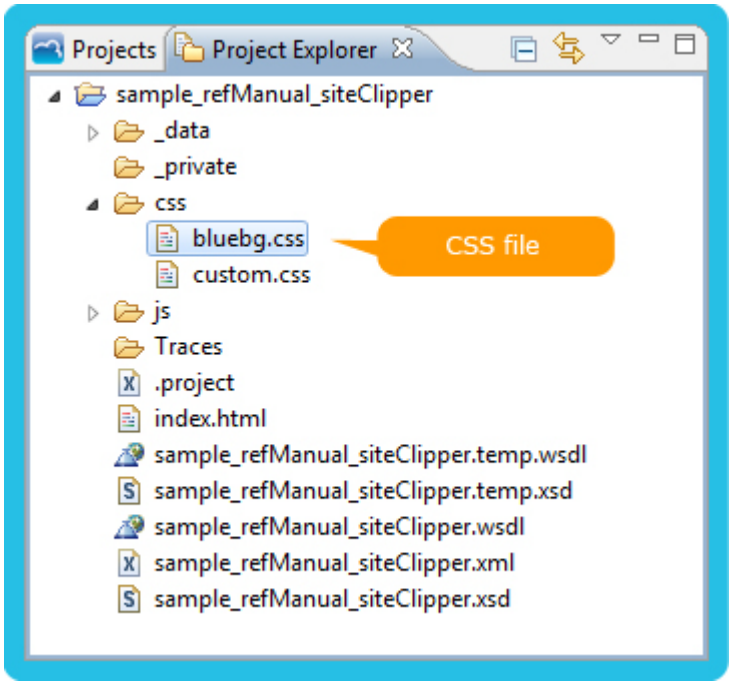


Figure 2 - 673: CSS injector extraction rule - Creating bluebg.css file on project's css folder

The area we are interested in is defined by a `DIV` HTML element with an `id` attribute which value is `main` (`<div id="main">...</div>`). Thus we add the following code to modify this `DIV`'s background color:

```
#main {
   background-color: blue;
}
```

This code is edited in the Convertigo Studio:

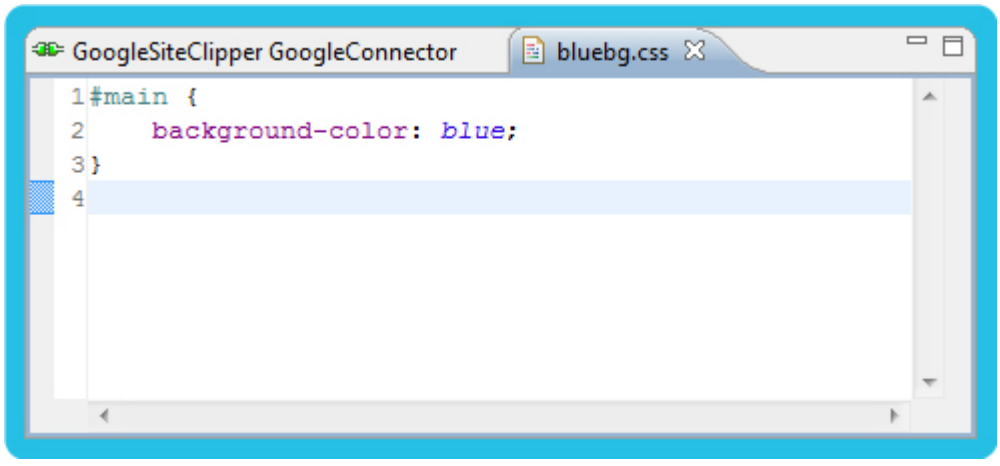


Figure 2 - 674: CSS injector extraction rule - Editing bluebg.css file

Now all configuration is finished, switch to the browser displaying the test platform of this project. Executing the Site Clipper `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google main page. When this page is loaded, the page's display is changed thanks to the added rule:

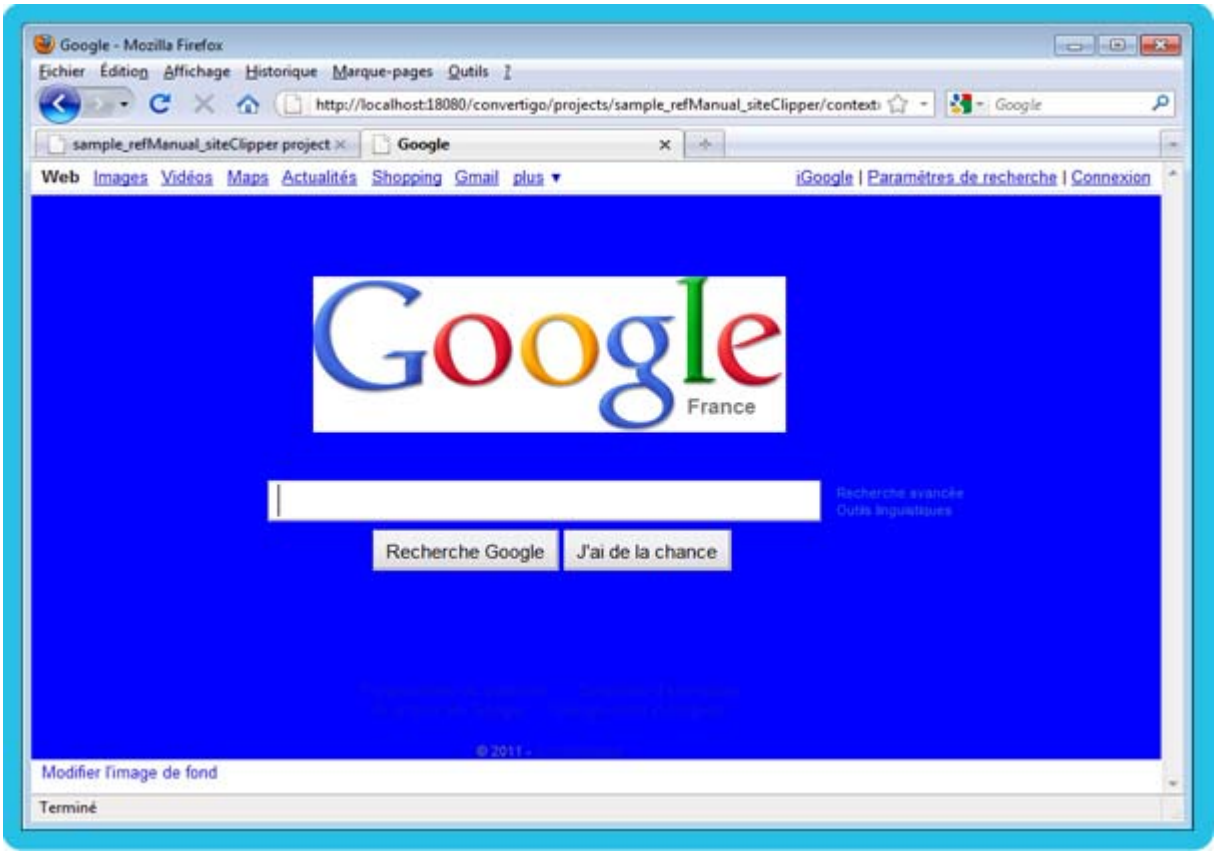


Figure 2 - 675: CSS injector extraction rule - Blue background added thanks to CSS code injected

REWRITE LOCATION HEADER 

OBJECT DESCRIPTION

Rewrites the "`Location`" header value of a Site Clipper response.

The *Rewrite location header* extraction rule rewrites the "`Location`" header value of a Site Clipper response.

This response header is mostly used to redirect the recipient to a location other than the requested one for completion of the request or identification of a new resource. The extraction rule rewrites this URL in order to access the new location through Convertigo Site Clipper.

The *Rewrite location header* extraction rule's behavior depends on the type of URL found in the header. If the URI specified by the location's value is absolute, two cases are possible:

- if the URI doesn't match a black listed domain defined in *Site Clipper connector*, the header value is automatically rewritten,
- else, the header value remains unchanged.

If the URI specified by the location's value is relative, it is automatically rewritten.

Note: If applicable, location's URI is rewritten to an absolute value so that next client's request for the given resource is correctly handled by Convertigo.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let's consider the following page from the US version of Google website (`www.google.com`).

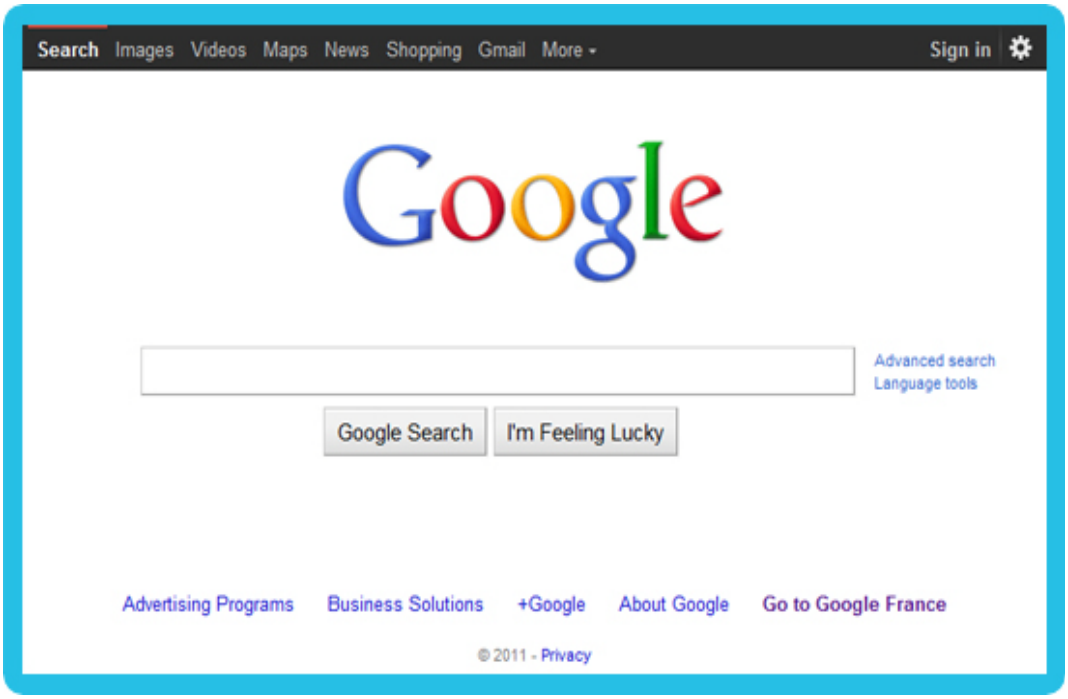


Figure 2 - 676: Rewrite location header extraction rule - US version of Google website

When you access this website with your favorite browser, you're automatically redirected to another address, for example `www.google.fr` if you're located in France.

This redirection is transparent to you because it is fully supported by your browser.

The information is sent by the server located at `www.google.com` in reply to your request through an HTTP header (`Content-Location` or `Location`).

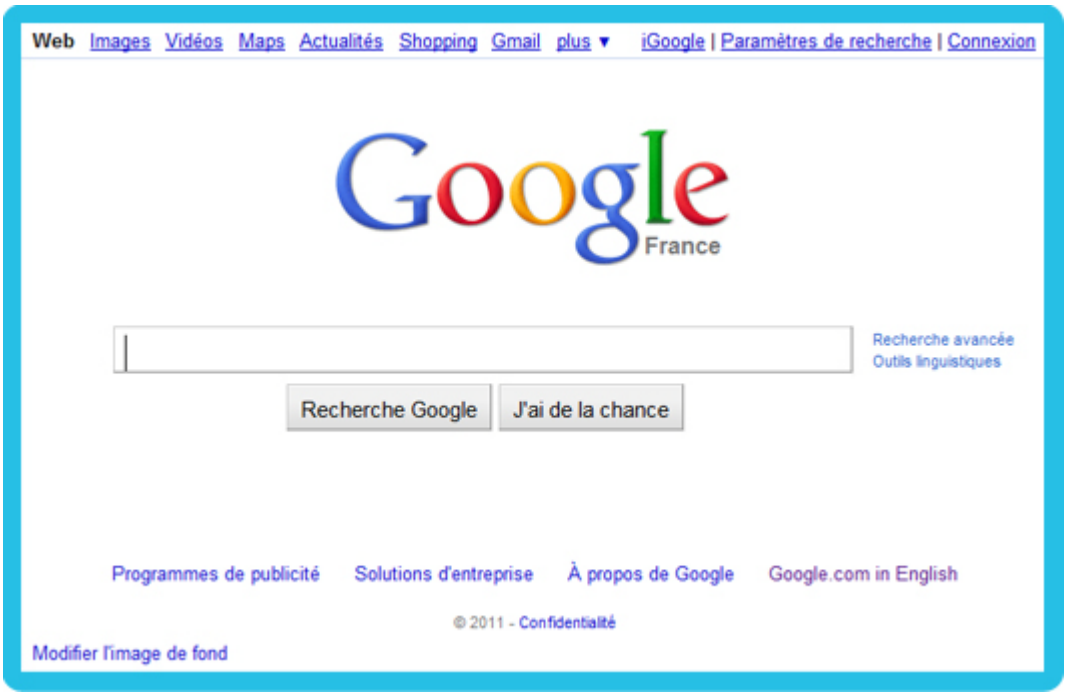


Figure 2 - 677: Rewrite location header extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to automatically rewrite the targeted location's URL so that redirection request is processed by Convertigo rather than the targeted

server.

A transaction, named `Google_com_transaction`, is created in the *Site Clipper connector* named `Google_com_Connector`. It defines the URL `http://www.google.com` as target URL to connecto to Google search page.

Switch to a browser displaying the test platform of this project. Executing the `Google_com_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google.com page. When the automatic redirect to Google France is performed, the page is accessed directly. You can see the URL `http://www.google.fr/` directly reached in the browser:

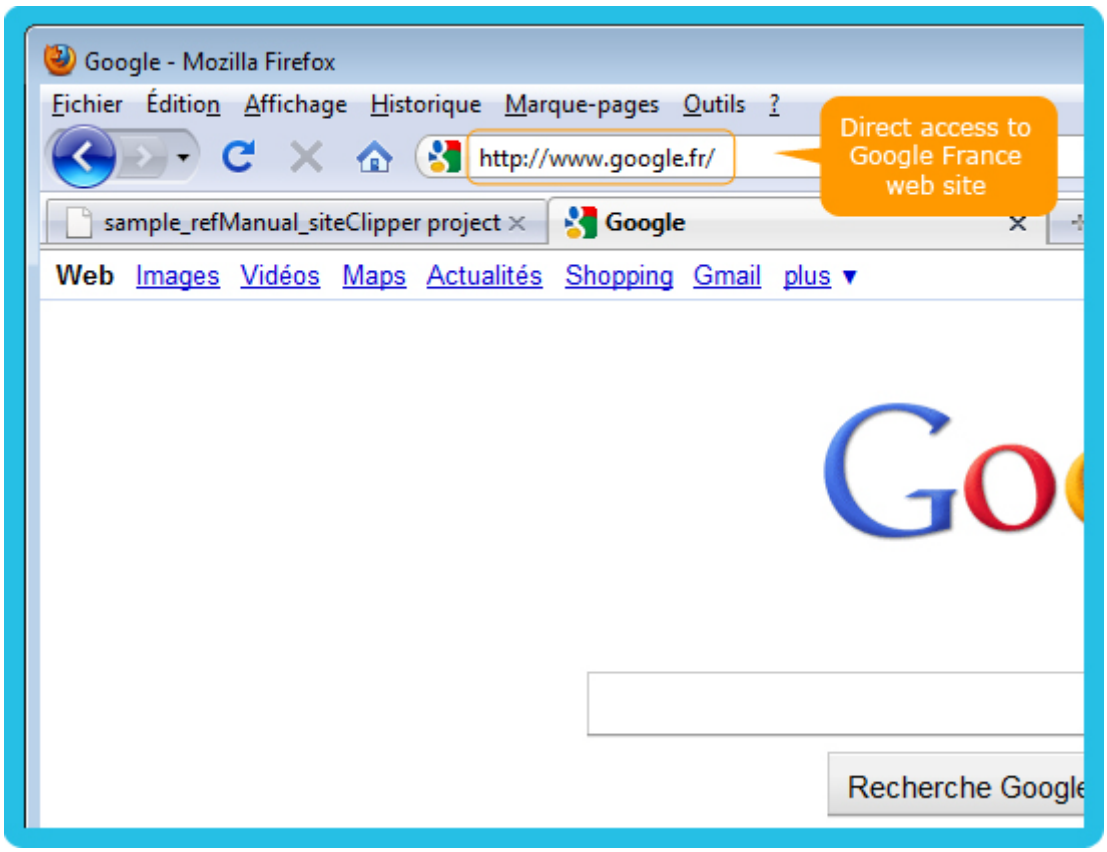


Figure 2 - 678: Rewrite location header extraction rule - Redirect to Google France reached directly

In the *Site Clipper connector*, a root screen class, named `Google`, is defined thanks to a request *URL* criterion to handle Google ressources only. On this screen class, a *Rewrite location header* extraction rule is added in order to automatically rewrite target URL of redirects.

A *Rewrite location header* extraction rule has no properties to configure:

```
Rewrite location header [
]
```

It appears as follows in the **Properties** view of the Convertigo Studio:

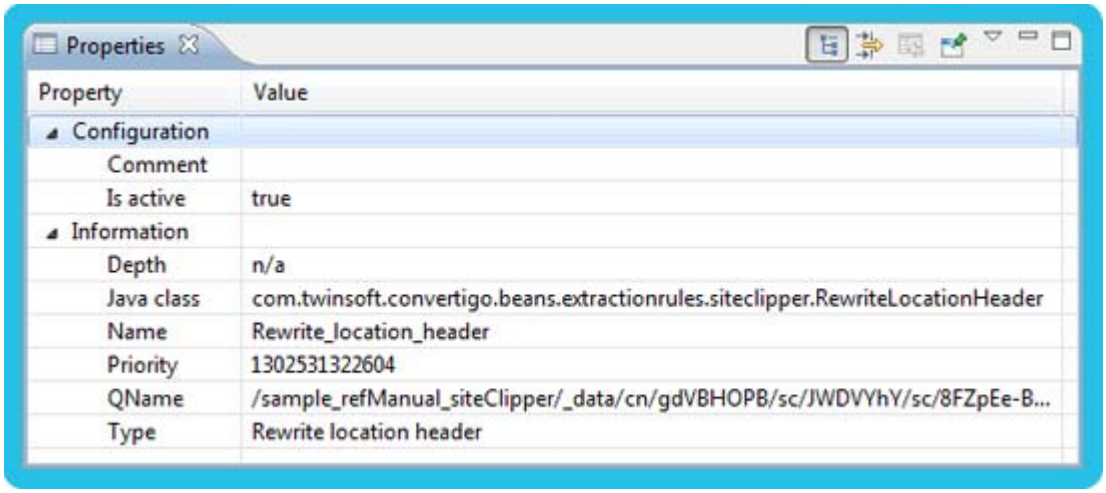


Figure 2 - 679: Rewrite location header extraction rule - Configuration example

The extraction rule appears as follows in the **Projects** view:

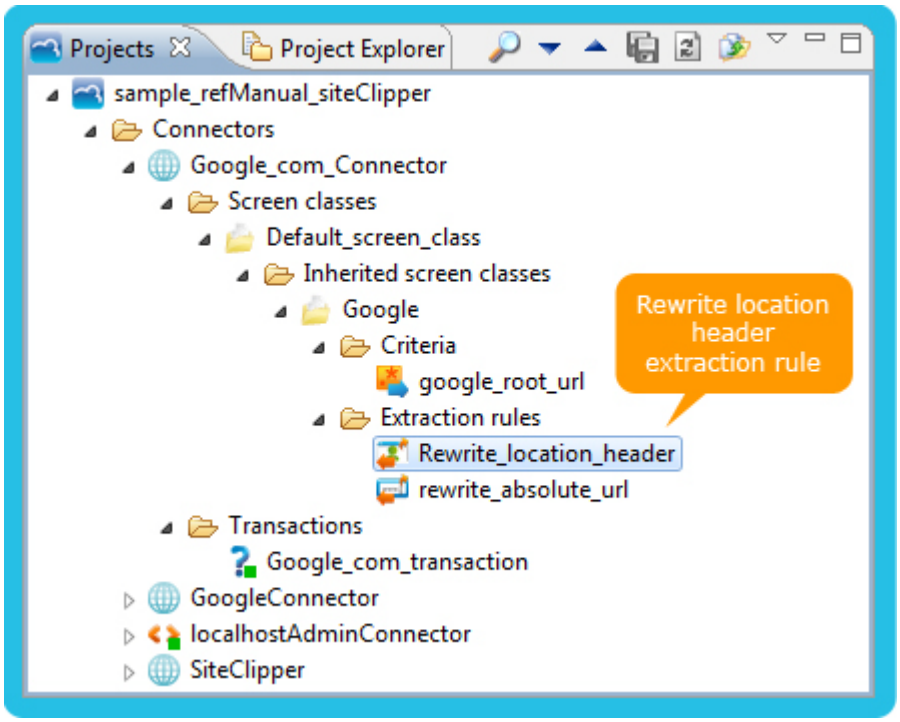


Figure 2 - 680: Rewrite location header extraction rule - Object in Projects view

Switch back to the browser displaying the test platform of this project. Executing the `Google_com_transaction` transaction again (in a new tab thanks to **Execute full screen** button) reaches the Google.com page. When the automatic redirect to Google France is performed, the page is accessed through Convertigo. You can see the URL starting by the Convertigo server URL `http://localhost:18080/convertigo/` reached in the browser:

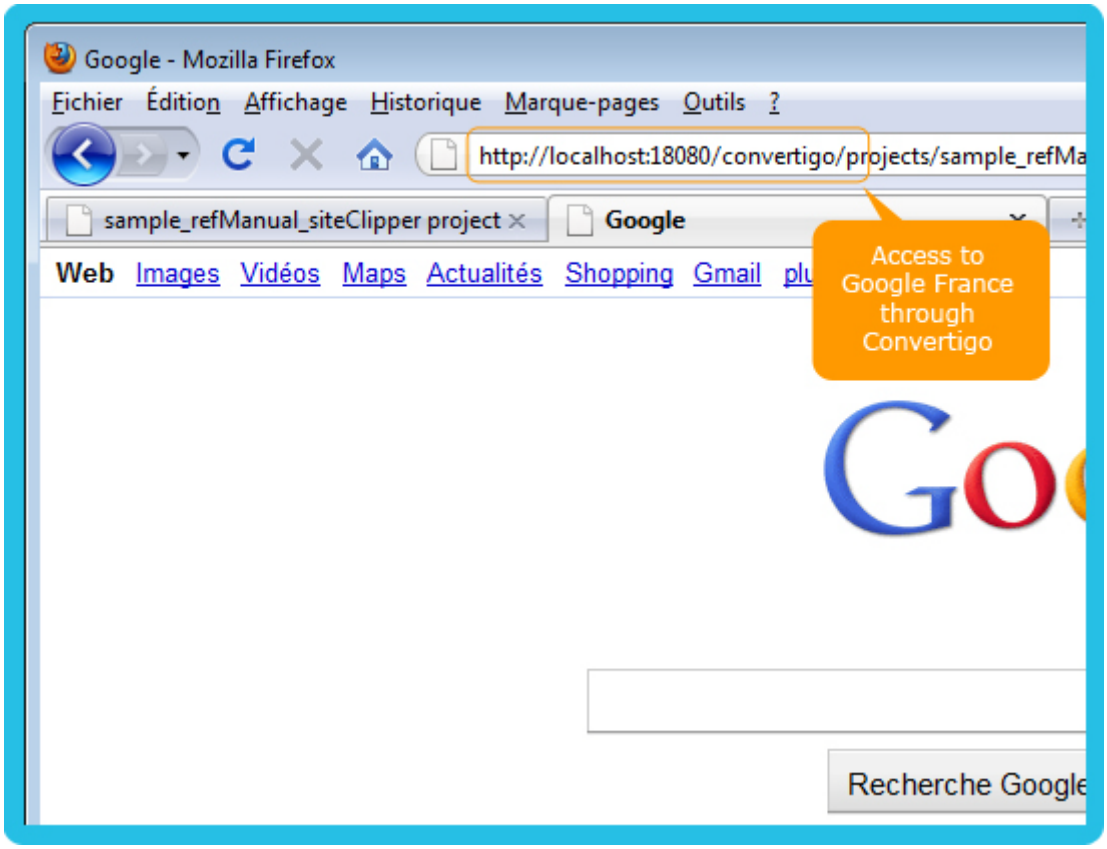


Figure 2 - 681: Rewrite location header extraction rule - Redirect to Google France through Convertigo

REWRITE ABSOLUTE URL 

OBJECT DESCRIPTION

Rewrites absolute URLs found in a Site Clipper response.

The *Rewrite absolute URL* extraction rule rewrites absolute URLs found in Site Clipper HTTP responses, for the links or resources accessed by these URLs to be accessed through Convertigo Site Clipper.

This extraction rule will be executed only on HTTP responses that match certain MIME types:

- If **Rewrite HTML code** property value is set to `true`, URLs found in HTML code are rewritten. That means the extraction rule is searching for URLs to rewrite in HTTP responses of `text/html` or `application/xhtml+xml` MIME types.
- If **Rewrite CSS code** property value is set to `true`, URLs found in CSS code are rewritten. That means the extraction rule is searching for URLs to rewrite in HTTP responses of `text/css`, `text/html` or `application/xhtml+xml` MIME types.

Notes:

- Absolute URLs are rewritten only if they don't match a black listed domain defined in the **Domains listing** property of the associated *Site Clipper connector*.
- If applicable, relative URLs are rewritten to absolute ones.
- URLs found in JavaScript code will not be rewritten by this rule. To do so, use a *Replace string* extraction rule parametered for your specific case.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.
Rewrite CSS code	boolean	configuration	Automatically rewrite absolute URLs in CSS code. This property allows to specify whether absolute URLs found in the CSS code of a Site Clipper response should be rewritten or not. Specifically to this rule, CSS code of a Site Clipper response is defined by an HTTP response of the following MIME types: • `text/css`: Cascading Style Sheet text resource, • `text/html`: HTML text resource, • `application/xhtml+xml`: XHTML file resource. If this property is set to `true`, URLs specified in following CSS keyword are rewritten: • `url` keyword.

Property	Type	Category	Description
Rewrite HTML code	boolean	configuration	Automatically rewrite absolute URLs in HTML code. This property allows to specify whether absolute URLs found in the HTML code of a Site Clipper response should be rewritten or not. Specifically to this rule, HTML code of a Site Clipper response is defined by an HTTP response of the following MIME types: • `text/html`: HTML text resource, • `application/xhtml+xml`: XHTML file resource. If this property is set to `true`, URLs specified in following HTML attributes are rewritten: • `action` attribute, • `background` attribute, • `cite` attribute, • `classid` attribute, • `codebase` attribute, • `data` attribute, • `href` attribute, • `longdesc` attribute, • `profile` attribute, • `src` attribute, • `usemap` attribute.

EXAMPLES

Let's consider the french version of the Google website.

Figure 2 - 682: Rewrite absolute URL extraction rule - French version of Google website

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically access all resources and pages from Google website through Convertigo.

A transaction, named `Google_fr_transaction`, is created in the *Site Clipper connector* named `Google_com_Connector`. It defines the URL `http://www.google.fr` as target URL to connect to Google France search page.

Switch to a browser displaying the test platform of this project. Executing the `Google_fr_transaction` transaction (in a new tab thanks to **Execute full screen** button) reaches the Google.fr page through Convertigo:

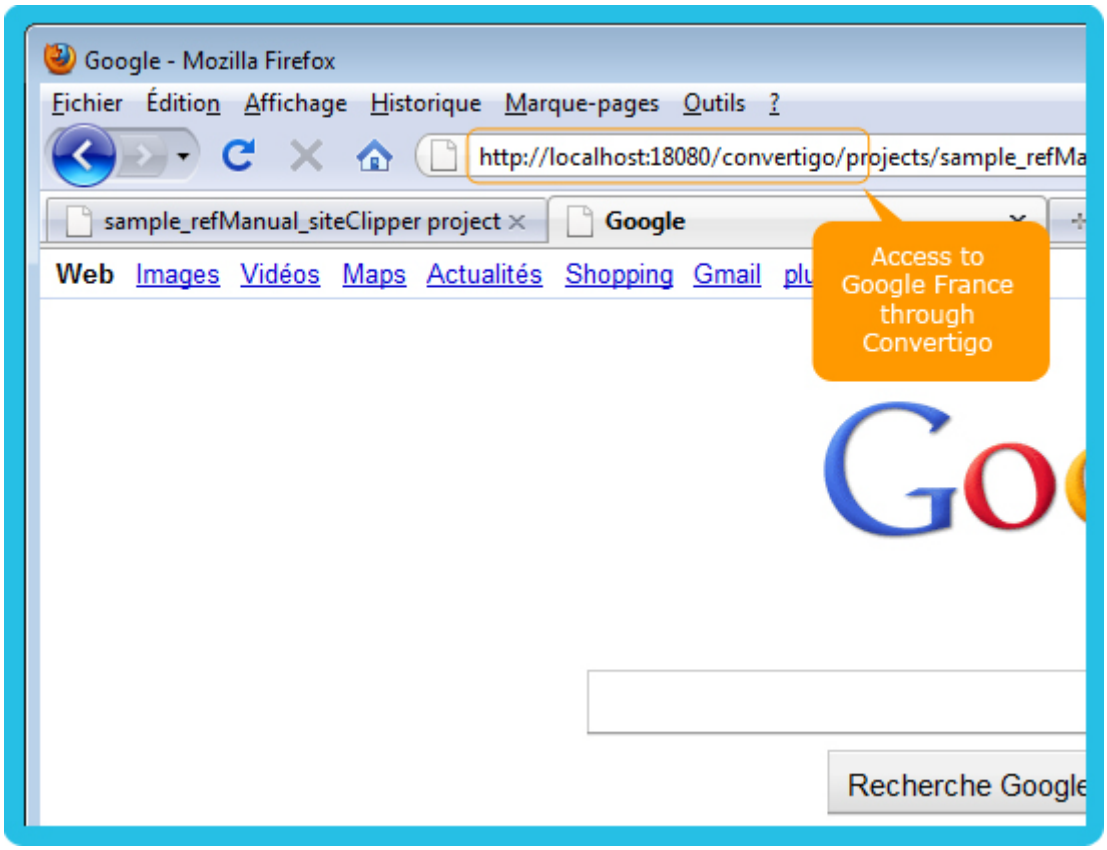


Figure 2 - 683: Rewrite absolute URL extraction rule - Google France accessed through Convertigo

Rolling the mouse over the top left links (Images, Maps, etc.) shows links reaching directly Google website (`http://maps.google.fr/...`, `http://news.google.fr/...`, etc.):

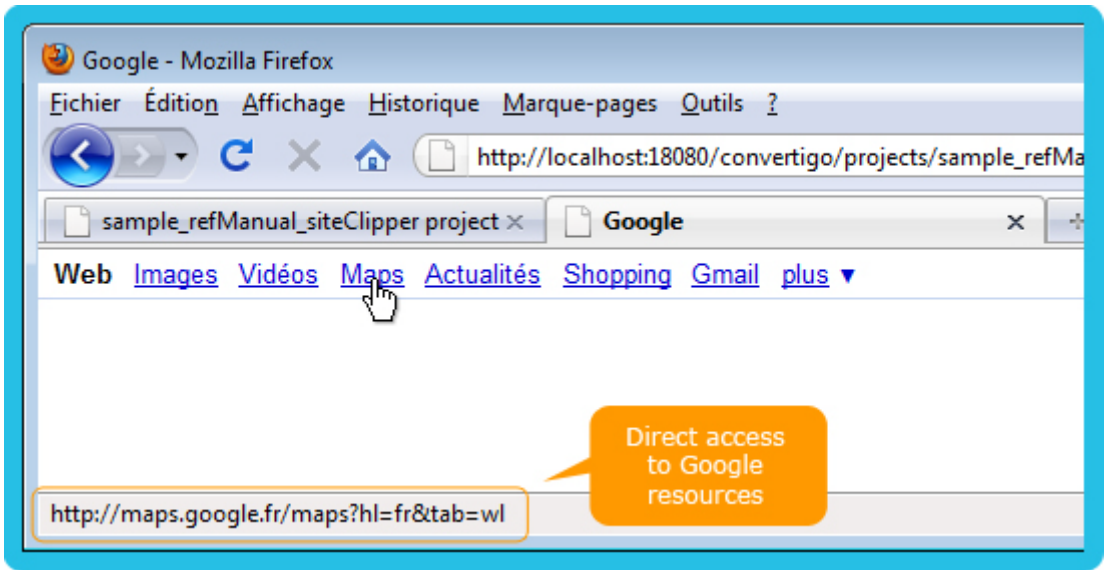


Figure 2 - 684: Rewrite absolute URL extraction rule - Links not reaching Convertigo

In the *Site Clipper connector*, a root screen class, named `Google`, is defined thanks to a request *URL* criterion to handle Google ressources. On this screen class, a *Rewrite absolute*

URL extraction rule is added in order to automatically rewrite target URL of every link and resources.

The rule is created with the following parameters:

```
Rewrite absolute URL [
   rewrite CSS code=true
   rewrite HTML code=true
]
```

It appears as follows in the **Properties** view of the Convertigo Studio:

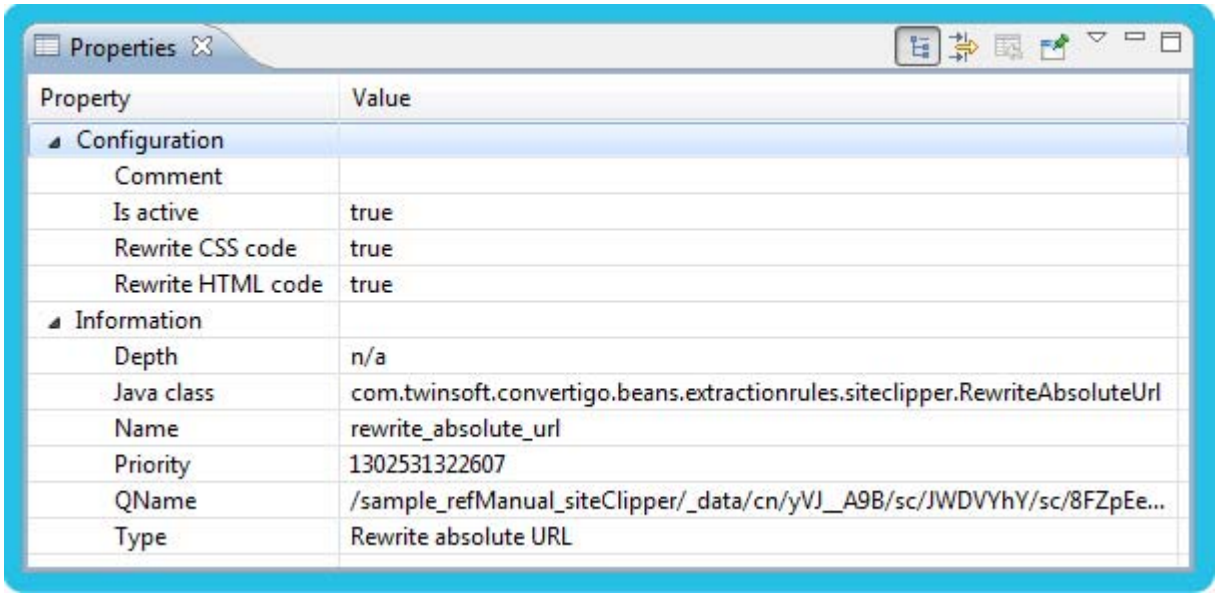


Figure 2 - 685: Rewrite absolute URL extraction rule - Configuration example

The **Rewrite CSS code** and **Rewrite HTML code** properties are set to `true` for URLs included in HTML code and in CSS code to be rewritten. This leads to access all linked resources through Convertigo.

Only JavaScript code cannot be automatically rewritten by this rule: each case is specific to the target website. To access resources linked in JavaScript code through Convertigo, you can add a *String replace* extraction rule and manually configure the corresponding case.

The extraction rule appears as follows in the **Projects** view:

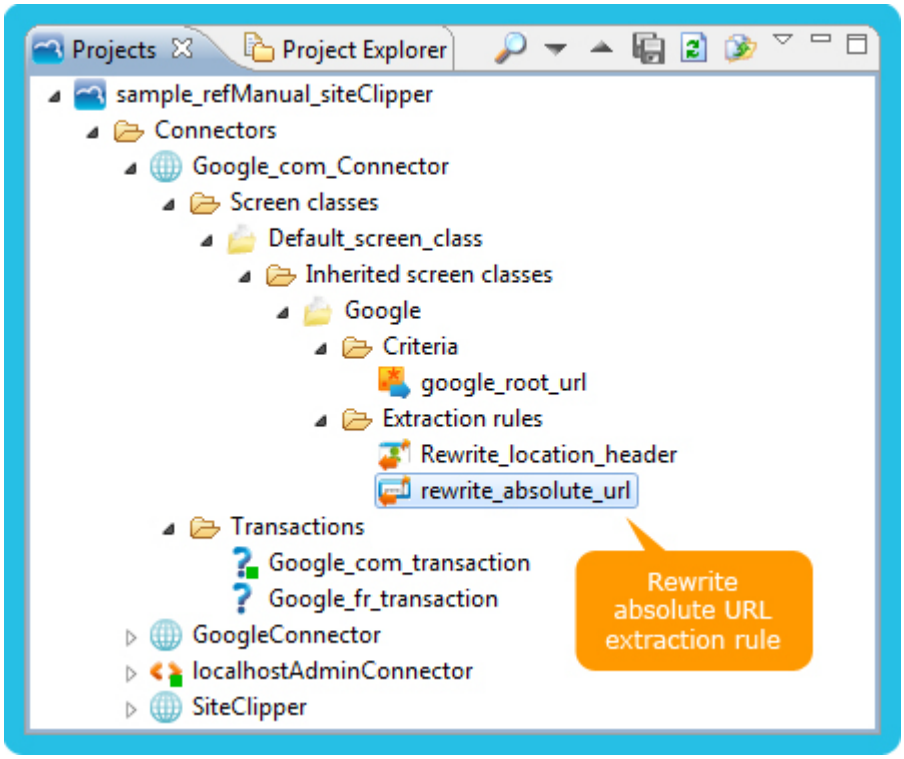


Figure 2 - 686: Rewrite absolute URL extraction rule - Object in Projects view

Switch back to the browser displaying the test platform of this project. Executing the `Google_fr_transaction` transaction again (in a new tab thanks to **Execute full screen** button) reaches the Google.fr page through Convertigo.

Rolling the mouse over the top left links (Images, Maps, etc.) shows links accessing Google resources through Convertigo (URL starting with `http://localhost:18080/convertigo/...`):

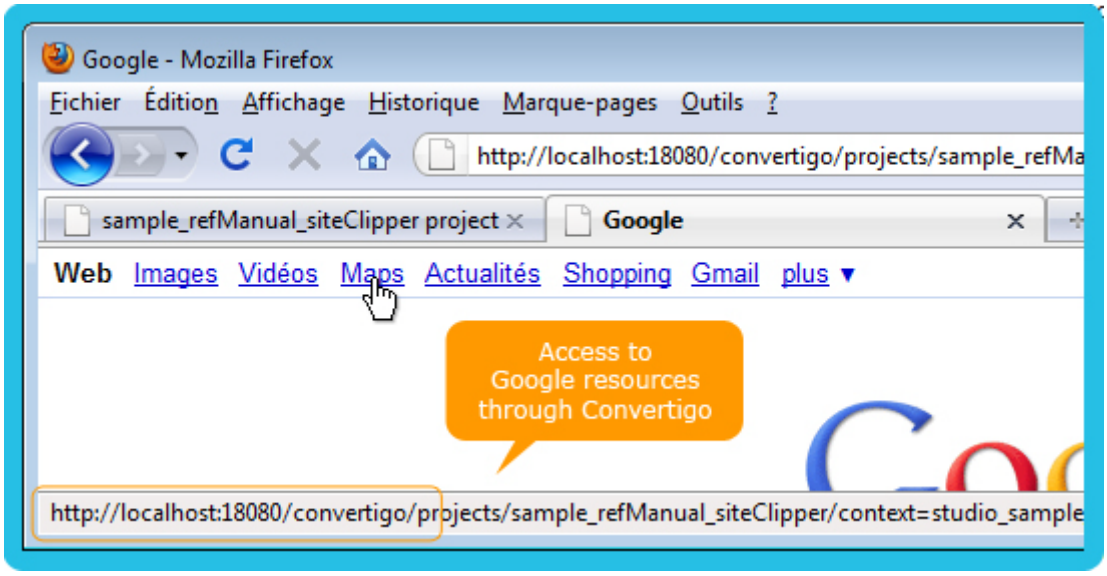


Figure 2 - 687: Rewrite absolute URL extraction rule - Accessing Google resources through Convertigo

CLIENT INSTRUCTION SET VALUE 

OBJECT DESCRIPTION

Automatically sets a value in a target element from a web page accessed through Convertigo Site Clipper. The action is performed in the client browser after the page is loaded.

The *Client instruction set value* extraction rule stores an instruction of setting a value in an element in a queue of client instructions. This queue is then unstacked when the page is client-side loaded. The instruction of setting a value in a target element is performed on the page when it is loaded by the client browser.

The target element (input field or text area) is defined thanks to the **JQuery selector** property and the value to enter in the field is defined by the **Value** property. These properties are JavaScript expressions, evaluated by Convertigo when the extraction rule is applied.

In order to process the unstacking of client instructions in the web page, `client instruction engine` code is injected into the page after all response extraction rules are applied. When the page is client-side loaded, the `client instruction engine` runs and consumes each registered client instruction, in the same order as extraction rules.

Consuming the *Client instruction set value* instruction, the engine selects the target element using the **JQuery selector** and sets the `value` attribute with the value computed by the rule using the **Value** property.

Note: The value is set using the JQuery `val` function. For more information about this function, see the following page: `http://api.jquery.com/val/`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
JQuery selector	JS expression	configuration	JavaScript expression defining a JQuery selector matching the target element from the HTML document. This property defines a JQuery selector used to retrieve the target element (input field, text area, select, checkbox, radio button, etc.) on the client browser when the page is loaded. It is defined thanks to a JavaScript expression evaluated using the JavaScript scope of the current context. It should return a string to be used as a JQuery selector. Here are some default syntaxes for JQuery selectors. An id selector starts with # character, an attribute selector is between `[]` characters, a class selector starts with `.` character. The > character separating several selectors defines a constraint to the direct ancestor. The space character separating several selectors defines a constraint to any ancestor. The followings are some simple selectors that can be used in JQuery: • Select an element by id: `#the_id_to_search`, • Select an any type of element by name: `[name="the_name_to_search"]` or also `*[name="the_name_to_search"]`, • Select an input element by name: `input[name="the_name_of_the_input_to_search"]`, • Select an any type of element by class: `.class_to_search`, • Select a div element by class: `div.class_of_a_div_to_search`, • Select an input by name, direct child of a form element selected by class: `form.form_class > input[name="input_name"]`, • Select an hidden input, descendant of a specified id: `#specified_id input[type="hidden"]`. Note: The full JQuery selector documentation is available on the official JQuery website: `http://api.jquery.com/category/selectors/`.
Value	JS expression	configuration	JavaScript expression defining the string value to set in the target element. This property defines the value to be entered in the target element. It is defined thanks to a JavaScript expression evaluated using the JavaScript scope of the current context. It should return a string to be used as a text value. The target element should be: • an `INPUT` element of `text`, `hidden` or `password` type, • a `SELECT` element, • a `TEXTAREA` element.

EXAMPLES

Let's consider the US directory White Pages website, US directory pages to search for people. This search form needs a last name and may need a first name, a city, a state and some other optional fields:

Figure 2 - 688: Client instruction set value extraction rule - Whites pages on US directory website

An *HTML transaction*, named `SearchWhitePages`, is created to navigate in the US directory website to access the White Pages search page. This transaction gives the control back to the user on this form page through a Site Clipper connector, thanks to a *Continue with Site Clipper* statement.

This transaction defines several variables, such as `first_name`, `last_name` and `state` (and possibly other variables), corresponding to the interesting fields of the previous page's form. When invoked, the HTML transaction creates a new Convertigo context and sets its variables values into its JavaScript scope.

When the user gets the control back through the Site Clipper connector, we want the search form to be prefilled with the values declared in the previous HTML transaction variables.

In the *Site Clipper connector*, named `USdirectory`, a screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A `Person` screen class is defined to match on the previous web page.

In order to fill each input field of the form, *Client instruction set value* rules are created on the `Person` screen class, one per field.

- For `text` type inputs, rules are created with the following parameters:

```
Client instruction set value [
   jQuery selector="input[name='firstname']"
   value=first_name
]
Client instruction set value [
   jQuery selector="input[name='lastname']"
   value=last_name
]
```

- For `select` combo box, the rule is created with the following parameters:

```
Client instruction set value [
   jQuery selector="select[name='state_id']"
   value=state
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

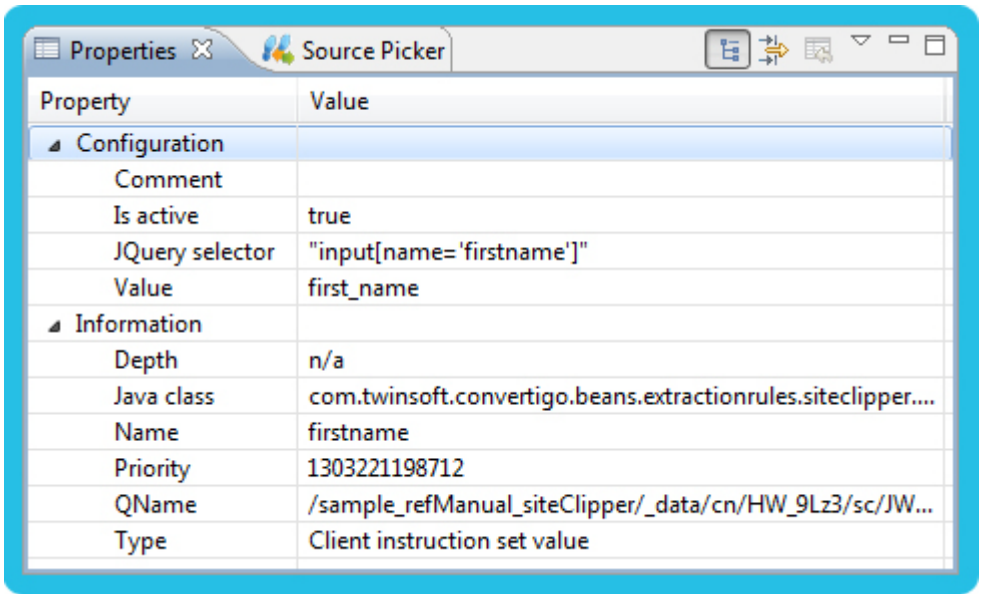


Figure 2 - 689: Client instruction set value extraction rule - Configuration example for text type input

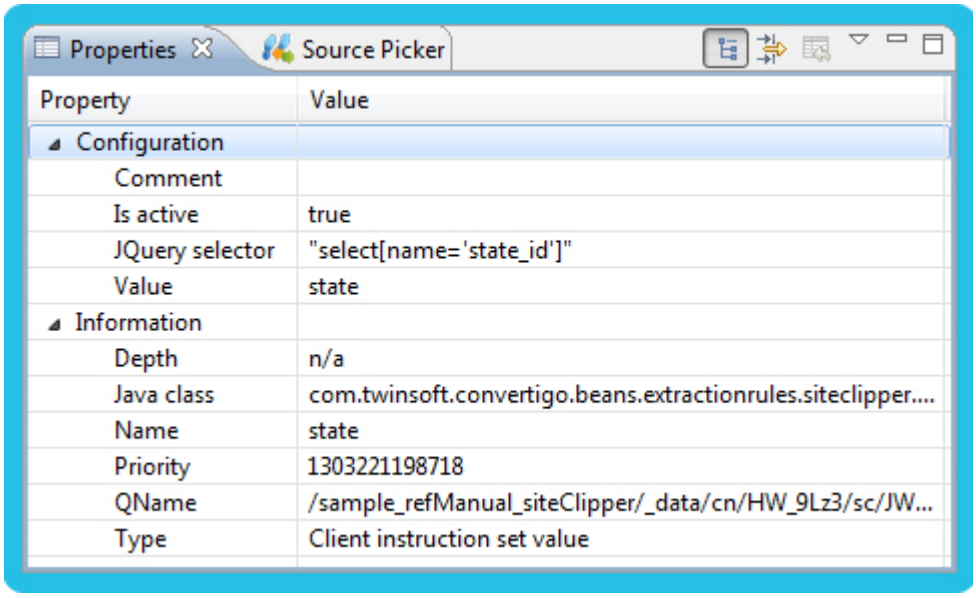


Figure 2 - 690: Client instruction set value extraction rule - Configuration example for select

The **JQuery selector** properties are defined to select `input` or `select` element by their `name` attribute. The **Value** properties are set to JavaScript expressions using the values of the corresponding variables available in the Site Clipper context JavaScript scope because they were set in the scope by the HTML transaction which received them as input.

The extraction rules appear as follows in the **Projects** view:

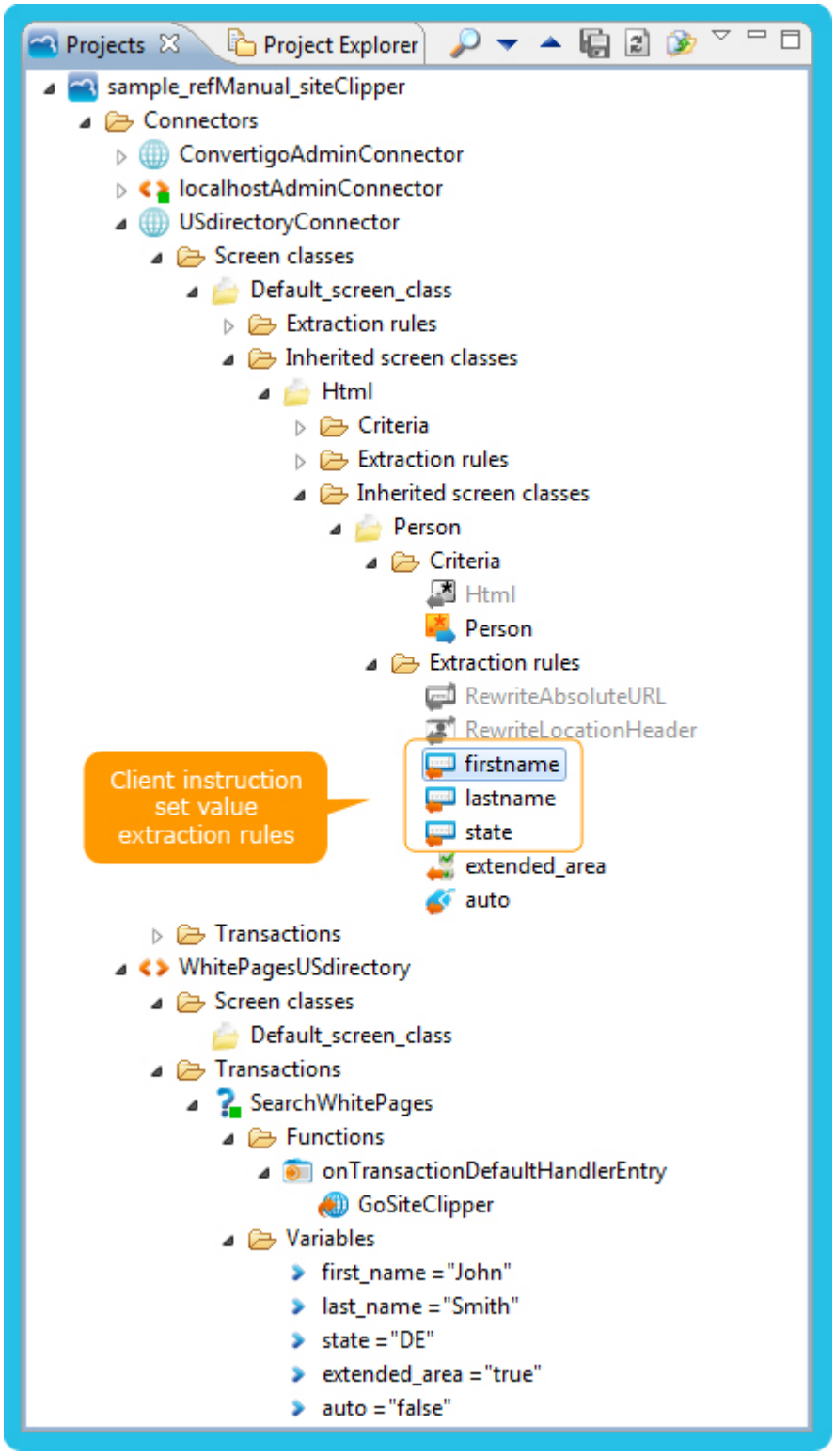


Figure 2 - 691: Client instruction set value extraction rule - Objects in Projects view

Switch to a browser displaying the test platform of this project. Executing the `SearchWhitePages` transaction (in a new tab thanks to **Execute full screen** button) with custom values for defined variables:

- reaches the form page,
- gives back the control on the page through the Site Clipper,

- and automatically fills the input fields and selects value in combo box using variables custom values:

Figure 2 - 692: Client instruction set value extraction rule - Automatically filled inputs and combo box

CLIENT INSTRUCTION SET CHECKED 

OBJECT DESCRIPTION

Automatically checks a target element from a web page accessed through Convertigo Site Clipper. The action is performed in the client browser after the page is loaded.

The *Client instruction set checked* extraction rule stores an instruction of checking/unchecking an element in a queue of client instructions. This queue is then unstacked when the page is client-side loaded. The instruction of checking/unchecking an element is performed on the page when it is loaded by the client browser.

The target element (checkbox or radio button) is defined thanks to the **JQuery selector** property and the state to which change the element is defined by the **Checked state** property. These properties are JavaScript expressions, evaluated by Convertigo when the extraction rule is applied.

In order to process the unstacking of client instructions in the web page, `client instruction engine` code is injected into the page after all response extraction rules are applied. When the page is client-side loaded, the `client instruction engine` runs and consumes each registered client instruction, in the same order as extraction rules.

Consuming the *Client instruction set checked* instruction, the engine selects the target element using the **JQuery selector** and depending on the **Checked state** property value:

- sets the attribute `checked="checked"` on the target element, if the **Checked state** evaluation is `true`,
- removes the `checked` attribute from the target element otherwise.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Checked state	JS expression	configuration	JavaScript expression defining the boolean value used to change the target element state. This property defines the checked state to set on the target element. It is defined thanks to a JavaScript expression evaluated using the JavaScript scope of the current context. It should return a boolean to be used as follows: • `true` for checked state, • `false` for unchecked state. The target element should be an `INPUT` element of `checkbox` or `radio` type.
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
JQuery selector	JS expression	configuration	JavaScript expression defining a JQuery selector matching the target element from the HTML document. This property defines a JQuery selector used to retrieve the target element (input field, text area, select, checkbox, radio button, etc.) on the client browser when the page is loaded. It is defined thanks to a JavaScript expression evaluated using the JavaScript scope of the current context. It should return a string to be used as a JQuery selector. Here are some default syntaxes for JQuery selectors. An id selector starts with # character, an attribute selector is between `[]` characters, a class selector starts with `.` character. The > character separating several selectors defines a constraint to the direct ancestor. The space character separating several selectors defines a constraint to any ancestor. The followings are some simple selectors that can be used in JQuery: • Select an element by id: `#the_id_to_search`, • Select an any type of element by name: `[name="the_name_to_search"]` or also `*[name="the_name_to_search"]`, • Select an input element by name: `input[name="the_name_of_the_input_to_search"]`, • Select an any type of element by class: `.class_to_search`, • Select a div element by class: `div.class_of_a_div_to_search`, • Select an input by name, direct child of a form element selected by class: `form.form_class > input[name="input_name"]`, • Select an hidden input, descendant of a specified id: `#specified_id input[type="hidden"]`. Note: The full JQuery selector documentation is available on the official JQuery website: `http://api.jquery.com/category/selectors/`.

EXAMPLES

Let's consider the US directory White Pages website, US directory pages to search for people. This search form needs a last name and may need a first name, a city, a state and some other optional fields:

Figure 2 - 693: Client instruction set checked extraction rule - Whites pages on US directory website

An *HTML transaction*, named `SearchWhitePages`, is created to navigate in the US directory website to access the White Pages search page. This transaction gives the control back to the user on this form page through a Site Clipper connector, thanks to a *Continue with Site Clipper* statement.

This transaction defines several variables, such as `first_name`, `last_name` and `state`, corresponding to the interesting fields of the previous page's form, including one variable named `extended_area`. When invoked, the HTML transaction creates a new Convertigo context and sets its variables values into its JavaScript scope.

When the user gets the control back through the Site Clipper connector, we want the search form to be prefilled with the values declared in the previous HTML transaction variables, and the "**Include surrounding area**" checkbox to be checked (or not) depending on the `extended_area` variable value.

In the *Site Clipper connector*, named `USdirectory`, a screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A `Person` screen class is defined to match on the previous web page, it contains several extraction rules to fill all form inputs.

To check/uncheck the "**Include surrounding area**" checkbox of the form, a *Client instruction set checked* rule is created on the `Person` screen class with the following parameters:

```
Client instruction set checked [
   jQuery selector="input[name='metro_area']"
   checked state= extended_area=="true"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

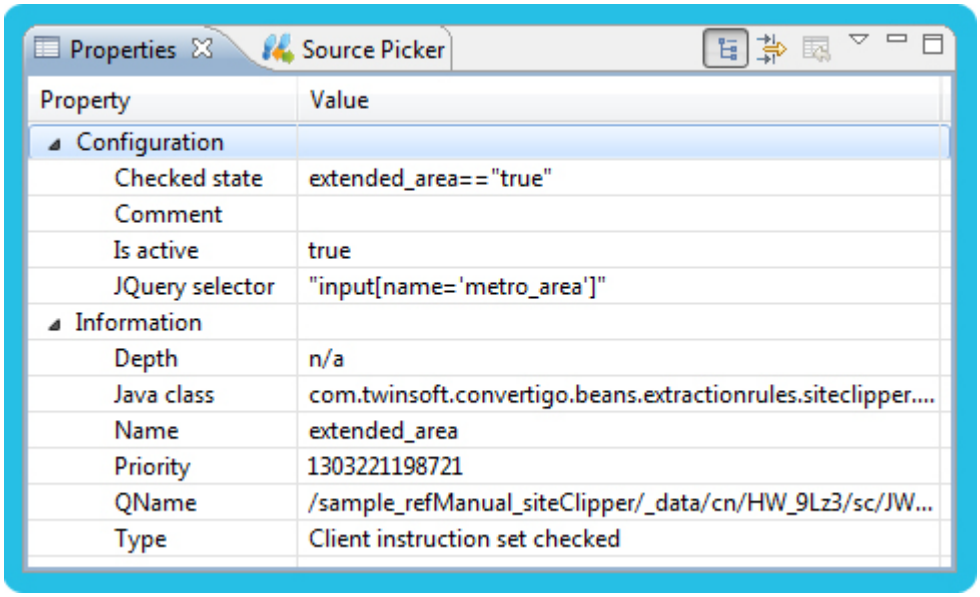


Figure 2 - 694: Client instruction set checked extraction rule - Configuration example

The **JQuery selector** is defined to select an `input` element by its `name` attribute. The **Checked state** property is set to a JavaScript expression testing the value of the `extended_area` variable available in the Site Clipper context JavaScript scope because it was set in the scope by the HTML transaction which received it as input.

The extraction rule appears as follows in the **Projects** view:

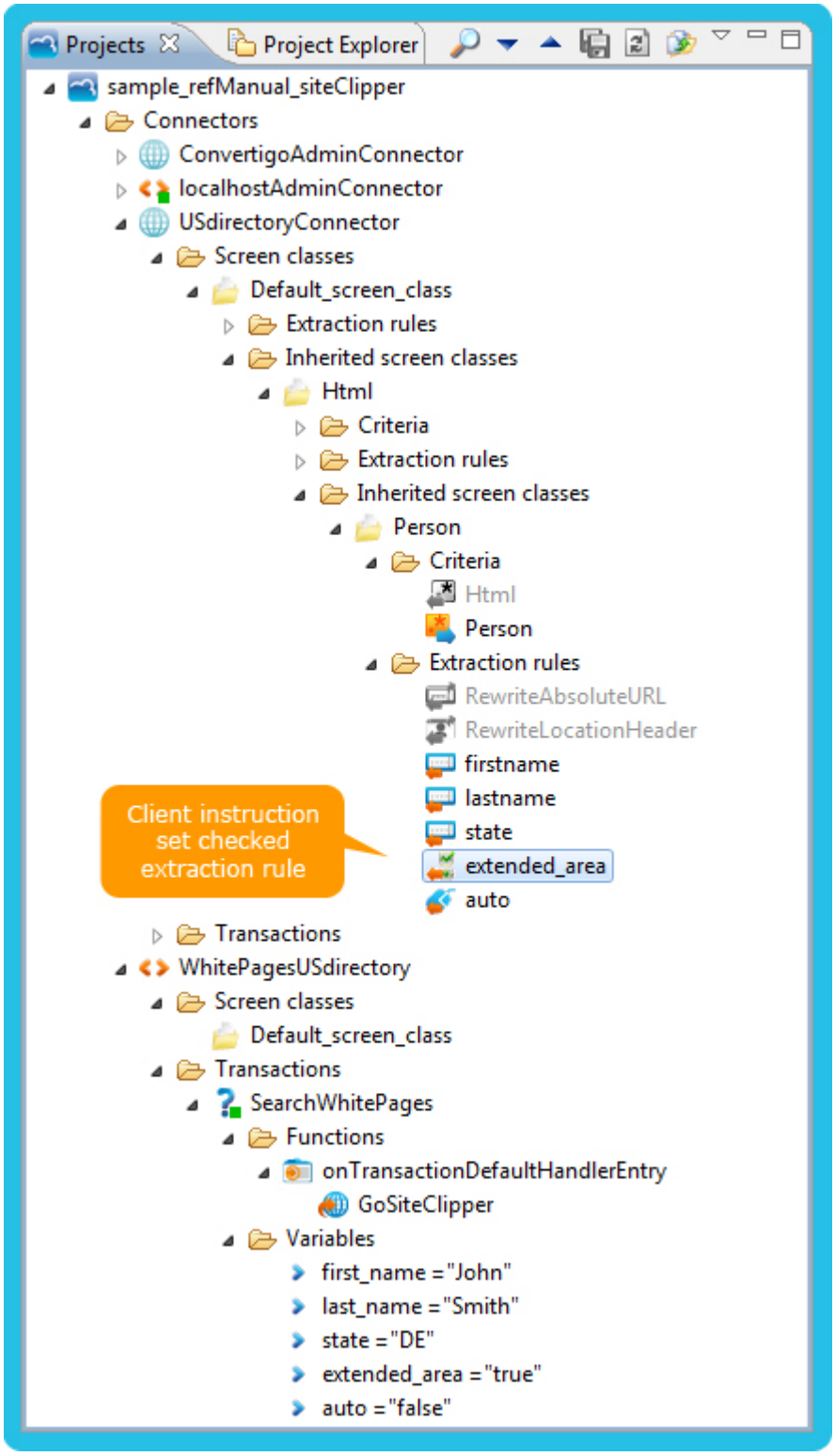


Figure 2 - 695: Client instruction set checked extraction rule - Object in Projects view

Switch to a browser displaying the test platform of this project. Executing the `SearchWhitePages` transaction (in a new tab thanks to **Execute full screen** button) with custom values for defined variables:

- reaches the form page,
- gives back the control on the page through the Site Clipper,

- automatically fills the input fields using variables custom values,
- and checks or unchecks the checkbox using `extended_area` variable custom value.

With the `extended_area` variable value set to `"true"`, the checkbox is automatically checked:

Figure 2 - 696: Client instruction set checked extraction rule - Automatically checked checkbox

CLIENT INSTRUCTION CLICK 

OBJECT DESCRIPTION

Automatically clicks on a target element from a web page accessed through Convertigo Site Clipper. The action is performed in the client browser after the page is loaded.

The *Client instruction click* extraction rule stores an instruction of clicking on an element in a queue of client instructions. This queue is then unstacked when the page is client-side loaded. The instruction of clicking on an element is performed on the page when it is loaded by the client browser.

The target element is defined thanks to the **JQuery selector** property. This property is a JavaScript expression, evaluated by Convertigo when the extraction rule is applied.

In order to process the unstacking of client instructions in the web page, `client instruction engine` code is injected into the page after all response extraction rules are applied. When the page is client-side loaded, the `client instruction engine` runs and consumes each registered client instruction, in the same order as extraction rules.

Consuming the *Client instruction click* instruction, the engine selects the target element using the **JQuery selector** and clicks on it.

Note: The click is perfomed using the JQuery `click` function. For more information about this function, see the following page: `http://api.jquery.com/click/`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

Property	Type	Category	Description
JQuery selector	JS expression	configuration	JavaScript expression defining a JQuery selector matching the target element from the HTML document. This property defines a JQuery selector used to retrieve the target element (input field, text area, select, checkbox, radio button, etc.) on the client browser when the page is loaded. It is defined thanks to a JavaScript expression evaluated using the JavaScript scope of the current context. It should return a string to be used as a JQuery selector. Here are some default syntaxes for JQuery selectors. An id selector starts with # character, an attribute selector is between `[]` characters, a class selector starts with `.` character. The > character separating several selectors defines a constraint to the direct ancestor. The space character separating several selectors defines a constraint to any ancestor. The followings are some simple selectors that can be used in JQuery: • Select an element by id: `#the_id_to_search`, • Select an any type of element by name: `[name="the_name_to_search"]` or also `*[name="the_name_to_search"]`, • Select an input element by name: `input[name="the_name_of_the_input_to_search"]`, • Select an any type of element by class: `.class_to_search`, • Select a div element by class: `div.class_of_a_div_to_search`, • Select an input by name, direct child of a form element selected by class: `form.form_class > input[name="input_name"]`, • Select an hidden input, descendant of a specified id: `#specified_id input[type="hidden"]`. Note: The full JQuery selector documentation is available on the official JQuery website: `http://api.jquery.com/category/selectors/`.

EXAMPLES

Let's consider the US directory White Pages website, US directory pages to search for people. This search form needs a last name and may need a first name, a city, a state and some other optional fields:

Figure 2 - 697: Client instruction click extraction rule - Whites pages on US directory website

An *HTML transaction*, named `SearchWhitePages`, is created to navigate in the US directory website to access the White Pages search page. This transaction gives the control back to the user through a Site Clipper connector, thanks to a *Continue with Site Clipper* statement.

This transaction defines several variables, such as `first_name`, `last_name` and `state`, corresponding to the interesting fields of the previous page's form, and including one variable named `auto`. When invoked, the HTML transaction creates a new Convertigo context and sets its variables values into its JavaScript scope.

When the user gets the control back through the Site Clipper connector, we want the search form to be prefilled with the values declared in the previous HTML transaction variables, and possibly automatically validated, depending on the `auto` variable value.

In the *Site Clipper connector*, named `USdirectory`, a screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. A `Person` screen class is defined to match on the previous web page, it contains several extraction rules to fill all form inputs.

To validate the form, a *Client instruction click* rule is created on the `Person` screen class with the following parameters:

```
Client instruction click [
   jQuery selector=
   (auto=="true")?"input.button[value='Search']":"void"
]
```

These parameters are edited in the **Properties** view of the Convertigo Studio:

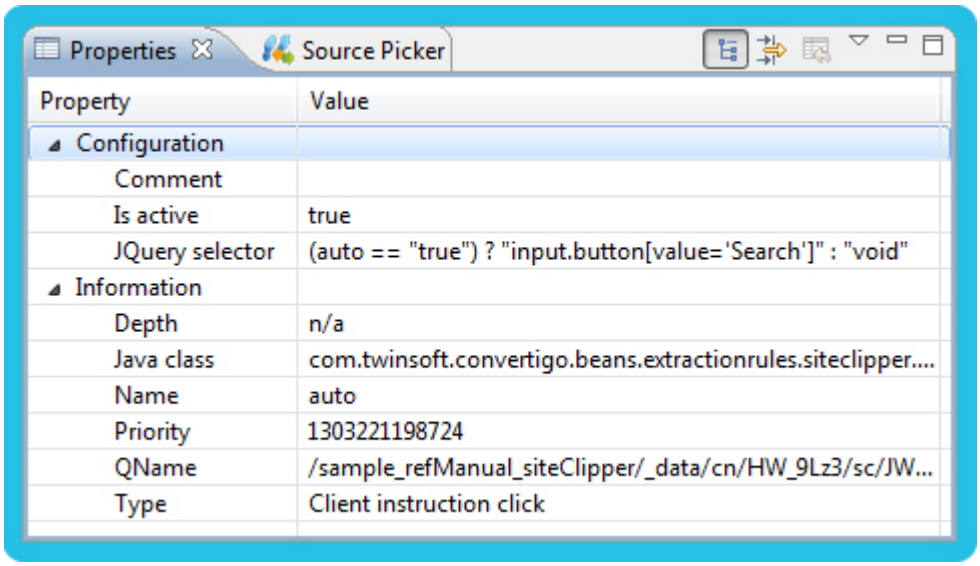


Figure 2 - 698: Client instruction click extraction rule - Configuration example

The **JQuery selector** is defined thanks to a JavaScript expression testing the value of the `auto` variable, to select an `input button` element by its `name` attribute when the `auto` variable is `"true"`, or nothing (`"void"`) otherwise. The `auto` variable is available in the Site Clipper context JavaScript scope because it was set in the scope by the HTML transaction which received it as input.

The extraction rule appears as follows in the **Projects** view:

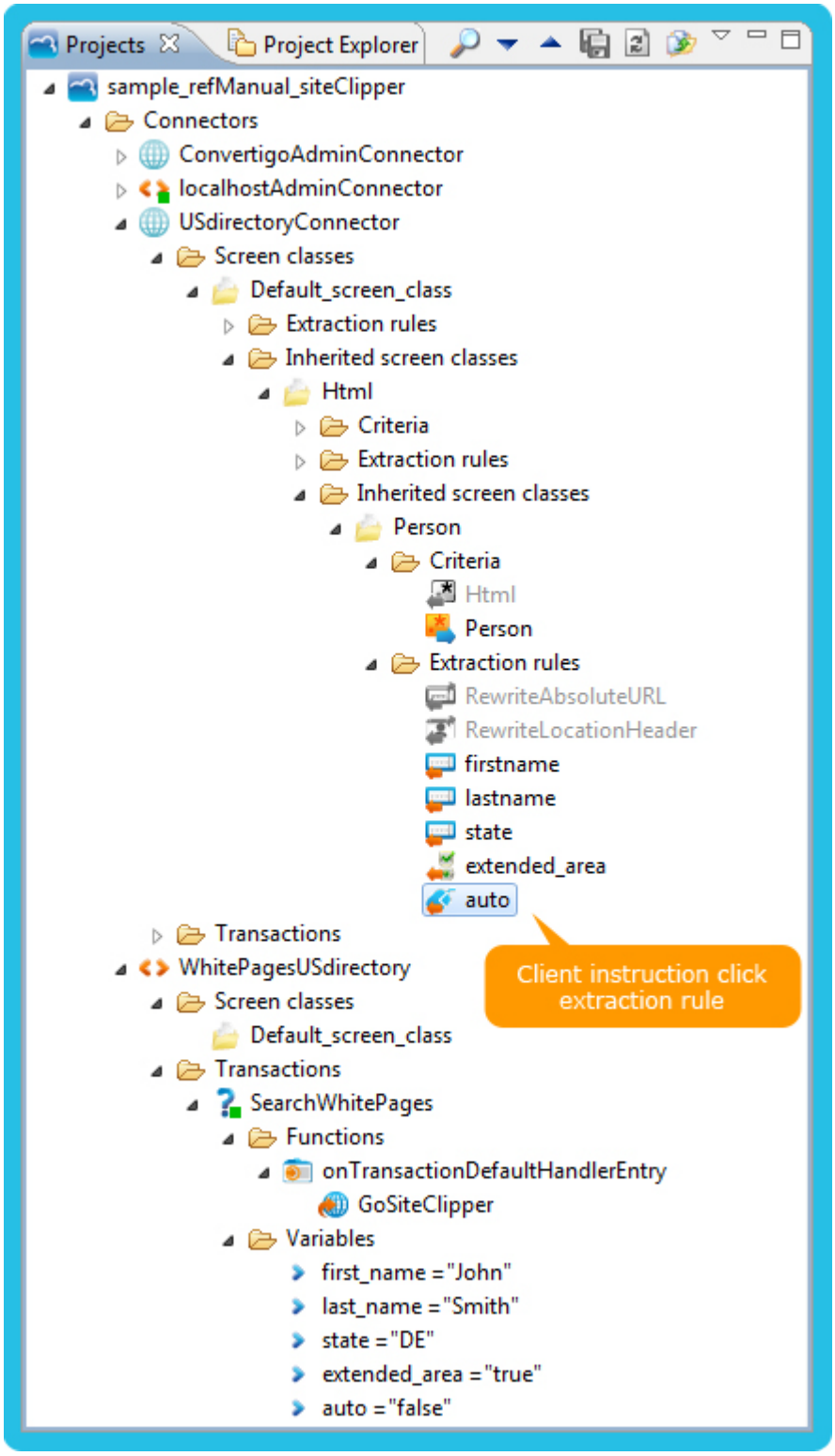


Figure 2 - 699: Client instruction click extraction rule - Object in Projects view

Switch to a browser displaying the test platform of this project. Executing the `SearchWhitePages` transaction (in a new tab thanks to **Execute full screen** button) with custom values for defined variables:

- reaches the form page,
- gives back the control on the page through the Site Clipper,

- automatically fills the input fields and checks the checkbox using variables custom values,
- and validates the form by clicking on the **Search** button or not using `auto` variable custom value.

With the `auto` variable value set to `"true"`, the form is automatically validated:

Figure 2 - 700: Client instruction click extraction rule - Automatically validated form

REMOVE RESPONSE CACHE HEADERS 

OBJECT DESCRIPTION

Removes HTTP cache-related headers from a Site Clipper response.

The *Remove request cache headers* extraction rule removes existing cache-related headers from a response. The following headers are removed:

- `Last-Modified`,
- `Cache-Control`,
- `ETag`.

OBJECT PROPERTIES

The table below describes the object properties:

Property	Type	Category	Description
Comment	String	configuration	Describes the object comment to include in the documentation report. This property generally contains an explanation about the object.
Is active	boolean	configuration	Defines whether the extraction rule is active.

EXAMPLES

Let's consider the Convertigo website.

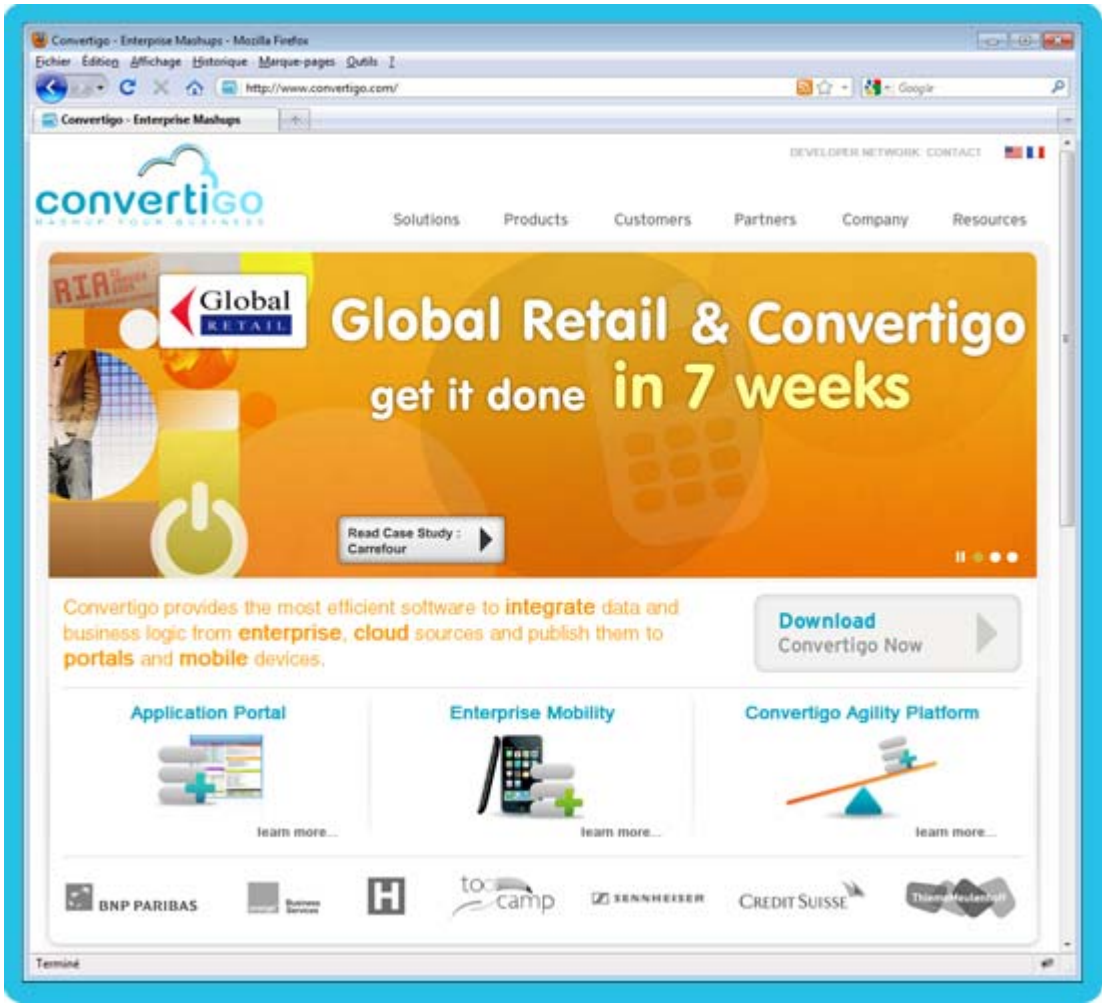


Figure 2 - 701: Remove response cache headers extraction rule - Convertigo website

Before proceeding with the example, let's activate the Firebug extension of Firefox browser. Observing the response HTTP headers of the `convertigo_home_banner.swf` resource, i.e. the Flash animated banner of the welcome page, we can see the cache-related headers named `Last-Modified` and `Cache-Control` are present:

Figure 2 - 702: Remove response cache headers extraction rule - Cache-related headers present on Flash resource HTTP response

In the context of a *Site Clipper connector*, we would like to clip the entire site and dynamically remove the cache-related headers of Flash resources accessed through Convertigo.

A transaction, named `GoTo_Convertigo`, is defined as default transaction for the *Site Clipper connector* named `ConvertigoWebSiteConnector`. It defines the URL `http://www.convertigo.com` as target URL to connect to Convertigo website.

A screen classes hierarchy is defined thanks to criteria and contains default extraction rules to identify and handle accessed data and resources. Two screen classes are defined in order to handle Convertigo website pages and more precisely HTML resources from this website.

Finnaly, a screen class, named `FlashResources`, is defined thanks to a response *MIME type*

criterion to handle shockwave-flash resources. On this screen class, an *Add response header* extraction rule is added in order to add a new response header named `convertigoFlash` to be sure the screen class has been detected.

On the `FlashResources` screen class, a *Remove response cache headers* extraction rule is added in order to remove the cache-related response headers. The rule has no parameters to configure:

```
Remove response cache headers [
]
```

This extraction rule appears as follows in the **Properties** view of the Convertigo Studio:

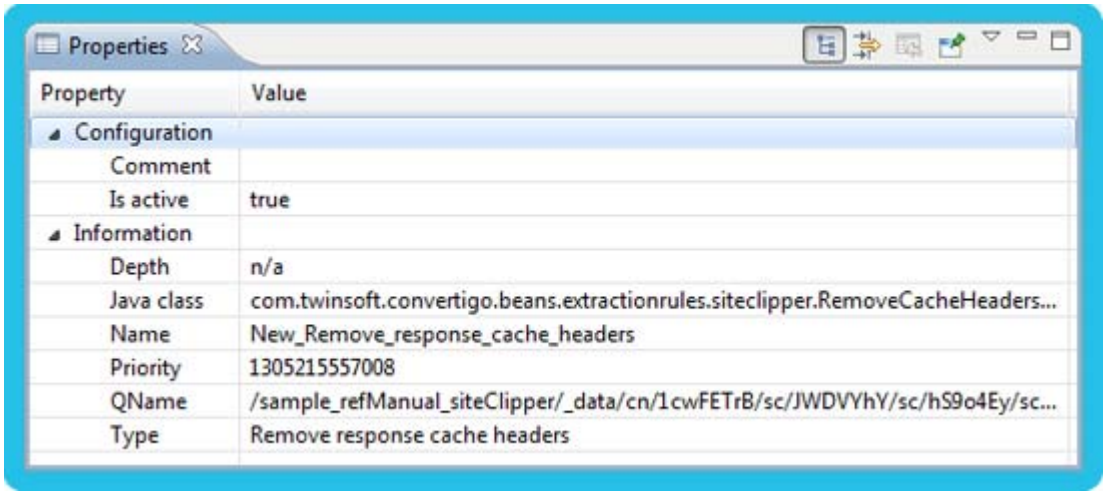


Figure 2 - 703: Remove response cache headers extraction rule - Configuration example

The extraction rules appear as follows in the **Projects** view:

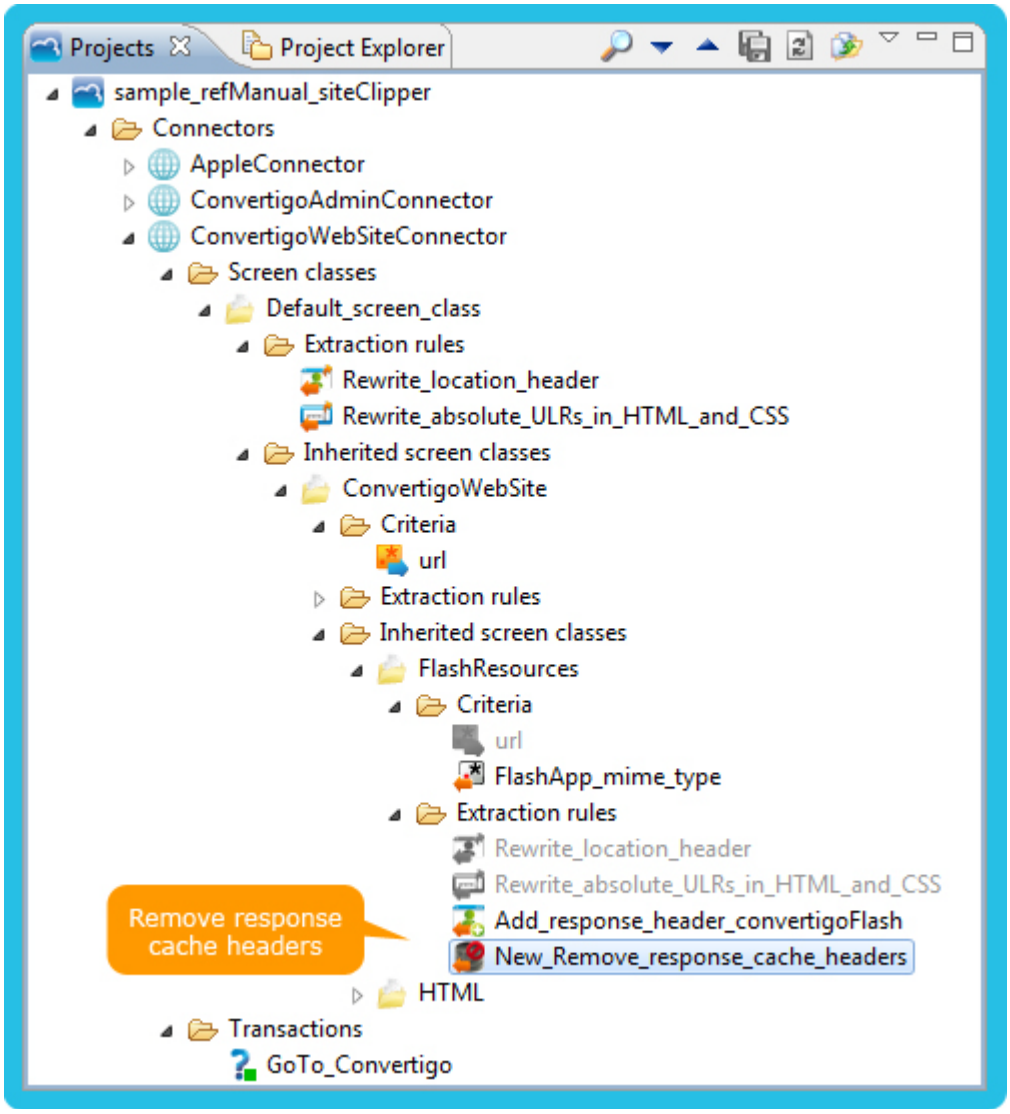


Figure 2 - 704: Remove response cache headers extraction rule - Object in Projects view

Switch back to Firefox browser, display the test platform of this project with the Firebug extension activated. Executing the `GoTo_Convertigo` transaction (in a new tab thanks to **Execute full screen** button) reaches the Convertigo website welcome page.

In Firebug, we can see that the response HTTP header named `convertigoFlash` is added to the Flash resource, meaning that the `FlashResources` screen class has matched. The `Cache-Control` and `Last-Modified` headers are not appearing anymore, they were removed by the *Remove response cache headers* extraction rule:

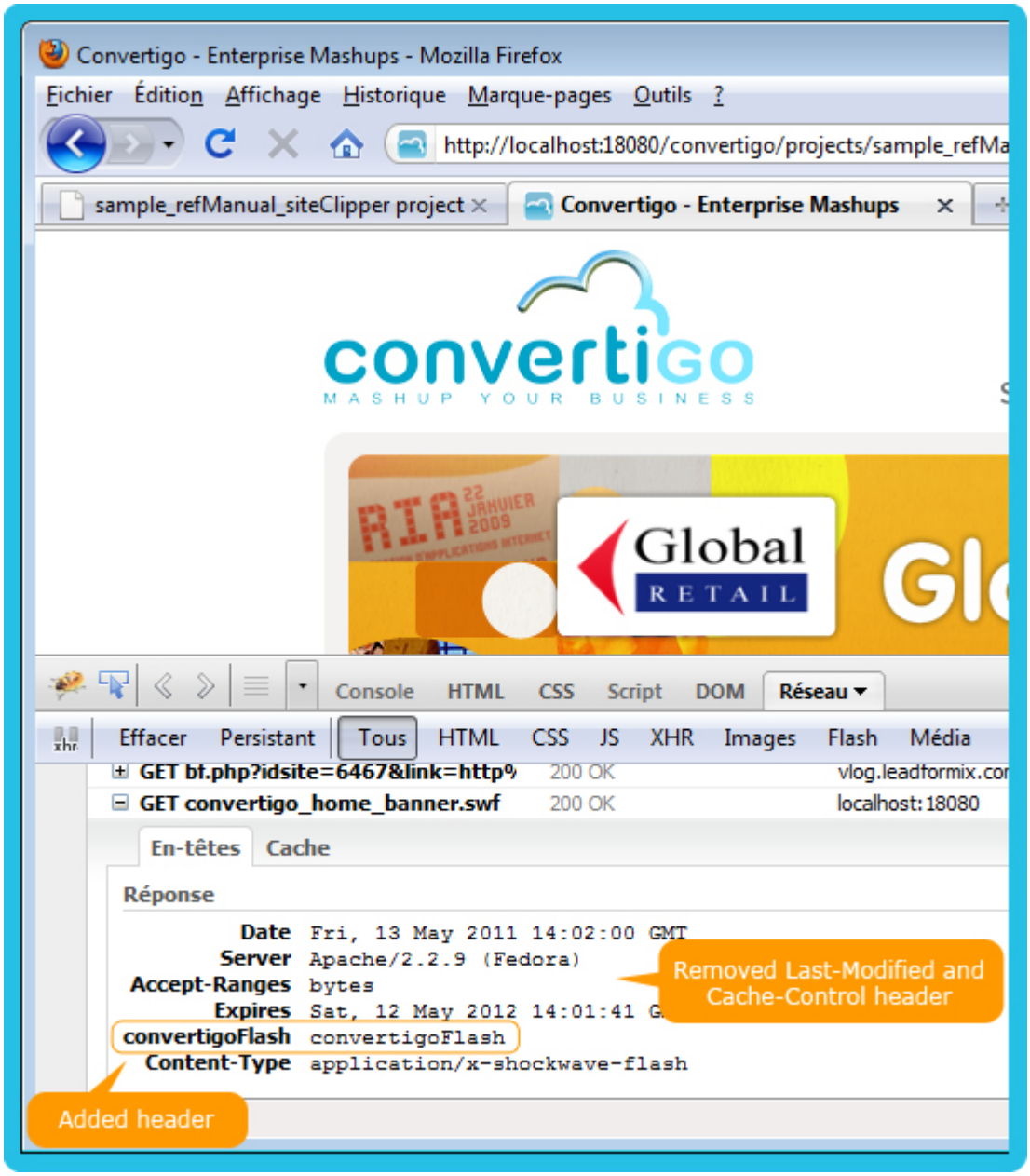


Figure 2 - 705: Remove response cache headers extraction rule - Added and removed headers on Convertigo Flash banner resource

3 JavaScript Objects APIs

This chapter offers information about JavaScript objects available in Convertigo transactions or sequences.

3.1 Javelin object javadoc

Javelin is the object representing the legacy screen. Actions can be defined and performed on the legacy screen by programming code using the *Javelin* object. This section contains the API documentation of the *Javelin* object.

3.1.1 Fields detailed list

This section presents the list of fields contained in *Javelin* object, with their description.

Table 3 - 1: Fields list

Type	Field name	Description
static char	***AT_AUTO_ENTER***	Mask for DRCS (downloadable font) attribute bit: 0x8000 (0100 0000 0000 0000). Note: relevant for Videotex only. For 3270 and 5250 this bit indicates an AUTO_TAB flag for the field.
static char	***AT_AUTO_TAB***	Mask for double height: 0x0040 (0000 0000 0100 0000). For 32370/5250 also indicates field is AUTO_ENTER.
static char	***AT_BLINK***	Mask for blink attribute bit: 0x1000 (0001 0000 0000 0000).
static char	***AT_BOLD***	Mask for bold attribute bit: 0x0800 (0000 0100 0000 0000).
static int	***AT_COLOR_BLACK***	
static int	***AT_COLOR_BLUE***	
static int	***AT_COLOR_CYAN***	
static int	***AT_COLOR_GREEN***	
static int	***AT_COLOR_MAGENTA***	
static int	***AT_COLOR_RED***	
static int	***AT_COLOR_WHITE***	
static int	***AT_COLOR_YELLOW***	
static char	***AT_DOUBLEHEIGHT***	Mask for double height: 0x0040 (0000 0000 0100 0000). For 32370/5250 also indicate field is AUTO_ENTER.
static char	***AT_DOUBLEWIDTH***	Mask for double width attribute bit: 0x0080 (0000 0000 1000 0000).
static char	***AT_DRCS***	Mask for DRCS (downloadable font) attribute bit: 0x8000 (0100 0000 0000 0000). Note: relevant for Videotex only. For 3270 and 5250 this bit indicates an AUTO_TAB flag for the field.
static int	***AT_FIELD_ATTRIBUTE***	
static int	***AT_FIELD_HIDDEN***	Mask for hidden field attribute: 0x0C0000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.
static int	***AT_FIELD_HIGH_INTENSITY***	Mask for high intensity field attribute: 0x040000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.

Table 3 - 1: Fields list (...)

Type	Field name	Description
static int	*AT_FIELD_MODIFIED*	Mask for modified field attribute: 0x010000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.
static int	*AT_FIELD_NUMERIC*	Mask for numeric field attribute: 0x100000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.
static int	*AT_FIELD_PEN_SELECTABLE*	Mask for pen selectable field attribute: 0x080000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.
static int	*AT_FIELD_PROTECTED*	Mask for protected Field attribute bit: 0x200000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.
static int	*AT_FIELD_RESERVED*	Mask for reserved field attribute: 0x020000. Note: relevant for IBM 3270 / 5250 and BULL DKU7xxx only.
static char	*AT_HIDDEN*	Mask for hidden attribute bit: 0x8000 (1000 0000 0000 0000).
static char	*AT_INK*	Mask for INK attribute: 0x0007 (0000 0000 0000 0111).
static char	*AT_INVERT*	Mask for inverse attribute bit: 0x0200 (0000 0001 0000 0000).
static char	*AT_MASKED*	Mask for masked attribute bit: 0x2000 (0010 0000 0000 0000). Note: relevant for VT220 only.
static char	*AT_NPTUI*	Mask for NPTUI attribute bit: 0x2000 (0010 0000 0000 0000). Note: relevant for 5250 only.
static char	*AT_PAPER*	Mask for PAPER attribute: 0x0038 (0000 0000 0011 1000).
static char	*AT_PROTECTED*	Mask for protected attribute bit: 0x4000 (0100 0000 0000 0000). For 5250 reprsent also the NPTUI attribute.
static char	*AT_UNDERLINE*	Mask for underline attribute bit: 0x0100 (0000 0001 0000 0000).
static char	*AT_UPHALF*	Mask for double height upper row attribute bit: 0x0400 (0000 0100 0000 0000). Note: relevant for VT220 only.
static java.lang.String	*AS400*	Defines an IBM 5250 (AS/400) session.
static java.lang.String	*DKU*	Defines a Bull DKU session.
static java.lang.String	*SNA*	Defines an IBM 3270 (SNA) session.
static java.lang.String	*TN5250*	Defines a Twinsoft 5250 (AS/400) session.
static java.lang.String	*VDX*	Defines a videotex session.
static java.lang.String	*VT*	Defines a VT220 session.

3.1.2 Methods detailed list

This section presents the list of methods contained in *Javelin* object, with their description and parameters.

Table 3 - 2: Methods list

Return Type	Method signature	Description
void	***addKeyListener***(java.awt.event.KeyListener l) **Parameters:** `l` - the component interested by the events.	Registers a client on the `keyListener`. `KeyListener` delivers events for keystokes on *Javelin*.
void	***addZoneListener***(com.twinsoft.twinj.zoneListener l) **Parameters:** `l` - the component interested by the events.	Registers a client on the `zoneListener`. `ZoneListener` delivers events for selection changes.
void	***clearTrigger***()	Remove all arrived trigger from the `MsgQueue`.
void	***connect***() **See also:** connect(int), disconnect()	Connects *Javelin* to a server. The connection is asynchronous. When the connection is made, the `Connected` event is thrown by *Javelin*. The connection is made with the parameters provided by helper functions such as `setDteAddress()`, `setCommDevice()`, `setServiceCode()`...
boolean	***connect***(int timeout) **Parameters:** `timeout` - the maximum amount of milliseconds during which the connection is expected. **Returns:** `true` if connected, `false` if the timeout expired. **See also:** connect(), disconnect()	Connects synchronously *Javelin* to a server. The connection is made with the parameters provided by helper functions such as `setDteAddress()`, `setCommDevice()`, `setServiceCode()`...
void	***deleteWaitAts***() **See also:** waitAt(String, int, int, long), waitAtId(int, String, int, int), waitForId(int, String), waitFor(String, long), deleteWaitFors()	Deletes all triggers set by the `waitAt()` method.
void	***deleteWaitFors***() **See also:** waitAt(String, int, int, long), waitAtId(int, String, int, int), waitForId(int, String), waitFor(String, long), deleteWaitAts()	Deletes all triggers set by the `waitFor()` method.
void	***disconnect***() **See also:** connect(), connect(int)	Disconnects *Javelin* from the current connected server. The disconnection is synchronous, i.e. *Javelin* is disconnected when the method returns.
void	***doAction***(java.lang.String action) **Parameters:** `action` - the action to execute, see Appendix "*Legacy emulator actions table*".	Executes an action on the emulator.
char	***getChar***(int column, int line) **Parameters:** `column` - the horizontal coordinate from left to right beginning by 0. `line` - the vertical coordinate from top to bottom beginning by 0. **Returns:** the read character at (`column`, `line`) coordinates. **See also:** getCharAttribute(int, int), getString(int, int, int), setChar(int, int, char)	Returns the ASCII code of the character at coordinates (`column`, `line`).

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
int	***getCharAttribute***(int column, int line) **Parameters:** `column` - the horizontal coordinate from left to right beginning by 0. `line` - the vertical coordinate from top to bottom beginning by 0. **Returns:** the character attribute at (`column`, `line`) coordinates. **See also:** getChar(int, int), getString(int, int, int), setChar(int, int, char)	Returns the character attribute at coordinates (`column`, `line`). You can access specific attribute bit with `AT_xxx` masks.
int	***getCurrentColumn***() **Returns:** the current column. **See also:** getCurrentLine()	Returns the current column of the caret. Columns are designed from top to bottom and beginning by 0.
int	***getCurrentLine***() **Returns:** the current line. **See also:** getCurrentColumn()	Returns the current line of the caret. Lines are designed from left to right and beginning by 0.
boolean	***getDataStableOnCursorOn***() **Returns:** the DataStableOnCursorOn.	Retrieves the `DataStableOnCursorOn` flag. This flag controls how the data stable condition is detected. For Videotex emulators. if `true`, this condition is met when a Videotex cursor is shown by the host.
int	***getDataStableThreshold***() **Returns:** the dataStableThreshold value.	Gets the `dataStableThreshold` value.
int	***getFieldAttribute***(int fieldIndex) **Parameters:** `fieldIndex` - the index or the field from 0 to *n*. **Returns:** the attribute of the field.	Gets the attribute of a given field.
int	***getFieldColumn***(int fieldIndex) **Parameters:** `fieldIndex` - the index or the field from 0 to *n*. **Returns:** the column of the field.	Gets the column of a given field.
int	***getFieldLength***(int fieldIndex) **Parameters:** `fieldIndex` - the index or the field from 0 to *n*. **Returns:** the length of the field.	Gets the lenght of a given field. If the lenght is greater than the width of the screen, it means that the field spans on several lines.
int	***getFieldLine***(int fieldIndex) **Parameters:** `fieldIndex` - the index or the field from 0 to *n*. **Returns:** the line of the field.	Gets the line of a given field.
java.lang.String	***getFieldText***(int fieldIndex) **Parameters:** `fieldIndex` - the index or the field from 0 to *n*. **Returns:** the text of the field.	Gets the text of a given field.
int	***getNumberOfFields***() **Returns:** the number of fields.	Gets the number of fields on the screen. If the number of fields is 0, then the screen is called "unformatted". This is always the case for VT and VDX emulators but can be the case for IBM in the case of a unformatted screen.

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
int	***getScreenHeight***() **Returns:** the number of lines. **See also:** getScreenWidth()	Gets the number of lines of the current emulator screen.
int	***getScreenWidth***() **Returns:** the number of columns. **See also:** getScreenHeight()	Gets the number of columns of the current emulator screen.
java.awt.Rectangle	***getSelectionZone***() **Returns:** zone - the rectangle that holds the zone (x = column 0 based, y = line 0 based, width = width of the selection in chars, height = height of the selection in chars)	Gets the current selection zone on *Javelin*.
java.lang.String	***getString***(int column, int line, int length) **Parameters:** `column` - the horizontal coordinate from left to right beginning by 0. `line` - the vertical coordinate from top to bottom beginning by 0. `lenght` - the string length to return. **Returns:** the read string at (`column`, `line`) coordinates. **See also:** getChar(int, int), getCharAttribute(int, int), setChar(int, int, char)	Returns the string at coordinates (`column`, `line`).
java.lang.String	***getTerminalClass***() **Returns:** the terminal class.	Retrieves the terminal class of the *Javelin* object.
boolean	***isConnected***() **Returns:** `true` if connected, `false` otherwise. **See also:** connect(), connect(int), disconnect()	Retrieves the *Javelin* connection state.
void	***moveCursor***(int column, int line) **Parameters:** `column` - the horizontal coordinate from left to right beginning by 0. `line` - the vertical coordinate from top to bottom beginning by 0.	Moves the cursor to coordinates (`column`, `line`).
boolean	***printBuffer***(java.lang.String printerPort, java.lang.String buffer) **Parameters:** `printerPort` - the printer port. `buffer` - the buffer to be printed. **Returns:** `true` if ok, `false` otherwise.	Writes a string on a given printer port. This port can be any string containing LPT**x** or a network printer specification as `\\SERVER\PRINTER` or a file as `print.txt`. The specified port will be opened, then the buffer will be written to it and the port will be closed in this sequence. If the port is `null`, then the port will be the default port specified as RawPrinter in the applet.
void	***removeAllZoneListeners***()	Unregisters all clients on the `zoneListener`. `ZoneListener` delivers events for selection changes.
void	***removeKeyListener***(java.awt.event.KeyListener l) **Parameters:** `l` - the component not anymore interested by the events.	Unregisters a client on the `keyListener`. `KeyListener` delivers events for keystokes on *Javelin*.

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
void	***removeZoneListener***(com.twinsoft.twinj.zoneListener l) **Parameters:** `l` - the component not anymore interested by the events.	Unregisters a client on the `zoneListener`. `ZoneListener` delivers events for selection changes.
void	***send***(java.lang.String keystrokes) **Parameters:** `keystrokes` - the string to send.	Sends a string as if the user has stroken it on keyboard. All characters between `0x00` and `0xFF` can be sent by this way.
void	***setAutoConnect***(boolean bAutoConnect) **Parameters:** `bAutoConnect` - indicates if automatic connection should be performed without having to call one of the `connect()` methods.	Sets the auto connect flag.
void	***setChar***(int column, int line, char c) **Parameters:** `column` - the horizontal coordinate from left to right beginning by 0. `line` - the vertical coordinate from top to bottom beginning by 0. c - the ASCII character code to set. **See also:** getChar(int, int), getCharAttribute(int, int), getString(int, int, int),	Sets the ASCII code of the character at coordinates (`column`, `line`).
void	***setClientConfig***(java.util.Properties p) **Parameters:** p - the properties ti set in the emulator.	Sets the current terminal properties. The properties depends on the emulator technology. Only the 3270 and 5250 emulators support this api.
void	***setClientConfig***(java.lang.String configuration) **Parameters:** `configuration` - the new configuration; this is a string of the form `param1=value1&param2=value2&; ... &paramN=valueN`. *NB:* each `valuei` should be in UTF8 format.	Sets a new configuration for *Javelin*.
void	***setCommDevice***(java.lang.String commDevice) **Parameters:** `commDevice` - the communication device.	Sets the communication device, i.e. the address (IP or DNS) of the PAVI gateway, eventually followed by `:port`.
void	***setDataStableOnCursorOn***(boolean bool) **Parameters:** `bool` - the DataStableOnCursorOn flag.	Sets the `DataStableOnCursorOn` flag.
void	***setDataStableThreshold***(int val) **Parameters:** `val` - the dataStableThreshold value.	Sets the `dataStableThreshold` value. This value is the maximum time allowed with no data received from the host to consider a dataStable condition.
void	***setDoCapture***(boolean bDoCapture) **Parameters:** `bDoCapture` - indicates if capture should be written or not.	Sets the capture flag.

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
void	***setDteAddress***(java.lang.String connectionString) **Parameters:** `connectionString` - the connection string using the following format: `<path>#<destination>-<source sub address>.<destination sub address>/<max level>`.	Sets the connection string. This method is ineffective for 5250 and 3270 emulators.
	For a *direct telnet connection* (mandatory for IBM and Bull emulators): • `path` = DIR • `destination` = <IP address>:<port> To connect to a *PAVI + Eicon X25 board* (there is no "`-`" character between `destination` and `sourceSA` in this case only): • `path` = EIC • `destination` = <Iremote X25 address> • `source sub address` = called sub-address (for Intelmatique billings) • `destination sub address` = calling sub-address (for services filter selection) • `max level` = 0..255 To connect to a *PAVI + SAP/IP*: • `path` = TCP • `destination` = <Intelmatique address>@port • `source sub address` = called sub-address (for Intelmatique billings) • `destination sub address` = calling sub-address (for services filter selection) • `max level` = 0..255 To connect to a *PAVI + iMinitel* (same as DIR but using the PAVI as a gateway ; iMinitel address: `172.31.0.20@7516`): • `path` = MIP • `destination` = <iMinitel address>@port • `max level` = 0..255 To connect to a *PAVI + iMinitel + RAS* (using a modem or ISDN service ; iMinitel address: `172.31.0.20@7516`): • `path` = RAS • `destination` = <iMinitel address>@port • `max level` = 0..255	
void	***setGroup***(java.lang.String group) **Parameters:** `group` - the user group.	Sets the group of the user.
void	***setLogOutputStream***(java.io.OutputStream outputStream) **Parameters:** `outputStream` - the output stream for traces.	Defines the output stream for traces.
void	***setSelectionZone***(java.awt.Rectangle zone) **Parameters:** `zone` - the rectangle that holds the zone (x = column 0 based, y = line 0 based, width = width of the selection in chars, height = height of the selection in chars)	Sets a selection zone on *Javelin*. This will result as if the user had selected the zone with the mouse. After this api is called, the user will be able to modify the selection zone.
void	***setServiceCode***(java.lang.String serviceCode) **Parameters:** `serviceCode` - the service code using the following format: • VT: n/a, • Bull: mailbox, • Videotex: service code, • IBM: device name.	Sets the service code (depending of the terminal class).

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
void	***setVicUser***(java.lang.String vicUser) **Parameters:** `vicUser` - the user name.	Sets the name of the user.
void	***showZones***(java.util.Vector rv, java.util.Vector cv) **Parameters:** `rv` - the vector containing the zones rectangles. If `rv` is `null`, the zones will not be shown anymore. `cv` - the vector containing the color of each zone.	Shows Zones on the emulator screen. The zones will be represented as rectangles bounding characters. This feature can be used to enhance or to show a grid system on the emulator screen. Zones are described in a vector of rectangles in character 0 based coordinates. For example, `Rectangle (0, 0, 10, 1);` will describe a zone in column 0, line 0, of 10 chars.
void	***Trace***(java.lang.Exception e) **Parameters:** e - the exception to log.	Writes an exception in the emulator log output stream.
void	***Trace***(int level, java.lang.String message) **Parameters:** `level` - the log level to use. `message` - the message to write.	Writes a message in the emulator log output stream with a given log level.
void	***Trace***(java.lang.String message) **Parameters:** `message` - the message to write.	Writes a message in the emulator log output stream.
boolean	***waitAt***(java.lang.String searchedString, int column, int line, long timeOut) **Parameters:** `searchedString` - the string to be searched. `column` - the horizontal coordinate of the trigger. `line` - the vertical coordinate of the trigger. `timeout` - the maximum amount of milliseconds during which the wait is made. **Returns:** `true` if the event is fired, `false` otherwise. **See also:** waitAtId(int, String, int, int), waitForId(int, String), waitFor(String, long), deleteWaitAts(), deleteWaitFors()	Sets up a synchronous screen trigger. This method returns only when the string `searchedString` is recognized on the screen at coordinates (`column`, `line`).
void	***waitAtId***(int id, java.lang.String searchedString, int column, int line) **Parameters:** `id` - the trigger id. This ID is transmitted by the `WaitAtDone` event. `searchedString` - the string to be searched. `column` - the horizontal coordinate of the trigger. `line` - the vertical coordinate of the trigger. **See also:** waitAt(String, int, int, long), waitForId(int, String), waitFor(String, long), deleteWaitAts(), deleteWaitFors()	Sets up an asynchronous screen trigger. When the string `searchedString` is recognized on the screen at (`column`, `line`) coordinates, the `WaitAtDone` event is generated by the applet with the identificator `id`. You can set many observations at the same time. Once the event is generated, the observation is freed.

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
boolean	***waitCursorAt***(int column, int line, boolean here, long timeout) **Parameters:** `column` - the horizontal coordinate of the cursor. `line` - the vertical coordinate of the cursor. `here` - if `true`, the function waits for the cursor to be at the specified position. if `false`, the function waits for the cursor to be at any position different from the one specified. `timeout` - the maximum amount of milliseconds to wait. **Returns:** `true` if the cursor is found at this position before timeout, `false` otherwise. **See also:** waitAtId(int, int, int, boolean)	Waits synchronoulsy for the cursor to be at a specified position. This function is useful to detect a new page coming in Videotex and VTxxx emulators.
void	***waitCursorAtId***(int Id, int column, int line, boolean here) **Parameters:** `id` - the trigger id. This ID is received in the `WaitAtDone` event or in the `WaitTrigger()` function. `column` - the horizontal coordinate of the cursor. `line` - the vertical coordinate of the cursor. `here` - if `true`, the function waits for the cursor to be at the specified position. if `false`, the function waits for the cursor to be at any position different from the one specified.	Waits Asynchronoulsy for the cursor to be at a specified position.This function is useful to detect a new page coming in Videotex and VTxxx emulators.
boolean	***waitFor***(java.lang.String searchedString, long timeout) **Parameters:** `searchedString` - the string to be searched. `timeout` - the maximum amount of milliseconds during which the search is performed. **Returns:** `true` if the string is found before timeout, `false` otherwise. **See also:** waitAt(String, int, int, long), waitAtId(int, String, int, int), waitForId(int, String), deleteWaitAts(), deleteWaitFors()	Set up a synchrounous line trigger.
boolean	***waitForDataStable***(int timeout, int dataStableThreshold) **Parameters:** `timeout` - the maximum amount of milliseconds during which the event is expected. `dataStableThreshold` - this value is the maximum time allowed with no data received from the host to consider a dataStable condition. **Returns:** `true` if the screen is stable, `false` otherwise.	Waits for the screen to be stable. This routine will return after timeout, even if the screen isn't still stable.
void	***waitForId***(int id, java.lang.String Str) **Parameters:** `id` - the trigger id. This ID is received in the `WaitAtDone` event or in the `WaitTrigger()` function. `str` - the string to be observed. **See also:** waitAt(String, int, int, long), waitAtId(int, String, int, int), waitFor(String, long), deleteWaitAts(), deleteWaitFors()	Set up a screen trigger. When the string `Str` is recognized in the data stream the event `id` will be generated. You can set many Triggers at the same time. Once the event is generated, the Trigger is freed. You can wait for these triggers with the `WaitTrigger()` function.

Table 3 - 2: Methods list (...)

Return Type	Method signature	Description
void	***waitSync***(int timeOut) **Parameters:** `timeout` - the maximum amount of milliseconds during which the wait will occur.	Calls the *Javelin* `wait()` method.
int	***waitTrigger***(long timeout) **Parameters:** `timeout` - the maximum amount of milliseconds during which the event is expected. **Returns:** -1 if the timeout has expired, otherwise the trigger ID. **See also:** waitAt(String, int, int, long), waitAtId(int, String, int, int), waitForId(int, String), deleteWaitAts(), deleteWaitFors()	Waits for a trigger that has been set by one of the `waitAt()` or `waitFor()` functions.

3.2 Context object

Context is the object representing the context of execution of transactions or sequences. This section presents the context, how it works in Convertigo, and then contains the API documentation of the *Context* object.

- Context general presentation
- Context API documentation

3.2.1 Context general presentation

This section presents the Convertigo context:

- Definition
- Identification
- Context object

DEFINITION

Each time a request is sent to Convertigo, a *context* is created in the Convertigo Server engine. A *context* is a kind of dedicated environment and specific tunnel between the client and the Convertigo Server. It contains all the relevant information required to process the request: a copy of the requested project and all its necessary objects, the transaction or sequence execution scopes, cookies, variables... If a context already exists and is available, it can also be re-used, depending on conditions that are explained thereafter.

IDENTIFICATION

A Convertigo *context* is identified by a `contextId`, based on the session ID.

The standard template for generating the `contextId` is: `<JSESSIONID>_<contextName>`.

Let's detail these two parts of the `contextId`:

- **JSESSIONID**: It is the HTTP session ID, identifier of the user session (like `JSESSIONID` in Tomcat). All requests made by the same client have to be done using the same HTTP session in order to keep using the same context.

Session ID contains a reference to the informations of the session. This ID is transferred by the client browser, so that the application can link the HTTP request and the associated user session together. This transfer is made using browser cookies.

Be aware that different clients can have different behavior in terms of session management. For example:
Two different instances of Internet Explorer open two different HTTP sessions. But if you use the New Window menu function from IE, then the spawned window will retain its parent sessions.
When requesting the same server, Firefox tabs create different HTTP sessions, but Internet Explorer Tabs will only use one.
As for web services clients, like Microsoft .NET, or Java Axis, they most of the time use a specific parameter to retain sessions between succesive calls to a server.

- **contextName**: The context name differentiates several contexts created for the same HTTP session. By default, `contextName` value is "`default`", corresponding to only one Convertigo context for a given HTTP session. When no context name is specified by the request, this default context name is always used and re-used. It is not the case in the following situations:
  - Unlike the HTTP session ID, which is fixed by the server at the first connection from a client, the context name can be chosen by the client and sent as a request parameter: `__context=XXXX`. Therefore, consecutive requests must use not only the same HTTP session but also the same context name to re-use the same execution context.
  - In the case of a transaction or sequence initiated by a sequence, through a *Call Transaction* or a *Call Sequence* step (not initiated directly by a client), the context name is automatically generated, and can also be specified in the call step.

CONTEXT OBJECT

The Convertigo developer has access to the execution context of a transaction or a sequence directly in this transaction or sequence, simply by using the *context* object. This object is usable in JavaScript code (in transaction's core for a Legacy transaction, in a *Transaction JS* statement for an HTML transaction, in a *Sequence JS* step for a sequence, etc).

The *context* object can be useful to store data, for example variables, XML chunks, etc. and retrieve this data in another request execution during the same session, as well as to encode and decode data.

But the *context* object is also a useful tool to control some behaviors of the running transaction or sequence, like the cache storage or pool locking, or to add elements to the output XML, etc. See the complete API of the object on the section *"Context API documentation"* on page 3 - 13.

3.2.2 Context API documentation

The followng sections present the fields and methods of the *Context* object that are usable in transactions and sequences JavaScript code.

- Fields detailed list
- Methods detailed list
- Interesting methods in Context fields

FIELDS DETAILED LIST

This section presents the list of fields contained in *Context* object, with their description.

Table 3 - 3: Context fields list

Field name	Type	Description
contextID	String	The context unique identifier.
contextNum	int	The context number (it is incremented by 1 for each newly created context).
creationTime	long	The context creation time (as a timestamp).
httpServletRequest	HttpServletRequest	The HTTP servlet request object, when the transaction/ sequence currently executed is called externally, through HTTP, by the client. When called through `internal invoke` (for Call Transaction/ Call Sequence steps), the HTTP servlet request object is spread from parent sequences/parent context throughout the sequences hierarchy.
httpSession	HttpSession	The HTTP session object.
inputDocument	Document	The input XML document generated by the requester.
isCacheEnabled	boolean	Indicates whether the cache functionnality is enabled. Default value is `true`, you can set this parameter to `false` during the transaction/sequence execution in order not to store the XML response in the cache (for example, in case of error, the response should not be stored).
lastAccessTime	long	The last access time to this context.
lockPooledContext	boolean	Indicates whether the context is to be kept in the pool even if it is not in the expected state (i.e. wrong screen class). Usually, a pooled context is automatically destroyed after a transaction execution if it is left on a wrong screen class (not the stable state screen class for this context). This property enables keeping this context alive in the pool for a further use.
outputDocument	Document	The output XML document generated by the current transaction.
parentContext	Context	The context of parent sequence, if the currently executed transaction/sequence has a parent sequence, i.e. if the current transaction/sequence is executed through a Call Transaction/ Call Sequence step from another sequence (called parent sequence). Otherwise, this property is `null`.
remoteAddr	String	The remote address from the calling client.
remoteHost	String	The name of the calling client (if reverse DNS has been enabled).
requireEndOfContext	boolean	Indicates whether the context is to be destroyed after the transaction handling. By default set to `false`, a context is re-usable. This property can be set to `true` if the context needs to be destroyed after a transaction execution.
servletPath	String	The servlet path for the current request.
steps	Vector<String>	The steps objects, useful for asynchronous transaction. These are the asynchronous mode steps objects, not to be confused with Steps of Sequences.
tasCommDevice	String	The TAS (VIC or Carioca) communication device for establishing the connection.

Table 3 - 3: Context fields list (...)

Field name	Type	Description
tasDteAddress	String	The TAS (VIC or Carioca) remote address for connection.
tasServiceCode	String	The TAS (VIC or Carioca) service code.
tasSessionKey	String	The TAS (VIC or Carioca) session key.
tasUserGroup	String	The TAS (VIC or Carioca) user group.
tasUserName	String	The TAS (VIC or Carioca) user name.
tasUserPassword	String	The TAS (VIC or Carioca) user password.
tasVirtualServerName	String	The TAS (VIC or Carioca) virtual server name.
userAgent	String	The user agent from the calling client.

METHODS DETAILED LIST

This section presents the list of methods contained in *Context* object, with their description and parameters.

Table 3 - 4: Context methods list

Return Type	Method signature	Description
void	***abortRequestable***()	Requests the end of the transaction/sequence running in the current context as soon as possible and without any condition.
Node	***addTextNode***(Node parentNode, String tagName, String text) **Parameters:** `parentNode` - the parent node into which the new node should be inserted `tagName` - the new node tag name `text` - the new node text content **Returns:** the created `node`	Adds a new node containing text in the output XML document, under an identified parent node.
Node	***addTextNodeUnderBlock***(String tagName, String text) **Parameters:** `tagName` - the new node tag name `text` - the new node text content **Returns:** the created `node`	Adds a new node containing text in the output XML document, under the `blocks` node (in the context of a Legacy transaction).
Node	***addTextNodeUnderRoot*** (String tagName, String text) **Parameters:** `tagName` - the new node tag name `text` - the new node text content **Returns:** the created `node`	Adds a new node containing text in the output XML document, under the root node (`document` element).
String	***decodeFromHexString***(String s) **Parameters:** `s` - the string to decrypt; this string must have been encoded by the `encodeToHexString()` function in order to stay meaningfull. **Returns:** the decrypted string or `null` if any error occurs. **See also:** encodeToHexString(String), encodeToHexString(String, String)	Decrypts a string using the triple DES algorithm and the Convertigo default passphrase.

Table 3 - 4: Context methods list (...)

Return Type	Method signature	Description
String	***decodeFromHexString***(String passphrase, String s) **Parameters:** `passphrase` - the ciphering passphrase to use for decoding. `s` - the string to decrypt; this string must have been encoded by the `encodeToHexString()` function in order to stay meaningfull. **Returns:** the decrypted string or `null` if any error occurs. **See also:** encodeToHexString(String), encodeToHexString(String, String)	Decrypts a string using the triple DES algorithm and the passphrase passed as parameter.
String	***encodeToHexString***(String s) **Parameters:** `s` - the string to encrypt. **Returns:** the encrypted string; the script is of hexadecimal string format, i.e. it contains only hexadecimal (printable) characters, or `null` if any error occurs. **See also:** decodeFromHexString(String), decodeFromHexString(String, String)	Encrypts a string using the triple DES algorithm and the Convertigo default passphrase.
String	***encodeToHexString***(String passphrase, String s) **Parameters:** `passphrase` - the ciphering passphrase to use for encoding. `s` - the string to encrypt. **Returns:** the encrypted string; the script is of hexadecimal string format, i.e. it contains only hexadecimal (printable) characters, or `null` if any error occurs. **See also:** decodeFromHexString(String), decodeFromHexString(String, String)	Encrypts a string using the triple DES algorithm and the passphrase passed as parameter.
Object	***get***(String key) **Parameters:** `key` - the key identifying the requested object. **Returns:** the object bound with the key in the context, or `null` if no object is bound with this key in the context. **See also:** keys(), set(String, Object), remove(String)	Gets the object bound with the specified key in the context, or `null` if no object is bound with this key in the context..
String	***getAbsoluteRequestedUrl***() **Returns:** the absolute requested URL. **See also:** getConvertigoUrl(), getProjectUrl()	Gets the absolute URL of currently executed request.
String	***getAuthenticatedUser***() **Returns:** the authenticated user ID from the context/session, or `null` if the context/session is not authenticated. **See also:** setAuthenticatedUser(String), removeAuthenticatedUser()	Gets the authenticated user ID from the context/session, if the context/session is authenticated. Otherwise, returns `null`.
String	***getConvertigoUrl***() **Returns:** the absolute URL to Convertigo Server web application. **See also:** getAbsoluteRequestedUrl(), getProjectUrl()	Gets the absolute URL to Convertigo server web application.
String	***getProjectDirectory***() **Returns:** the path to project directory.	Returns the path to the current project directory.

Table 3 - 4: Context methods list (...)

Return Type	Method signature	Description
String	***getProjectName***() **Returns:** the project name.	Returns the name of the current project.
String	***getProjectUrl***() **Returns:** the absolute URL to the project root. **See also:** getAbsoluteRequestedUrl(), getConvertigoUrl()	Gets the absolute URL of the current project root.
Context	***getRootContext***() **Returns:** the context of initial parent sequence, i.e. the sequence that is called externally, through HTTP, by the client. **See also:** parentContext	Gets the context object of the sequence called by the client, the first sequence called in currently executed sequences hierarchy. This can be the current context when no call hierarchy is executed, i.e. when the current context is the context of a transaction/sequence called directly by the client.
Object	***getTransactionProperty***(String propertyName) **Parameters:** `propertyName` - the name of the property to retreive. **Returns:** the property value; depending on the property this value may be an object .	Retrieves the value of a property of the current transaction.
boolean	***isSOAPRequest***() **Returns:** `true` if the request is a SOAP request (i.e. if the request path ends by `.ws` or `.wsl`), otherwise returns `false`.	Says if the request that initiated the transaction/ sequence is a SOAP request or not.
Set<String>	***keys***() **Returns:** the collection of keys identifying objects stored in the context. **See also:** get(String), set(String, Object), remove(String)	Returns the collection of keys identifying objects stored in the context.
Properties	***loadPropertiesFromProject***(String fileName) **Parameters:** `fileName` - the name of the properties file to load. **Returns:** the loaded properties.	Load properties from a file located in current project folder.
Properties	***loadPropertiesFromWebInf***(String fileName) **Parameters:** `fileName` - the name of the properties file to load. **Returns:** the loaded properties.	Load properties from a file located in WEB-INF folder.
void	***remove***(String key) **Parameters:** `key` - the key identifying the object to remove. **See also:** get(String), keys(), set(String, Object)	Removes the object bound with the requested key from the context.
void	***removeAuthenticatedUser***() **See also:** getAuthenticatedUser(), setAuthenticatedUser(String)	Removes the authenticated user ID from the context/session. The context/session is not authenticated anymore.
boolean	***savePropertiesToProject***(String fileName, Properties properties) **Parameters:** `fileName` - the name of the file to save properties. `properties` - the properties to save. **Returns:** `true` if the save succeeds.	Saves properties in a properties file in current project folder

Table 3 - 4: Context methods list (...)

Return Type	Method signature	Description
boolean	***savePropertiesToWebInf***(String fileName, Properties properties) **Parameters:** `fileName` - the name of the file to save properties. `properties` - the properties to save. **Returns:** `true` if the save succeeds.	Saves properties in a properties file in WEB-INF folder.
void	***set***(String key, Object value) **Parameters:** `key` - the key identifying the object. `value` - the object to bind with the key. **See also:** get(String), keys(), remove(String)	Stores an object identified by a key into the context.
void	***setAuthenticatedUser***(String userID) **Parameters:** `userID` - the user ID that has to be positioned in context/session as authenticated user. **See also:** getAuthenticatedUser(), removeAuthenticatedUser()	Sets the context/session as authenticated and stores the userID as the authenticated user ID.
boolean	***waitAtScreenClass***(int timeout, int hardDelay) **Parameters:** `timeout` - the time (in ms) we have to wait for the screen class. `hardDelay` - a delay (in ms) added after the screen class has arrived. **Returns:** `true` if we the screen did arrive, `false` otherwise.	**This method only concerns Minitel projects.** Waits for one of the screens described by the screen classes in the project to arrive. The method waits for all the screen classes except the current one. You can use `waitAtScreenClass()` method to synchronize your handler before returning `"redetect"`, `"accumulate"` or `"skip"`.
boolean	***waitNextPage***(String action, int timeout, int hardDelay) **Parameters:** `action` - the action to perform before waiting. `timeout` - the time (in ms) we have to wait for the screen class. `hardDelay` - a delay (in ms) added after the screen class has arrived. **Returns:** `true` if we the screen did arrive, `false` otherwise.	**This method only concerns Minitel projects.** Waits for a new page for the same screen class or a new screen class. The method wait for one of the screens described by the screen classes in the project to arrive. We wait for all the screen classes except the current one. In the case of a next page on the same screen class, `waitNextPage()` will monitor the cursor position. the method will return when the cursor position returns to the same position it was before calling `waitNextPage()`. You can use `waitNextPage()` method to synchronize your handler before returning `"redetect"`, `"accumulate"` or `"skip"`.

Interesting methods in Context fields

Some fields of *Context* object are themselves objects, containing interesting methods to use in Convertigo transactions and sequences JavaScript code. The following list present these objects methods, with their description and parameters.

Table 3 - 5: Interesting methods in Context fields

Return Type	Method signature	Description
String	***context.httpServletRequest.getMethod***()	Returns the name of the HTTP method with which the request to Convertigo was made. It can be GET, POST, PUT, DELETE, HEAD.

4 Interfaces to Convertigo

This chapter offers information about how to access to Convertigo projects and execute transactions and sequences.

- HTTP protocol interface to Convertigo
- Web service interface to Convertigo

4.1 HTTP protocol interface to Convertigo

Convertigo projects contain "requestable" objects, transactions and sequences, that can be invoked by HTTP protocol. This section describes possible URLs to access Convertigo projects and all reserved variables usable in these URLs.

4.1.1 Convertigo URLs

GENERAL PROCESS

The following general process describes the HTTP protocol used from the client (web browser or application) to the Convertigo server.

1 The client issues an HTTP request to Convertigo to the following root URL:

```
http(s)://<ConvertigoServer>/<ConvertigoAppName>/projects/
<ProjectName>/<ConvertigoRequester>
```

- ***ConvertigoServer*** is the host name or IP address of your Convertigo server, with possibly a port number**.**
  - A local Convertigo Studio has as host name `localhost` and, in HTTP, `18080` as default port number, in HTTPS, `18081` as default port number.
  - A Convertigo Server installed locally on your computer shares the same host name: `localhost`. The Convertigo Server default port number is `28080` for HTTP and `28443` for HTTPS.
  - When installed in an existing application server, it has as IP address and ports the IP address and ports of the application server.
  - A Convertigo Cloud server is accessed through its server name, for example: `me.convertigo.net` (no port to provide in this case, either for HTTP or HTTPS).
- ***ConvertigoAppName*** defines the name of the Convertigo web application.
  - A local Convertigo Studio or a Convertigo Server installed locally has `convertigo` as Convertigo app name.
  - A Convertigo Cloud server has `cems` as Convertigo app name.
- ***ProjectName*** defines the name of the project you want to invoke in Convertigo. If the project doesn't exist, the application server generates an error (`HTTP 404 Not Found`).
- ***ConvertigoRequester*** defines the requester to use on the Convertigo server. It can take several values that are described further (see *"Convertigo requesters"* on page 4-3).

For example, with a project named `test_project` in a local Convertigo Studio:

- *ConvertigoServer* = `localhost:18080`
- *ConvertigoAppName* = `convertigo`
- *ProjectName* = `test_project`

2 This HTTP request can use the following HTTP verbs: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`. Convertigo would handle all these verbs the same way, by executing the called transaction / sequence. Only `HEAD` verb will have a different result, see point 4 of this procedure.

The verb can be retrieved inside JavaScript code in Convertigo transaction / sequence using the following code line:
`context.httpServletRequest.getMethod()`
For more information about ***context*** *object fields and methods, see "Context object" on page 3-12.*

3 The root URL can be followed by a query string, possibly containing reserved variables and variables to be passed to the invoked transaction / sequence. These variables may also be passed in `POST` data.

For more information about specific parameters, see "Convertigo reserved parameters" on page 4-6.

4 Convertigo handles the request, executing the requested transaction / sequence, and returns the response to the client.

- Depending on the invoked requester, the response can contain different documents and formats, see *"Convertigo requesters"* on page 4-3.
- Depending on the HTTP verb used in the request, the response may be different: `HEAD` verb will only return headers in response (following HTTP specifications).

5 The client (web browser or application) retrieves and manages the response, also depending on the invoked requester and the project's parametrization.

CONVERTIGO REQUESTERS

Several requesters can be invoked in Convertigo to execute a transaction / sequence. This section explains in details the different values that can be passed in ***ConvertigoRequester*** part of the URL.

WEBLIB REQUESTER

This requester is invoked by calling the `index.html` file of the project, i.e. setting the ***ConvertigoRequester*** at the end of the URL to `index.html`.

For example, with a project named `test_project` in a local Convertigo studio, the URL to call a transaction / sequence is:

```
http://localhost:18080/convertigo/projects/test_project/index.html
```

The weblib requester automatically includes a complete *JavaScript AJAX* library based on

JQuery that allows:

- invoking Convertigo transactions / sequences / project resources,
- managing the Convertigo responses,
- performing XSL transformation (if necessary),
- encoding request parameters,
- sequencing several calls to Convertigo,
- including in a portal the widget generated from the transaction / sequence response,
- developing interactions between widgets,
- etc.

This is the recommended requester to use.

The client requests the project's index, its request is handled by the JavaScript library, which performs an AJAX call to Convertigo engine and handles the response.

Some parametrization of this library framework can be performed by updating global variables values. Lots of these variables can be parametered thanks to weblib reserved variables, that have to be passed in the request. For more information about weblib specific parameters, see *"Weblib reserved parameters"* on page 4-11.

The parametrization of the engine request, i.e. the definition of the transaction / sequence to call, etc. is done by using engine reserved variables. For more information about engine specific parameters, see *"Engine reserved parameters"* on page 4-7.

XML REQUESTERS

These requesters are invoked by setting the ***ConvertigoRequester*** at the end of the URL to the following expression: ***<PoolName>.<Extension>***.

- ***PoolName*** defines the name of the pool to invoke, in the case of pools use in the project. It is an optionnal parameter, it can be empty. Any value can be passed in this part of the URL: if a matching pool is found, it is used by Convertigo, if no pool is found, it is ignored and Convertigo returns to the default behavior. For more information about *Pool* object, see "*Pool*" documentation and examples.
- ***Extension***, ending the URL, can take the following values:
  - `xml`,
  - `cxml`,
  - `pxml`,
  - `cpdf`.

For example, with a project named `test_project` not including pools, in a local Convertigo studio, the URL to call a transaction / sequence can be:

```
http://localhost:18080/convertigo/projects/test_project/.xml
```

The XML requesters allow calling a transaction / sequence of a Convertigo project, possibly in the context of a pool. They allow retreiving the XML response and possibly performing an XSL transformation.

The difference between the four extensions is about the XSL transformation. It is described in the table above:

Table 4 - 1: XML requesters extensions

Extension	Description
`.xml`	Convertigo sends the transaction / sequence XML as response, possibly including the reference to the XSL file to use for the transformation (defined in the Convertigo project). Then, the client (web browser or application) should perform the XSL transformation.
`.cxml`	Abbreviation for Convertigo-XML. Convertigo performs the XSL transformation on server side, if a style sheet is defined in the Convertigo project. Then, it sends the result as response to the client.
`.pxml`	Abbreviation for Pure-XML. Convertigo sends the transaction / sequence XML as response, no XSL transformation is performed nor referenced, whatever is defined in the project.
`.cpdf`	Abbreviation for Convertigo-PDF. Convertigo performs the `XSL:FO` transformation on server side, if an `XSL:FO` style sheet is defined in the Convertigo project. Then, it sends the result as PDF file to the client.

Using these requesters, the client interacts directly with Convertigo engine.

The parametrization of the engine request, i.e. the definition of the transaction / sequence to call and related parameters, is done by using engine reserved variables. For more information about engine specific parameters, see *"Engine reserved parameters"* on page 4-7.

JSON REQUESTERS

These requesters are invoked by setting the *`ConvertigoRequester`* at the end of the URL to the following expression: ***`<PoolName>.<Extension>`***.

- ***PoolName*** defines the name of the pool to invoke, in the case of pools use in the project. It is an optionnal parameter, it can be empty. Any value can be passed in this part of the URL: if a matching pool is found, it is used by Convertigo, if no pool is found, it is ignored and Convertigo returns to the default behavior. For more information about *Pool* object, see "*Pool*" documentation and examples.
- ***Extension***, ending the URL, can take the following values:
  - `json`,
  - `jsonp`.

For example, with a project named `test_project` not including pools, in a local Convertigo studio, the URL to call a transaction / sequence can be:

```
http://localhost:18080/convertigo/projects/test_project/.json
```

The JSON requesters allow calling a transaction / sequence of a Convertigo project, possibly

in the context of a pool, and retreiving the JSON response.

The difference between the two extensions is about cross-domain issues. It is described in the following table:

Table 4 - 2: JSON requesters extensions

Extension	Description
`.json`	This extension allows an application to request Convertigo through an AJAX call, in the same domain. Then, the application retrieves the transaction / sequence response as a JSON stucture and should handle the data treatment.
`.jsonp`	This extension allows an application to request Convertigo through a fake AJAX call (dynamic inclusion of script elements), not in the same domain (**cross-domain compatibility**). Then, the application is called back, receiving as parameter the transaction / sequence response as a JSON stucture, and the call-back function should handle the data treatment.

Using these requesters, the client interacts directly with Convertigo engine, dynamically transforming standard output XML data to JSON structure.

The parametrization of the engine request, i.e. the definition of the transaction / sequence to call and related parameters, is done by using engine reserved variables. For more information about engine specific parameters, see *"Engine reserved parameters"* on page 4-7.

Call-back function specific to JSONP requester is parametered using a specific engine reserved variable. For more information about specific engine reserved parameters, see *"JSONP specific reserved parameters"* on page 4-8.

BINARIES REQUESTER

This requester is invoked by setting the *`ConvertigoRequester`* at the end of the URL to the following expression: ***`.bin`***.

For example, with a project named `test_project`, in a local Convertigo studio, the URL to call a transaction / sequence can be:

```
http://localhost:18080/convertigo/projects/test_project/.bin
```

The binaries requester allows retrieving the last attachment file downloaded by a transaction / sequence of a Convertigo project. It can be called directly with the transaction extracting the attachment.

Using this requester, the client interacts directly with Convertigo engine.

The parametrization of the engine request, i.e. the definition of the transaction / sequence to call and related parameters, is done by using engine reserved parameters. For more information about engine specific parameters, see *"Engine reserved parameters"* on page 4-7.

4.1.2 Convertigo reserved parameters

All Convertigo specific parameters are prefixed with '__' (double underscored). Depending on the invoked requester, some reserved parameters can be used by the *weblib AJAX framework*, some by *Convertigo engine*.

- Engine reserved parameters

- Weblib reserved parameters

ENGINE RESERVED PARAMETERS

Convertigo engine request parametrization, i.e. the definition of the transaction / sequence to call and related execution parameters, is done by using **generic** engine reserved parameters that have to be passed in the request.

Other specific engine reserved parameters exist and can be used to handle **connector-related** specific cases.

GENERIC ENGINE RESERVED PARAMETERS

Here is the list of Convertigo engine-managed reserved parameters, valid and usable for all types of projects, and their description:

Table 4 - 3: Generic engine reserved parameters

Parameter name	Description
__connector	Name of the requested connector. If this parameter is not present, the default connector of the requested project is used.
__transaction	Name of the transaction to run. If this parameter is not present or its value is an empty string, the default transaction of specified connector is executed.
__sequence	Name of the sequence to run. As no default sequence exists for a project, if this parameter is not present or its value is an empty string, the default transaction from default connector is executed.
__project	Overrides the name of the requested project. The project name is mostly set in the URI: `http(s)://<ConvertigoServer>/<ConvertigoAppName>/projects/<ProjectName>/<ConvertigoRequester>?<parameters>`. For more information on HTTP request URI, see *"Convertigo URLs"* on page 4-2 If this parameter is present, it overrides the project name from the URI.
__testcase	Name of the test case to run, in specified transaction / sequence. If this parameter is present, the variable values will be retrieved from the test case definition.
__context	Name of the context in which run the specified transaction / sequence. If this parameter is not present, Convertigo automatically creates or reuses a context, named `default`, and attaches it to the client HTTP session cookie (`JSessionId`). When XML response is returned from Convertigo, the context name is present in the `context` attribute of the document element. To reuse a context that was previously created, the `__context` parameter should be set to the context name returned by Convertigo.
__user_reference	This parameter is a user reference passed in entry to Convertigo transaction / sequence, which is automatically inserted unchanged in the resulting output response. It can be useful for the caller to be able to exactly determine from which request a response belongs. The value of this parameter is automatically added in the generated XML in the `userReference` attribute of the `document` element.

Table 4 - 3: Generic engine reserved parameters

Parameter name	Description
__stub	If `true`, the requested transaction/sequence response is retrieved from the stub response (if a stub response is present for this transaction/sequence in the project). Notes: • When executing a transaction/sequence from stub, the **Authenticated context required** property of the transaction/sequence is ignored: the context would never be authenticated as the transaction/sequence setting the context as authenticated could also be executed from stub • In Convertigo Studio, a stub response can be easily created for a transaction/sequence. To do so, execute the transaction/sequence, possibly using a test case. Then, right-click on the transaction/sequence and select the **Create stub from current generated XML** option. A stub file is created from the current response XML for the transaction/sequence.
__nocache	If `true`, the requested transaction / sequence response is not retrieved from the cache (in the case of a cached transaction / sequence). Convertigo ignores the cached response and returns a freshly built response. For more information about transaction / sequence cache parametrization, see **Response life-time** property documentation in chapter 2 of this manual.
__supervision	If this parameter is present, the requested transaction / sequence response is not stored in the cache (in the case of a cached transaction / sequence). If a response was already stored in the cache, it is not updated. For more information about transaction / sequence cache parametrization, see **Response life-time** property documentation in chapter 2 of this manual.
__removeContext	If this parameter is present and its value not equal to `false`, the end of Convertigo context is required at the end of the transaction / sequence execution. This is mainly used in case of web Services projects. This parameter is similar to the following JavaScript statement set in the transaction / sequence core: `context.requireEndOfContext = true;`
__removeNamespaces	If this parameter is present, the namespaces or namespace suffixes that appear in the XML output are removed before Convertigo sends the response. Note: For example, namespace suffixes can appear in XML ouput after XML nodes are copied from Call Transaction/Call Sequence step responses.
__content_type	Overrides the `Content-type` HTTP header value of the Convertigo response for the REST requesters (`.xml`, `.pxml`, `.json`).
__lang	Defines the output language for the requested transaction/sequence response. This parameter value is added as a `lang` attribute in the document element of the output XML and has to be managed with the response XML. Once the `__lang` parameter is received for a transaction/sequence, the context keeps and re-uses this value in every other transaction/sequence output XML. The `lang` attribute of the document element is automatically used by the legacy translation extraction rule. For more information, see *Translate text* extraction rule documentation in chapter 2 of this manual.
__async	If `true` (or value `1`), the requested transaction will be or is being processed asynchroneously, an asynchronous job is created. ***Warning!*** Asynchronous mode should be used in specific cases and has to be evaluated by a Convertigo expert developer.
__abort	If this parameter is present, the end of the asynchronous job is required. ***Warning!*** Asynchronous mode should be used in specific cases and has to be evaluated by a Convertigo expert developer.

JSONP SPECIFIC RESERVED PARAMETERS

The following is the engine reserved parameter specific to JSONP requester and its description:

Table 4 - 4: JSONP specific engine reserved parameters

Parameter name	Description
__callback	Defines the name of the client application function to call back after the Convertigo request execution on a cross-domain platform.

WEB CONNECTOR-SPECIFIC RESERVED PARAMETERS

The following list is the list of web connector-specific engine reserved parameters and their description:

Table 4 - 5: Web connector-specific engine reserved parameters

Parameter name	Description
__header_<headerName>	Allows to dynamically pass an HTTP header to an HTTP transaction. It results into sending an HTTP header from Convertigo to the target HTTPserver. This specific parameter defines two pieces of information: • <headerName>: defines the HTTP header name to send to the target server, • the parameter value is sent as header value to the target server. For example, the following specific parameter can be sent to a transaction: __header_test=myTestHeader which results in sending the following header to the target server reached by the HTTP connector: test=myTestHeader.
__<HTTPverb>_<varName>	Allows to dynamically pass an HTTP variable to an HTTP transaction. It results into sending an HTTP variable from Convertigo to the target HTTPserver. This specific parameter defines three pieces of information: • <HTTPverb>: defines the HTTP verb to use to send this variable to the target server, can take both values POST or GET, • <varName>: defines the variable name to send to the target server, • the parameter value is sent as variable value to the target server. For example, the following specific parameters can be sent to a transaction: __GET_login=johnsmith&__POST_password=mypassword which results in sending the following variables to the target server reached by the HTTP connector: • login=johnsmith - sent in the URL query string, • password=mypassword - sent in POST data.
__uri	Allows to dynamically change the value of the **Sub path** property of the requested HTTP transaction. Warning! At the end of the transaction execution, the **Sub path** property is set back to its original value.
__statefull	Allows to dynamically change the value of the **Maintains connector state** property of the requested HTML transaction. If set to true, the running transaction property is changed to true. If set to false, the running transaction property is changed to false. Warning! At the end of the transaction execution, the **Maintains connector state** property remains to the updated value: the original value is not set back.

LEGACY EMULATOR-SPECIFIC RESERVED PARAMETERS

The following list is the list of legacy emulator-specific engine reserved parameters and their description:

Table 4 - 6: Legacy emulator-specific engine reserved parameters

Parameter name	Description
__service	Optional. Used at connection time only to input a specific connection parameter depending on connector type: • IBM 3270,5250: device name • Bull DKU: mailbox • VDX 40/80col: service code. When this parameter is used, it also overrides the mainframe connection IP adress or name set in the project connector. The correct syntax for this parameter value is: `<devicename>,DIR|<mainframe_ip>:<mainframe_port>`
__javelin_current_field	This parameter holds the name of the current selected field. Field names must be in the form `___field_l<Y>_c<X>`, with: • *X*: the field column, starting at 1, • *Y*: the field line,starting at 1. Convertigo uses this value to position the input cursor on the specified field before issuing an action as 'KEY_ENTER'.

__javelin_action	This parameter holds the name of the action to be done on the target application. Actions can be:
	• any emulator actions as described in the Appendix *"Legacy emulator actions table"* on page A - 8,
	• one of the following Convertigo commands:
	  • `convertigo_reconnect`: disconnects and reconnects the emulator session,
	  • `convertigo_refresh`: does nothing, just get the current XML,
	  • `convertigo_destroy_session`: disconnects, and recreates an emualtor instance (this may be used to reset completly an emulator session).

__field_l<y>_c<x>	Any parameter of this form has its value inserted automatically in the target field, where: • *X*: the field column, starting at 1, • *Y*: the field line,starting at 1.

CARIOCA PORTAL-SPECIFIC RESERVED PARAMETERS

The following list is the list of specific engine reserved parameters used when Convertigo is accessed through Carioca portal and their description:

Table 4 - 7: Engine reserved parameters for access through Carioca

Parameter name	Description
__sesskey	Authentication key generated by Carioca portal. This key is decoded by Convertigo to check the consistency and origin of the request.
__bCarioca	Set to "`true`" defines the origin of the request as coming from Carioca portal. If set to "`true`", the `__sesskey` variable has to be passed and is verified by Convertigo.
__user	Carioca portal user name.
__password	Carioca portal user password.
__bVic	Set to "`true`" defines the user as a former VIC portal user.
__VicUser	Former VIC portal user name.
__VicGroup	VIC portal authorization group. Authorization groups are used in the portal to define to which services the user has the right to access.
__VicServiceCode	VIC portal service name.
__VicDteAddress	Connection address of the VIC service.

Table 4 - 7: Engine reserved parameters for access through Carioca

Parameter name	Description
__VicCommDevice	CommDevice address used for a three thirds architecture (case of the PAVI used for Videotex emulator) through Carioca or VIC portal.

WEBLIB RESERVED PARAMETERS

Weblib framework parametrization can be performed by updating global variables values. Lots of these variables can be set thanks to weblib reserved parameters that have to be passed:

- in the request, using a query string that sends all parameters to the network, or
- by a hash query (using a #) that only sends needed parameters in POST data.

Here is the list of Convertigo weblib-managed reserved parameters and their description:

Table 4 - 8: Weblib reserved parameters

Parameter name	Description
__ajax_method	This parameter enables modifying the HTTP method of the AJAX requests from the weblib to Convertigo. It can take both values: "GET" or "POST". Default variable value is "POST".
__auto_refresh	In case of a Web Clipper project, weblib framework is able to check current page's DOM changes, and to automatically refresh the clipped page displayed to the user. This variable is by default set to "true", enabling auto refreshing. Using this parameter to set it to "false" disables auto refreshing. The auto refresh needs more network trafic because the weblib framework pools the connector state regularly to monitor DOM changes.
__auto_resize	In case of a response presented as a widget in a portal, the weblib framework is able to automatically adapt the height of the widget to its content, when the content is modified by a transaction or sequence result. This variable is by default set to "true", enabling auto resizing. Using this parameter to set it to "false" disables auto resizing.
__enc	If set to "true", activates RSA encoding. Default variable value is "false".
__first_call	This parameter defines whether a call to Convertigo has to be performed using the page's query/hash parameters, after the `init_finished` hook. By default set to "true", the page automatically calls convertigo using these parameters. Beware that the return value of the `init_finished` hook has to be in adequacy with this parameter value or can affect the expected behavior.

Table 4 - 8: Weblib reserved parameters

Parameter name	Description

__localCache	This parameter allows to configure the use of the Local Cache feature on C8O calls and responses. Thanks to the Local Cache, you can save network traffic between the device and the server, and you are able to display data when the device is not connected to the network.
	When enabled, the Local Cache permits to store the responses to a C8O call locally on the device, using the variables and their values as cache key.
	This parameter takes a JSON structure composed as follows:
	• "`enabled`": enables (`true` value) or disables (`false` value) the Local Cache on a Convertigo requestable. Default value is `true`.
	• "`policy`": defines whether the response should be retrieved from Local Cache or from Convertigo server when the device can access the network. Can take two values:
	  • "`priority-server`": for server priority, meaning that if the device can access the network, the call is performed and the response is retrieved from server,
	  • "`priority-local`": for local priority, meaning that even when the device can access the network, the call is not performed if a response is locally cached.
	  In any case, when the device has no network access, the local cached response is used, if existing.
	  The `policy` parameter is mandatory as it has no default value.
	• "`ttl`": defines the time to live of the cached response, in milliseconds. If no value is passed, the time to live is infinite.

__requester_prefix	This parameter defines a prefix before the requester extension in the AJAX requests from the weblib to Convertigo. It matches the pool name part of the URL. This parameter can take any value: if a matching pool is found, it is used by Convertigo, if no pool is found, it is ignored and Convertigo returns to the default behavior.
__resize_offset	Linked to the auto-resizing fonctionnality. This parameter enables defining an offset height (in pixels) to add to the automatically calculated height, in order to adjust the resizing height. This variable is by default set to "`50`" (pixels), the value must be a number (in pixels).
__send_portal_username	In case of a response presented as a widget in gatein portal (Convertigo Mashup Composer), the weblib framework is able to automatically add a `portal_username` parameter to the request to Convertigo with the name of the user logged in the portal as value. This variable is by default set to "`true`", enabling sending portal username. Using this parameter to set it to "`false`" disables sending portal username.
__target_append	The response of a Convertigo transaction / sequence execution, possibly presented thanks to an XSL transformation, can either: • replace the whole previsouly displayed content, • be appended to the previsouly displayed content, thanks to the weblib framework. This variable enables changing the appending mode. Using this parameter to: • set it to "`false`", the response replaces previous content, • set it to "`true`", the response is appended to previous content. Default variable value is "`false`". Paired with the *__target_id* variable, these settings enable to replace or add content to the whole content or a part of the widget.

Table 4 - 8: Weblib reserved parameters

Parameter name	Description
__target_id	The response of a Convertigo transaction / sequence execution, possibly presented thanks to an XSL transformation, can either: • be added at the root of the widget, • be added in an Element from the widget, thanks to the weblib framework. This variable enables defining the `id` of the Element into which the response has to be added. Default value is "" (empty string) and aims the `body` element. If this variable is left empty, the response is added at the root of the widget. Paired with the *__target_append* variable, these settings enable to replace or add content to the whole content or a part of the widget.
__testplatform	If called with no parameters, the weblib framework automatically redirects the client to the test-platform of the project. This parameter enables changing this default behavior. • If set to "`true`", the client is always redirected to the test-platform. • If set to "`false`", the client is never redirected to the test-platform.
__use_siteclipper_plugin	In case of a Site Clipper project, this parameter enables using the iframe encapsulation for Site Clipper requests. If set to "`true`", the clipped web page is loaded through an iframe, which enables: • hiding the Site Clipper URL, • handling interactions and resizing, in case of a response presented as a widget in a portal. If set to "`false`", the clipped page is loaded directly through Site Clipper URL. Default value is "`true`".
__xsl_side	This parameter enables modifying the side of response's XSL transformation, client XSL transformation or server XSL transformation. It matches the requester extension to set in the AJAX requests from the weblib to Convertigo. This variable can take two values: • "`client`" that will call "`.xml`" requester extension, or • "`server`" that will call "`.cxml`" requester extension. Default value is "`client`".

4.2 Web service interface to Convertigo

Convertigo projects can be accessed using web services. This section describes possible URLs to access Convertigo projects through the two types of web services that are supported, then it describes the problem of context state conservation.

- SOAP web services
- REST web services
- Context state conservation

4.2.1 SOAP web services

SOAP WSDL of a Convertigo project can be accessed by using the following URL:

```
http(s)://<ConvertigoServer>/<ConvertigoAppName>/projects/
<ProjectName>/<WsType>?WSDL
```

Where:

- ***ConvertigoServer*** is the host name or IP address of your Convertigo server, with possibly a port number**.**
  - A local Convertigo Studio has as host name `localhost` and as default port number `18080` for HTTP.
  - A Convertigo Server installed locally on your computer shares the same host name: `localhost`. The Convertigo Server default port number is `28080` for HTTP.
  - A Convertigo Cloud server is accessed through its server name, for example: `me.convertigo.net` (no port to provide in this case).

For more information about Convertigo URLs, see "Convertigo URLs" on page 4-2.

Accessing a Convertigo project's WSDL on a Studio or on a Server has a different behavior. In Studio, it uses temporary files dynamically updated while developing the project. In Server, it uses the complete generated WSDL file with schemas.
Prefer to retrieve a project's WSDL from a Convertigo Server.

- ***ConvertigoAppName*** defines the name of the Convertigo web application.
  - A local Convertigo Studio or a Convertigo Server installed locally has `convertigo` as Convertigo app name.
  - A Convertigo Cloud server has `cems` as Convertigo app name.
- ***ProjectName*** defines the name of the project you want to invoke in Convertigo. If the project doesn't exist, the application server generates an error (`HTTP 404 Not Found`).
- ***WsType*** defines the type of encoding you want for SOAP web service. It corresponds to **SOAP web service requesters** that can be invoked on Convertigo. For more information

about the other available requesters, see *"Convertigo URLs"* on page 4-2.

This ***WsType*** part of the URL can take several values that are detailed in following table:

Table 4 - 9: SOAP encoding types

Extension	Description
`.ws`	This extension is used for RPC encoded web services.
`.wsl`	This extension is used for `document/literal` web services.

We strongly recommend to use only `document/literal` SOAP web services nowadays.

For example, with a project named `test_project` in a local Convertigo Server, the URL to retrieve the SOAP WSDL can be:

```
http://localhost:28080/convertigo/projects/test_project/.wsl?WSDL
```

For the same project in a Convertigo Cloud Server, the URL to retrieve the SOAP WSDL is:

```
http://me.convertigo.net/cems/projects/test_project/.wsl?WSDL
```

The project accessed through SOAP web service interface should declare at least one public transaction or sequence, with its schemas extracted. Refer to corresponding "Quick start tutorials" for more information about development and parametrization of such transactions and sequences.

4.2.2 REST web services

Any Convertigo project can be invoked as a REST web service, using the following URL:

```
http(s)://<ConvertigoServer>/<ConvertigoAppName>/projects/
<ProjectName>/.pxml
```

Where:

- ***ConvertigoServer*** is the host name or IP address of your Convertigo server, with possibly a port number**.**
  - A local Convertigo Studio will have as host name `localhost` and as default port number `18080`.
  - A Convertigo Server installed locally on your computer will share the same host name: `localhost`. When installed in an existing application server, it will have as IP address the IP address of the application server. The Convertigo Server default port number is `28080`.
  - A Convertigo Cloud server is accessed through its server name, for example:

`me.convertigo.net` (no port to provide in this case).

- ***ConvertigoAppName*** defines the name of the Convertigo web application.
  - A local Convertigo Studio or a Convertigo Server installed locally has `convertigo` as Convertigo app name.
  - A Convertigo Cloud server has `cems` as Convertigo app name.
- ***ProjectName*** defines the name of the project you want to invoke in Convertigo. If the project doesn't exist, the application server generates an error (`HTTP 404 Not Found`).

You can notice that this URL uses an **XML requester**; for more information about the other available requesters, see *"Convertigo URLs"* on page 4-2.

For more information about HTTP protocol to call transactions or sequences and the variables to pass to this request, see *"HTTP protocol interface to Convertigo"* on page 4-2.

4.2.3 Context state conservation

When using web services, you can have:

- ***Stateless web services***: when the response of a web service method does not depend on a previous state left by a preceding web service call. This is also called "atomic" web services.
- ***Stateful web services***: when the response of a web service method does depend on a previous state left by a preceding call. Convertigo projects mostly generate stateful web services. For example, you cannot get information from a legacy application or a web application if you don't call a "login" transaction before.

Consuming stateful web services, the programmer must maintain the HTTP session from its calling application to Convertigo between two methods calls. See the corresponding web service client documentation to do so.

A FEW EXAMPLES FOR SOAP CLIENTS

If you are using Java AXIS web service client, simply use the following line of code:

```
locator.setMaintainSession=true;
```

If your are using Microsoft .NET web service client, use this code construction (C# Syntax):

```
MyWs = new ConvertigoWebService();

// Creates an instance of the .NET WebServiceClient.

MyCookieJar = new CookieContainer();

// Creates an new Instance of a CookieContainer, this will hold the
HTTP Session Cookie.

MyWs.CookieContainer = fccCookieJar;

// Links the web service client with the cookie Container.
```

CASE OF REST CALLS

For REST web service calls, be sure that the HTTP client will maintain session cookies with Convertigo.

5 Convertigo Templating Framework

This chapter introduces the *Convertigo Templating Framework*, concepts and purpose.

- Convertigo Templating Framework presentation
- Launching a Convertigo requestable
- Listening for a C8O requestable response
- HTML templating
- Routing

5.1 Convertigo Templating Framework presentation

This section introduces the *Convertigo Templating Framework*:

- Objectives
- Templating system

5.1.1 Objectives

The *Convertigo Templating Framework* (CTF) aims at helping Convertigo programmers to execute transactions and/or sequences in a mobile or web application, and to provide them with a very efficient way to fill out data inside HTML pages.

The CTF also provides a *routing table* to automate the transition from one screen to another according to the result of Convertigo requestable calls. It aims at automatically and easily deal with errors.

The CTF follows the **MVC** (Model-View-Controller) model, where:

- Convertigo sequences provide the data (model),
- HTML templates provide the view,
- The routing table provides the controller.

A Convertigo sample project, named CTF gallery, lists all Convertigo Templating Framework features, shows pieces of code for each feature and uses the CTF to dynamically display the result.
*You can find the CTF gallery in your Convertigo Studio, in the new project wizard, choose **Convertigo Samples and Demos** > **Mobile samples** category.*
*The CTF gallery is also available in **Mobile demos** page in our Developer Network website: http://www.convertigo.com/en/demos/mobile-demos/395-demos-ctf-gallery.html*

5.1.2 Templating system

Inside a web/mobile page, the main idea is to insert special tags to handle the followings:

- Launch a Convertigo requestable (i.e. a transaction or a sequence),
- Register some HTML elements as listeners of specific requestables responses,
- Fill out HTML parts with the data returned in XML response of a requestable call,
- Update data from the requestable results each time the requestable is called (i.e. it can automatically update data).

5.2 Launching a Convertigo requestable

This section presents how the Convertigo programmer should use the *Convertigo Templating Framework* to easily call a Convertigo transaction or sequence:

5.2.1 C8O call - Calling transactions or sequences

The CTF hides all the complexity of calling Convertigo transactions or sequences, i.e. the Convertigo programmer does not have to write a single line of JavaScript to launch a Convertigo requestable.

REQUESTABLE CALL FORMAT

A C8O call is defined by adding the attribute `data-c8o-call` to any clickable HTML element.

We have a unique entry point, whether we call transactions or sequences.

A requestable will be addressed with the following syntax:

- For a transaction: `[project].connector.transaction`
  The project name is optional, i.e. if not specified, the current project will be used.
- For a sequence: `[project].sequence`
  The project name is optional, i.e. if not specified, the current project will be used.

When the CTF displays an HTML page, it searches for all C8O call declarations and adds click listeners on each element declaring a C8O call. In this way, a call is performed when a click event is launched on one of these elements.

- If the data-c8o-call attribute is declared on a clickable element, the call is performed when the item is clicked.

EXAMPLE

```
<button data-c8o-call="APIs.Login" id="loginButton" value="Login">
</button>
```

This will call the `Login` sequence of the `APIs` project.

- If the `data-c8o-call` attribute is added to a `<form>` element, the call is performed when the form is submitted (i.e after clicking on button or on submit type input, or hitting enter in any field).

EXAMPLE

```
<form id="formLogin" data-c8o-call="APIs.Login">
</form>
```

REQUESTABLE CALL AND VARIABLES

Any transaction or sequence call may require some parameters (i.e. variables).

The CTF automatically handles C8O call variables according to the following rules:

- If the `data-c8o-call` attribute is added to a `<form>` element, then all form elements (such as `<input>`, `<select>`, `<textarea>`...) declared within the `<form>` will be injected in the C8O call as requestable parameters (variables) when the form is submitted (i.e after clicking on button or on submit type input, or hitting enter in any field).

EXAMPLE

```
<form id="formLogin" data-c8o-call="APIs.Login">
        <label for="username">User name</label>
        <input type="text" id="username" name="username"/>
        <label for="password">Password</label>
        <input type="password" id="password" name="password"/>
        <button>Login</button>
</form>
```

In case of form submission through CTF, the parameters passed in the call to Convertigo requestable are named after the name attribute of the form elements, such as for a classic HTML form submission.

- If the `data-c8o-call` attribute is added to a node inside a `<form>` element, then all form elements (such as `<input>`, `<select>`, `<textarea>`...) declared within the `<form>` will be injected in the C8O call as requestable parameters (variables).

EXAMPLE

```
<form id="formLogin">
        <label for="username">User name</label>
        <input type="text" id="username" name="username"/>
        <label for="password">Password</label>
        <input type="password" id="password" name="password"/>
        <button data-c8o-call="APIs.Login">Login</button>
</form>
```

In this case, the submit event on the form is not intercepted and will perform a true submit. You may have to add a handler returning false to avoid the page from reloading.

- If both `<form>` and clickable element have a `data-c8o-call` attribute, the clickable element's attribute value is used in priority when clicked.

EXAMPLE

```
<form id="formLogin" data-c8o-call="APIs.Login">
        <label for="username">User name</label>
        <input type="text" id="username" name="username"/>
        <label for="password">Password</label>
        <input type="password" id="password" name="password"/>
        <button>Login</button>
        <button data-c8o-call="APIs.Regsiter">Register</button>
</form>
```

- If the node defining the `data-c8o-call` is **NOT** inside a `<form>` element, then all sub nodes of this node containing the `data-c8o-variable` attribute will be injected in the C8O call as a C8O requestable parameter (variable).

EXAMPLE

```
<a href="#" data-c8o-call=".GetDetails">
        Ticket ID: <span data-c8o-variable="ticketID">123456</span>
</a>
```

- To add several variables to a C8O call, simply add a `data-c8o-variables` attribute (note the ending 's') to the HTML node declaring the `data-c8o-call`. The `data-c8o-variables` attribute contains a JSON structure with several variables and their values.

EXAMPLE

```
<a href="#" data-c8o-call=".GetDetails"
data-c8o-variables='{"ticketID":"123456","price":"42"}'>
        Paris / Montpellier
</a>
```

Pay attention to the simple quotes/double quotes pairs: as a standard JSON object, it is mandatory that the variable names and values are surrounded by double quotes inside the structure. The data-c8o-variables attribute value is therefore surrounded by simple quotes.

- Variable declaration styles can be mixed in `<form>` element. A `data-c8o-call` attribute on an element inside a `<form>` element overrides the called transaction or sequence, in the same way, variables from `data-c8o-variables` attribute override the elsewhere declared variables:

EXAMPLE

```
<form id="formLogin" data-c8o-call="APIs.Login"
data-c8o-variables='{"__context":"ctxt1","key":"secret"}'>
        <label for="username" data-c8o-variable="userLabel">User
        name</label>
        <input type="text" id="username" name="username"
        value="John"/>
        <label for="password">Password</label>
        <input type="password" id="password" name="password"
        value="1234"/>
        <button>Login</button>
        <button data-c8o-call="APIs.Regsiter"
        data-c8o-variables='{"__context":"ctxt2"}'>Register</button>
</form>
```

Table 5 - 1: Expected result of above code example

	Click Login button	Click Register button
Call	Login sequence	Register sequence
__context	ctx1	ctx2
key	secret	secret
userLabel	User name	User name
username	John	John
password	1234	1234

- Variable styles can also be mixed in non-form element.

EXAMPLE

```
<a href="#" data-c8o-call=".GetDetails"
data-c8o-variables='{"ticketID":"123456","price":"42"}'>
        Paris / Montpellier
        Ticket ID: <span data-c8o-variable="ticketID">9999999</span>
</a>
```

When a variable is both declared in `data-c8o-variables` and `data-c8o-variable`, it is considered as a multi-valued variable: all values are sent to the requestable.

5.2.2 Call mode

A C8O call can be launched in three modes, thanks to the `data-c8o-call-mode` attribute that can be added to the HTML node declaring the `data-c8o-call`.

This attribute can take the following values:

- `click`: Default behavior (HTML element click or form submit).
- `auto`: The call is automatically triggered when the containing HTML element is first loaded, i.e. when the DOM is loaded, even if the element is not visible nor rendered by the CTF.

Pay attention that in jQuery Mobile framework, all pages are in the same HTML DOM. The auto mode would then be launched at the first start of the application, when all DOM is loaded.

- `timer:<number_of_seconds>`: This is the same as `auto` mode, but the call is triggered 'n' seconds after the document ready event.

EXAMPLES

- Default mode:

```
<a href="#" data-c8o-call=".GetList" data-c8o-call-mode="click">
        Logout
</a>
```

- Auto mode:

```
<span data-c8o-call=".GetList" data-c8o-call-mode="auto">
        Logout
</span>
```

- Timer mode, 30 seconds:

```
<span data-c8o-call=".GetList" data-c8o-call-mode="timer:30">
        Logout
</span>
```

5.2.3 Call condition

Sometimes we would like to trigger a C8O call on a condition. The CTF is able to operate conditional C8O calls by using the `data-c8o-call-condition` attribute (added to the HTML node declaring the `data-c8o-call`).

The `data-c8o-call-condition` attribute can be defined using one of the following expressions:

- a simple jQuery selector applied on the HTML DOM,
- a function name, the function should be declared with the following signature:

```
function () {
        // "this" equals to the HTML element containing the
        data-c8o-call attribute (DOM API)
        // returns true of false
}
```

The condition is considered as validated if the jQuery selector returns a non empty list, or if the JavaScript function returns `true`.

EXAMPLES

- Simple selector:

```
<a href="#" data-c8o-call=".GetList"
data-c8o-call-condition="h1:contains('1')">
        List of elements
</a>
```

- Function:

```
<a href="#" data-c8o-call=".GetDList"
data-c8o-call-condition="myConditionFunction">
        List of elements
</a>

<script>
        function myConditionFunction() {
          var $this = $(this);
          // work with $this...
          return true;
        }
</script>
```

5.2.4 Immediate action and call

In order to make the user feel like the mobile application is responsive, we sometimes want to be able to change page when a call is performed, before its response arrives. The CTF is able to perform an action (a change of page) when a C8O call is performed thanks to the `data-c8o-call-action` attribute (added to the HTML node declaring the `data-c8o-call`).

The `data-c8o-call-action` attribute takes as value a JSON structure of the following format:

```
{
        "goToPage": "<pageId>",
        "options": {
          ...
        }
}
```

Think about surrounding by quotes the keys (goToPage and options) and string values in your JSON structure!
Beware to use double quotes and not simple quotes as this is a standard JSON structure.
As this structure is put in an attribute, use simple quotes to delimit the attribute value.

This structure is a subset (limited to **goToPage** and **options** parameters) of what can be defined in an action of the routine table (see *"Routing table"* on page 5-46). To get the detailed definitions of the two **goToPage** and **options** parameters, see Table 5 - 2 on page 5-48.

EXAMPLE

```
<a href="#" data-c8o-call=".GetDetails"
data-c8o-variables='{"ticketID":"123456","price":"42"}'
data-c8o-call-action='{"goToPage": "#details",
                       "options": { "transition": "pop"}}'>
        Paris / Montpellier
        Ticket ID: <span data-c8o-variable="ticketID">9999999</span>
</a>
```

Do not forget that the **options** parameter contains a nested JSON structure.

5.2.5 Local cache on calls

Sometimes we would like to use local cache on C8O calls and responses, in order to:

- save network traffic between the device and the server,
- be able to display data when the device is not connected to the network.

The Local Cache feature allows to store locally on the device the responses to a C8O call, using the variables and their values as cache key.

To use the Local Cache, simply send the `__localCache` variable in the C8O call (added to the other call variables). This `__localCache` variable takes a JSON structure of the following format:

```
{
        "enabled":true/false,
        // allows to enable or disable the local cache on a Convertigo
requestable , default value is true

        "policy":"priority-server"/"priority-local",
        // defines whether the response should be retrieved from local
cache or from Convertigo server when the device can access the
network
        // when the device has no network access, the local cache
response is used, if existing
        // this parameter is mandatory as it has no default value

        "ttl":<time-to-live in ms>
        // defines the time to live of the cached response, in
milliseconds
        // if no value is passed, the time to live is infinite

}
```

Think about surrounding by quotes the keys (enabled, policy and ttl) and string values in your JSON structure!
Beware to use double quotes and not simple quotes as this is a standard JSON structure.

To get the detailed definition of the __**localCache** reserved parameter, see *"Weblib reserved parameters"* on page 4-11.

EXAMPLES

- Sending `__localCache` variable in an HTML input:

```
<form data-c8o-call=".Login">
        <input type="hidden" name="__localCache"
        value='{"enabled":true,
                "policy":"priority-server",
                "ttl":86400000}'
        />

        <div data-role="fieldcontain">
          <label for="userId">User </label>
          <input type="text" id="userId" name="user"
          value="username" placeholder="username"/>
        </div>

        <div data-role="fieldcontain">
          <label for="password">Password </label>
          <input type="password" id="password" name="password"
          value="password" placeholder="password"/>
        </div>

        <p id="info">Please type your username and password</p>

        <button data-theme="b" id="loginButton">Login</button>
</form>
```

In this case, the JSON structure is passed in an attribute value, so it has to be surrounded by quotes. As inside the JSON structure we already use double-quotes, we recommend to use simple quotes to surround the attribute value.

- Sending `__localCache` variable in a `data-c8o-variables` CTF attribute:

```
<a href="#" data-c8o-call=".GetDetails"
data-c8o-variables='{
        "id": "__=id__",
        "__localCache": {"enabled":true,
                         "policy": "priority-local",
                         "ttl": 120000}
}'>
        <h3>__=title__</h3>
</a>
```

In this case, the JSON structure is a value in another JSON structure, so it does not have to be surrounded by quotes. We can use the double-quotes inside the JSON structure as the `data-c8o-variables` attribute value is surrounded by simple quotes.

5.2.6 Non C8O-requestable calls

Third party web services are handled through sequences and HTTP connectors in Convertigo projects. Thus, they are then natively integrated into the CTF.

5.3 Listening for a C8O requestable response

This section presents how the Convertigo programmer should use the CTF to easily handle in a declarative way Convertigo transactions or sequences responses:

- Listener concept
- Listen condition
- Data accumulation

5.3.1 Listener concept

The CTF provides the `data-c8o-listen` attribute to declare that an HTML node is “interested” by the response of a given requestable.

SIMPLE LISTENER

The `data-c8o-listen` attribute follows the same rules as the `data-c8o-call` attribute concerning the requestable name (see *"Requestable call format"* on page 5-3).

EXAMPLE

```
<p data-c8o-listen=".OpenWorkOrder">Work order</p>
```

ANONYMOUS LISTENER

The `data-c8o-listen` attribute can define that the node listens to any requestable.

EXAMPLE

```
<p data-c8o-listen="*">Work order</p>
```

WILDCARD PATTERN LISTENER

The `data-c8o-listen` attribute can define that the node listens to requestables following a wildcard pattern.

EXAMPLES

- any sequence from the current project:

```
<p data-c8o-listen=".*">Work order</p>
```

- any sequence from the `APIs` project:

```
<p data-c8o-listen="APIs.*">Work order</p>
```

- any transaction from the `APIs` project and `MyConnector` connector:

```
<p data-c8o-listen="APIs.MyConnector.*">Work order</p>
```

- any transaction from the current project and MyConnector connector:

```
<p data-c8o-listen=".MyConnector.*">Work order</p>
```

MULTIPLE LISTENERS

The `data-c8o-listen` attribute can define that the node listens to several requestables. The requestable names (or patterns) separated by the comma character ','.

The list of listened requestables may contain space characters.

EXAMPLE

```
<div data-c8o-listen=".StopLabor, .GetLabors">
        ...
</div>
```

This will listen to both `StopLabor` and `GetLabor` sequences.

5.3.2 Listen condition

The CTF can handle nodes listening to a C8O call only if a condition is matching. A conditional C8O listen is defined by using the `data-c8o-listen-condition` attribute, added to the HTML node declaring the data-c8o-listen.

The `data-c8o-listen-condition` attribute can be defined using one of the following expressions:

- a simple jQuery selector applied on the received XML document,
- a function name, the function should be declared with the following signature:

```
function($doc, c8oData) {
        // $doc is the jQuery object of the response XML document
        // c8oData is the object containing key/value parameters sent
        to the request
        // "this" equals to the HTML element containing the
        data-c8o-listen attribute (DOM API)
        // returns true of false
}
```

The condition is considered as validated if the jQuery selector returns a non empty list, or if the JavaScript function returns `true`.

EXAMPLES

- Simple selector:

```
<div data-c8o-listen=".GetList"
data-c8o-listen-condition="items>item">
        ...
</div>
```

- Function:

```
<div data-c8o-listen=".GetList"
data-c8o-listen-condition="myConditionFunction">
        ...
</div>

<script>
        function myConditionFunction($doc, c8oData) {
          var $this = $(this);
          // work with $this, $doc...
          return true;
        }
</script>
```

5.3.3 Data accumulation

The CTF can accumulate listen data in a node that has already been templated using the `data-c8o-accumulate` attribute, added to the HTML node declaring the data-c8o-listen.

This can be useful for filling dynamically a table page per page for instance.

The `data-c8o-accumulate` attribute can take two values:

- `append`: will accumulate new data at the end of existing data,
- `prepend`: will accumulate new data at the beginning of existing data.

EXAMPLE

```
<div data-c8o-listen=".GetLabors" data-c8o-accumulate="append">
        ...
</div>
```

This will listen to `GetLabor` sequence but accumulate data instead of replacing data. This can be useful for filling dynamically a table page per page for instance.

5.4 HTML templating

Once an HTML element is listening to one or more requestable responses, we can construct the HTML with data coming from the requestable XML response.

This section presents how the Convertigo programmer should use the powerful declarative patterns of the *Convertigo Templating Framework* to easily fill out data from the requestables XML responses in the HTML document:

- Different types of patterns
- Simple templating
- Nested listeners
- Conditional templating
- Iterative templating
- Nested iterations
- References use
- Late rendering
- Before rendering callback
- After rendering callback
- Inline templating

5.4.1 Different types of patterns

The CTF provides several types of patterns that can be used to select XML data onto which to work, to display data in HTML elements, to decide whether or not to render a part of HTML. These different types of patterns are presented in this section:

- Templating patterns
- Selecting patterns

TEMPLATING PATTERNS

Templating patterns are those which can be used for rendering data inside HTML code. Basically, templating will be achieved by substituting the patterns by matching XML data.

These patterns are one of the following forms:

- **Simple templating pattern:** `__=<jQuery selector>__`

This pattern is composed of a jQuery selector surrounded by a "double underscore and equals" character at the begining, and a "double underscore" character at the end.

It refers to the simple data pointed by the jQuery selector and allows to display it as a string value. It provides an empty string if no result is found in the XML data.

Notes:

- The current element can be selected by selector `"."`.
- In the case of a node list matching the selector, the simple templating pattern displays

the text value of the first element of the node list.

If you need to use a "double underscore" character inside a simple templating pattern, you simply need to escape the underscore characters using back-slash characters: `\_\_`
For example, using the tag name `my__N_stand` *in a simple templating pattern would be:* `__=my\_\_N_stand__`

- **Enhanced JSON structure templating pattern:** `__{<enhanced JSON selection structure>}__`

This pattern is composed of a JSON structure surrounded by a "double underscore and opening brace" character at the begining, and a "closing brance and double underscore" character at the end.

TEMPLATING PATTERN JSON STRUCTURE SPECIFICATION:

```
__{
   "mode": "<find|index>",
```

// Default mode is `"find"` and enables the `"find"`, `"attr"`, `"default"`, `"formatter"` and `"ref"` options.
// The `"index"` mode returns the current iteration index in a `data-c8o-each` template and enables the `"formatter"` and `"ref"` options.

MODE FIND

```
   "find": "<jquery selector>",
```

// Combines text contents of all elements in the set of matched elements, including their descendants. The current element can be selected by selector `"."`.

```
   "attr": "<attribute name>",
```

// Attribute value of the first element of the found elements.

```
   "default": "<default value>",
```

// Default value if no results.

```
   "formatter": "<formatting function>",
```

// Name of the JS function to call in order to format data. See formatting function signature below.

```
   "ref": "<reference name>",
```

// Should contain the name of an ancestor `data-c8o-ref` anchor (see *"References use"* on page 5-33). The `"find"` or `"index"` mode will use the referenced XML data as current context.
// Be aware that the reference name used here can only refer to a reference from the same XML response. References from a response are kept until a new XML response arrives.

```
   "type": "<string|fragment>",
```

// The `"fragment"` type allows to copy the selected elements (as found by the `"find"` option) and their children as an HTML DOM fragment.
// Default type is `"string"`.

MODE INDEX

```
"formatter": "<formatting function>",
```

// Name of the JS function to call in order to format data. See formatting function signature below.

```
"ref": "<reference name>",
```

// Should contain the name of an ancestor `data-c8o-ref` anchor (see *"References use"* on page 5-33). The `"find"` selector or `"index"` mode will use the referenced XML data as current context.
// Be aware that the reference name used here can only refer to a reference from the same XML response. References from a response are kept until a new XML response arrives.

```
}__
```

Use double quotes and not simple quotes in your JSON structure! If you need to put this structure in a node attribute, use simple quotes to delimit the attribute value and double quotes in the enhanced JSON structure.

FORMATTING FUNCTION SIGNATURE

The formatting function must follow this signature:

```
function(value) {
	// "this" object provides the current DOM element, i.e. the
	DOM element containing the templating pattern

	// "value" parameter provides the current node value
	(string or fragment, depending on the selected type)

	// With string type, the function should return the new text
	to display.

	// With fragment type, the value is a "fragment" element
	containing the "find" result. Return isn't mandatory.
	Returning a string injects it instead of the fragment.
}
```

EXAMPLES

Examples of templating patterns are present in *"Simple templating"* on page 5 - 18 and in every following section of HTML templating chapter.

SELECTING PATTERNS

Selecting patterns are those which can be used to select a list of nodes onto which to work or to decide whether or not to render a part of HTML. This type of selector is used in `data-c8o-each` attribute to select the iterated elements and in `data-c8o-if` and `data-c8o-if-not` attributes to test the presence of elements.

Notes:

- In `data-c8o-each`, the current iterated item inside the selector's result becomes the reference element for templating,
- in `data-c8o-if`, the selector result does not become the reference element for templating.

Selecting patterns can be of two types :

- **Simple selecting pattern:** `<jQuery selector>`

This pattern is composed of a jQuery selector.

It allows selecting a node list (possibly containing only one element) pointed by the jQuery selector. It provides an empty node list if no result is found in the XML data.

Note: The current element can be selected by selector `"."`.

- **Enhanced JSON structure selecting pattern:** `{<enhanced JSON selection structure>}`

This pattern is composed of a JSON structure: key/value string pairs surrounded by braces.

SELECTING PATTERN JSON STRUCTURE SPECIFICATION:

```
{
   "find": "<jquery selector>",
// Selects a node list of matched elements, including their descendants.
   "ref": "<reference name>",
// Should contain the name of an ancestor data-c8o-ref anchor (see "References use" on page 5-33). The "find" selector will use the referenced XML data as current context.
// Be aware that the reference name used here can only refer to a reference from the same XML response. References from a response are kept until a new XML response arrives.
   "mode": "find",
// Default mode is "find" for this type of pattern. "index" mode would not give any result.
}
```

EXAMPLES

Examples of selecting patterns are present in *"Conditional templating"* on page 5 - 23 and in *"Iterative templating"* on page 5 - 28.

5.4.2 Simple templating

The simple templating has as effect to render data selected from XML response inside HTML elements. Templating is achieved by substititing a *templating pattern* (see *"Templating patterns"* on page 5-15) by the matching XML data turned into a string.

Simple templating can be used inside any part of HTML code, on condition that a requestable response is listened by a parent node, i.e. that a `data-c8o-listen` attribute is declared on a parent element in HTML document (see *"Listening for a C8O requestable response"* on page 5-12).

EXAMPLES

- Simple templating with a simple templating pattern:

```
<div data-c8o-listen=".GetData">
        <span>__=myValue__</span>
</div>
```

This example declares a div HTML element listening to the `GetData` sequence of the current project. When the sequence returns, the CTF will substitute `__=myValue__` by the value retrieved from the XML response by applying this jQuery selector. If not matching in the XML response, the span is filled with an empty value.

With the following XML response:

```
<document …>
        <myNum>12.45</myNum>
        <myValue>This is a sample text</myValue>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span>This is a sample text</span>
</div>
```

- Simple templating combining several simple templating patterns:

```
<h1 data-c8o-listen=".OpenWorkOrder">
        Work order <strong>#__=id__ - __=description__</strong>
</h1>
```

This template declares an `H1` HTML title element listening to the `OpenWorkOrder` sequence of the current project. When the `OpenWorkOrder` sequence returns an XML response, the CTF will substitute `__=identifiant__` and `__=description__` by their respective values issued from JQuery selectors.

With the following XML response:

```
<document …>
        <work_order>
          <id>123456</id>
          <description>First work order</description>
        </work_order>
</document>
```

The CTF will produce the following HTML:

```
<h1>
        Work order <strong>#123456 - First work order</strong>
</h1>
```

- Simple templating with a JSON structure templating pattern using a formatter function:

```
<div data-c8o-listen=".GetData">
        <span>__{"find":"myNum", "formatter":"formatNumber"}__
        </span>
</div>

<script>
        function formatNumber(value) {
          return value.replace(".", ",") + " €";
        }
</script>
```

This example declares a div HTML element listening to the `GetData` sequence of the current project. When the sequence returns, the CTF will display in a span, the `myNum` node value after it has been changed by the `formatNumber` function.

With the following XML response:

```
<document …>
        <myNum>12.45</myNum>
        <myValue>This is a sample text</myValue>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span>12,45 €</span>
</div>
```

- Simple templating combining several templating patterns (a simple templating pattern and a JSON structure using formatter function):

```
<div data-c8o-listen=".GetData">
        <span id='__{"find":"myNum","formatter":"getId"}__'>
          __=myValue__
        </span>
</div>

<script>
        function getId(value) {
          if (value > 10)
            return "bigBlock";
          else
            return "littleBlock";
        }
</script>
```

This example declares a div HTML element listening to the `GetData` sequence of the current project. When the sequence returns, the CTF will display the `myValue` node value in a span with a class name computed from the `myNum` node value using the `getClass` function.

With the following XML response:

```
<document …>
        <myNum>12.45</myNum>
        <myValue>This is a sample text</myValue>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span id="bigBlock">This is a sample text</span>
</div>
```

5.4.3 Nested listeners

Nested listeners that would like to mix data are not recommended.

Each listener block should render data from the result it listens to, otherwise when rendering the second data, the first one may be lost.

In case of HTML parts mixing data from several requestables, prefer using several serial listeners.

Nested listeners using separated scope can work correctly.

EXAMPLE

The following template will not fully work as it mix data from two nested listeners:

```
<div data-c8o-listen=".GetData">
    <p>__=myValue__</p>
    <div data-c8o-listen=".OpenWorkOrder">
       <span id='__{"find":"myNum", "formatter":"getId"}__'>
          <p>#__=id__ - __=description__</p>
       </span>
    </div>
</div>
```

In this example, depending on which XML response arrives first, the second arriving response's rendering will overwrite the first response's rendering.

Assuming that the `GetData` response XML arrives first, with the following XML response:

```
<document …>
       <myNum>12.45</myNum>
       <myValue>This is a sample text</myValue>
</document>
```

The p with `myValue` node and the `id` attribute computed from `myNum` node are rendered in the HTML.

When the `OpenWorkOrder` response XML arrives, with the following XML response:

```
<document …>
       <work_order>
          <id>123456</id>
          <description>First work order</description>
       </work_order>
</document>
```

The p with `id` and `description` nodes is rendered. But the `id` attribute using `myValue` node is also re-rendered, as it belongs to the listening `div`. As no `myValue` node is present in the `OpenWorkOrder` response XML, it will be rendered empty.

A way of listening to both sequences responses using nested listeners with separated scopes could be:

```
<div data-c8o-listen=".GetData">
       <p>__=myValue__</p>
       <span id='__{"find":"myNum", "formatter":"getId"}__'>
          <p data-c8o-listen=".OpenWorkOrder">
             #__=id__ - __=description__
          </p>
       </span>
</div>
```

In this case, it works only if the `GetData` response arrives before the `OpenWorkOrder` response. Otherwise, a similar problem of losing rendered data will occur.

The preferred way of listening to both sequences responses separately is:

```
<div data-c8o-listen=".GetData">
        <p>__=myValue__</p>
        <span id='__{"find":"myNum", "formatter":"getId"}__'>
          Something to display here
        </span>
</div>
<p data-c8o-listen=".OpenWorkOrder">
        #__=id__ - __=description__
</p>
```

In this case, the `id` attribute cannot be computed from a sequence's response and the data inside it to be retrieved from another sequence's response. It can be done using other CTF methods but not simple listeners and simple templating.

5.4.4 Conditional templating

The conditional templating allows the developer to make a test in order to decide whether or not to render a part of HTML.

Conditional templating attributes are presented in this section:

- CTF If
- Negative if
- Several conditions

CTF IF

The `data-c8o-if` attribute can be added to any HTML element in order to perform a test. It can contain:

- a *selecting pattern* (see *"Selecting patterns"* on page 5-17): it is a simple jQuery selector or a JSON structure pattern limited to `find` and `ref` ;
- or a function name.

If the condition present in the `data-c8o-if` attribute is validated, the piece of HTML inside the element containing the `data-c8o-if` attribute is rendered.

Otherwise, if the condition is not validated, the element containing the `data-c8o-if` attribute (and all sub HTML elements) is removed from the DOM.

CASE OF SELECTING PATTERN

When using a selecting pattern inside the `data-c8o-if` attribute, the condition is validated when the selector returns a non empty list.

Otherwise, if the list is empty, the condition is not validated.

Note: If the condition is validated, the list resulting from the selector does not become the reference for elements rendered inside: the parent `data-c8o-listen` or `data-c8o-each` remains the reference element for templating.

Case of Function

When using the function name inside the `data-c8o-if` attribute, the condition is validated if the function returns `true`.

Otherwise, if it returns `false`, the condition is not validated.

The function should have the following signature:

```
function($doc, refs) {
	// "this" object provides the current DOM element, i.e. the
	DOM element containing the data-c8o-if attribute

	//  $doc is the jQuery object of the XML node currently
	referenced for rendering

	// refs is an object containing all XML references declared
	in data-c8o-ref attributes
	// refs object contains key/value pairs, key being the
	reference name, and value being the jQuery object of matching
	XML

	// returns true or false
}
```

Examples

- Conditional templating with simple selecting pattern:

```
<div data-c8o-listen=".GetData">
	<span data-c8o-if="key">
	  <p>__=value__</p>
	</span>
</div>
```

This example declares a div HTML element listening to the `GetData` sequence of the current project. When the sequence returns, if the jQuery selector is matching in the XML response, i.e. if `key` node is present, the `span` and the p elements will be rendered. If not matching in the XML response, the `span` is removed from its parent `div`.

With the following XML response:

```
<document …>
        <data>
          <key>123456</key>
          <value>This is a standard value</value>
        </data>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span>
          <p>This is a standard value</p>
        </span>
</div>
```

- Conditional templating with JSON structure selecting pattern:

```
<div data-c8o-listen=".GetData">
        <span data-c8o-if='{"find":"key"}'>
          <p>__=value__</p>
        </span>
</div>
```

This example declares a div HTML element listening to the `GetData` sequence of the current project. When the sequence returns, if the JSON structure selecting pattern is matching in the XML response, i.e. if `key` node is present, the `span` and the p elements will be rendered. If not matching in the XML response, the `span` is removed from its parent `div`.

This would give the same exact result as the preceding example.

- Conditional templating with a function:

```
<div data-c8o-listen=".GetData">
        <span data-c8o-if="myDecidingFunction">
          <p>__=value__</p>
        </span>
</div>

<script>
        function myDecidingFunction($doc, refs) {
          if ($doc.find("key").length)
            return true;
          return false;
        }
</script>
```

This example declares a div HTML element listening to the `GetData` sequence of the current project. When the sequence returns, if the `myDecidingFunction` function returns `true`, i.e. if `key` node is present in the response, the `span` and the `p` elements will be rendered. If the function returns `false`, the `span` is removed from its parent `div`.

This would give the same exact result as the preceding example.

NEGATIVE IF

The `data-c8o-if-not` attribute works the same way as the `data-c8o-if` attribute but the condition is validated in the inverted case as for the `data-c8o-if`.

CASE OF SELECTING PATTERN

When using a selecting pattern inside the `data-c8o-if-not` attribute, the condition is validated when the selector does not match in the XML response.

Otherwise, if the selector matches a node list in the XML response, the condition is not validated.

CASE OF FUNCTION

When using the function name inside the `data-c8o-if-not` attribute, the condition is validated if the function returns `false`.

Otherwise, if it returns `true`, the condition is not validated.

EXAMPLE

This example completes the first example of the conditional templating (see *"Examples"* on page 5-24).

```
<div data-c8o-listen=".GetData">
        <span data-c8o-if="key">
          <p>__=value__</p>
        </span>
        <span data-c8o-if-not="key">
          <p>No valid data to display</p>
        </span>
</div>
```

A second `span` HTML element declares a `data-c8o-if-not` condition, with the same jQuery selector as the `data-c8o-if`: it manages the inverse case, when the `key` node is not present in the XML response.

When the sequence returns, if `key` node is present, the `span` with the `p` element containing the `value` node content will be rendered. If `key` node is not present in the XML response, the second `span` with the `p` element containing a default text is rendered.

With the following XML response:

```
<document …>
        <data>
          <key>123456</key>
          <value>This is a standard value</value>
        </data>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span>
          <p>This is a standard value</p>
        </span>
</div>
```

With the following XML response:

```
<document …>
        <data/>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span>
          <p>No valid data to display</p>
        </span>
</div>
```

SEVERAL CONDITIONS

When several conditions need to be validated on a same HTML element, a Convertigo developer can use two methods:

- One `data-c8o-if` attribute and one `data-c8o-if-not` attribute can be declared on a same element. They are two different attributes, HTML only prevents from using a same attribute several times in the same element. In this case, both conditions have to be validated in order to render the internal elements. If one of the conditions is not validated, the whole element is removed from the DOM.
- If the conditions you need to combine are more complex, simply use the function name in the `data-c8o-if` attribute. In JavaScript code, all tests can be combined.

5.4.5 Iterative templating

The iterative templating allows the developer to build a table, a list, or more generally a structure based on an iterative build process. To do so, the CTF defines the `data-c8o-each` attribute, which can contain a *selecting pattern* (see *"Selecting patterns"* on page 5-17): it is a simple jQuery selector or a JSON structure pattern limited to `find` and `ref`. The iteration is performed on the list of nodes resulting from the application of the selector on the XML response.

Notes:

- The currently iterated node becomes the reference XML jQuery object for elements rendered in HTML sub elements.
- Inside the iteration, use the "`.`" selector to refer to this currently iterated node.

EXAMPLE

Let's have the following template:

```
<div data-c8o-listen=".GetLabors">
        <table>
          <thead>
            <tr>
              <th>Task</th>
              <th>Labor</th>
              <th>Name</th>
              <th>Approved</th>
            </tr>
          </thead>
          <tbody data-c8o-each="labors>labor">
            <tr>
              <td><span class="__=type__">__=task__</span></td>
              <td>__=labor__</td>
              <td>__=name__</td>
              <td>__=approved__</td>
            </tr>
          </tbody>
        </table>
</div>
```

With the following XML reponse:

```
<document …>
        <labors>
          <labor>
            <type>type1</type>
            <task>Task for engine</task>
            <labor>engine management</labor>
            <name>Labor1</name>
            <approved>true</approved>
          </labor>
          <labor>
            <type>type2</type>
            <task>Task for house</task>
            <labor>house management</labor>
            <name>Labor2</name>
            <approved>false</approved>
          </labor>
        </labors>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <table>
          <thead>
            <tr>
              <th>Task</th>
              <th>Labor</th>
              <th>Name</th>
              <th>Approved</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td><span class="type1">Task for engine</span></td>
              <td>engine management</td>
              <td>Labor1</td>
              <td>true</td>
            </tr>
            <tr>
              <td><span class="type2">Task for house</span></td>
              <td>house management</td>
              <td>Labor2</td>
              <td>false</td>
            </tr>
          </tbody>
        </table>
</div>
```

5.4.6 Nested iterations

Nested iterations are allowed and help handling responses that provide multi dimensional data, i.e. iteration can be handled for possibly multidimensional list and arrays. To do so, a `data-c8o-each` attribute can be used inside another `data-c8o-each` attribute, and can contain a *selecting pattern* (see *"Selecting patterns"* on page 5-17) with a selector **relative** to the current iterated item from parent `data-c8o-each`.

Beware that inside the nested iteration, selectors can only be relative to the current iteration.
In the example below, inside `data-c8o-each="labors>labor"`, no templating pattern can directly access to data from the parent iteration `data-c8o-each="tasks>task"`.
To access to data from parent iteration, you need to use references (see** "References use" **on page 5-33).

EXAMPLE

Let's have the following template:

```
<div data-c8o-listen=".GetTasks">
        <table>
          <thead>
            <tr>
              <th>Task</th>
              <th>Labors</th>
              <th>Approved</th>
            </tr>
          </thead>
          <tbody data-c8o-each="tasks>task">
            <tr>
              <td>
                <span>__=task_type__ - __=task_name__</span>
              </td>
              <td data-c8o-each="labors>labor">
                <span>__=labor_type__ - __=labor_name__</span>
              </td>
              <td>__=approved__</td>
            </tr>
          </tbody>
        </table>
</div>
```

With the following XML reponse:

```
<document …>
    <tasks>
       <task>
         <task_type>type1</task_type>
         <task_name>Task for engine</task_name>
         <labors>
           <labor>
             <labor_type>engine management</labor_type>
             <labor_name>Labor1</labor_name>
           </labor>
           <labor>
             <labor_type>engine management</labor_type>
             <labor_name>Labor2</labor_name>
           </labor>
         </labors>
         <approved>true</approved>
       </task>
       <task>
         <task_type>type2</task_type>
         <task_name>Task for house</task_name>
         <labors>
           <labor>
             <labor_type>house management</labor_house>
             <labor_name>Labor3</labor_name>
           </labor>
           <labor>
             <labor_type>house management</labor_house>
             <labor_name>Labor4</labor_name>
           </labor>
         </labors>
         <approved>false</approved>
       </task>
    </tasks>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <table>
          <thead>
            <tr>
              <th>Task</th>
              <th>Labors</th>
              <th>Approved</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>
                <span>type1 - Task for engine</span>
              </td>
              <td>
                <span>engine management - Labor1</span>
                <span>engine management - Labor2</span>
              </td>
              <td>true</td>
            </tr>
            <tr>
              <td>
                <span>type2 - Task for house</span>
              </td>
              <td>
                <span>house management - Labor3</span>
                <span>house management - Labor4</span>
              </td>
              <td>false</td>
            </tr>
          </tbody>
        </table
</div>
```

5.4.7 References use

A reference anchor can be declared using the `data-c8o-ref` attribute. It can be added on the same element as a `data-c8o-listen` or `data-c8o-each` attribute. The `data-c8o-ref` attribute:

- declares an anchor name which is the value set in the `data-c8o-ref` attribute,
- and saves the current context XML element (i.e. the XML element used as current XML for the `data-c8o-listen` or `data-c8o-each` attribute next to which it is declared) for further use.

When templating an iteration, if you need to refer to an element of a parent listen or a parent

iteration (in case of nested iterations), a declared reference can be used in a JSON structure templating pattern: the selector must contain the `ref` key with the name of the reference you want to use.

A `data-c8o-each` can also refer an ancestor `data-c8o-ref` in its selector, using a a JSON structure selecting pattern.

Be aware that the reference name used in a pattern can only refer to a reference from the same XML response.
References from a response are kept until a new XML response arrives.

EXAMPLES

- Reference use in templating pattern:

Let's have the following template:

```
<div data-c8o-listen=".GetLabors" data-c8o-ref="labors">
    <table>
       <thead>
         <tr>
           <th>Task</th>
           <th>Labor</th>
         </tr>
       </thead>
       <tbody data-c8o-each="labors>labor">
         <tr title='For
           __{"ref":"labors", "find":"details>name"}__'>
           <td><span class="__=type__">__=task__</span></td>
           <td>__=labor__</td>
         </tr>
       </tbody>
    </table>
</div>
```

With the following XML:

```
<document …>
        <details>
          <name>Monday labors</name>
          <count>2</count>
        </details>
        <labors>
          <labor>
            <type>type1</type>
            <task>Task for engine</task>
            <labor>engine management</labor>
            <name>Labor1</name>
            <approved>true</approved>
          </labor>
          <labor>
            <type>type2</type>
            <task>Task for house</task>
            <labor>house management</labor>
            <name>Labor2</name>
            <approved>false</approved>
          </labor>
        </labors>
</document>
```

The CTF will produce the following HTML:

```
<div>
    <table>
        <thead>
          <tr>
            <th>Task</th>
            <th>Labor</th>
          </tr>
        </thead>
        <tbody>
          <tr tile="For Monday labors">
            <td><span class="type1">Task for engine</span></td>
            <td>engine management</td>
          </tr>
          <tr tile="For Monday labors">
            <td><span class="type2">Task for house</span></td>
            <td>house management</td>
          </tr>
        </tbody>
    </table>
</div>
```

- Reference use in selecting pattern:

Let's have the following template:

```
<div data-c8o-listen=".GetLabors" data-c8o-ref="labors">
    <table>
       <thead>
         <tr>
            <th>Title</th>
            <th>Task</th>
            <th>Labor</th>
         </tr>
       </thead>
       <tbody data-c8o-each="labors>labor">
         <tr>
            <td data-c8o-each='{"ref":"labors",
                                          "find":">details>*"}'>
               <span>__=.__</span><br/>
            </td>
            <td><span class="__=type__">__=task__</span></td>
            <td>__=labor__</td>
         </tr>
       </tbody>
    </table>
</div>
```

With the same XML as previous example:

```
<document …>
        <details>
          <name>Monday labors</name>
          <count>2</count>
        </details>
        <labors>
          <labor>
            <type>type1</type>
            <task>Task for engine</task>
            <labor>engine management</labor>
            <name>Labor1</name>
            <approved>true</approved>
          </labor>
          <labor>
            <type>type2</type>
            <task>Task for house</task>
            <labor>house management</labor>
            <name>Labor2</name>
            <approved>false</approved>
          </labor>
        </labors>
</document>
```

The CTF will produce the following HTML:

```
<div>
    <table>
       <thead>
         <tr>
            <th>Title</th>
            <th>Task</th>
            <th>Labor</th>
         </tr>
       </thead>
       <tbody>
         <tr>
            <td>
               <span>Monday labors</span><br/>
               <span>2</span><br/>
            </td>
            <td><span class="type1">Task for engine</span></td>
            <td>engine management</td>
         </tr>
         <tr>
            <td>
               <span>Monday labors</span><br/>
               <span>2</span><br/>
            </td>
            <td><span class="type2">Task for house</span></td>
            <td>house management</td>
         </tr>
       </tbody>
    </table>
</div>
```

5.4.8 Late rendering

The CTF enables a late rendering feature, which allows to render parts of HTML marked as delayed only when a user click on a specific element happens (and not when the response arrives).

To do so, the CTF declares the `data-c8o-late-render` and `data-c8o-render` attributes. These attributes work by pair and are linked if they contain the same value. This value can be first rendered by the CTF (i.e. interpreted from a pattern, for example `__=selector__`).

How does it work ?

The `data-c8o-late-render` attribute marks an element as delayed to be rendered. The CTF finds these elements and removes their content from the HTML DOM.

When an element with a `data-c8o-render` attribute containing the same value is clicked, the CTF:

- retrieves the content of the element marked as delayed,
- performs a rendering on it with the initial XML context element,
- and displays it in the HTML.

This late rendering can be very useful in case of very large data: the browser can be very slow if too many elements are present, so a late rendering can limit the amount of elements to render (for instance by JQuery Mobile UI library) at a given time.

EXAMPLES

- Simple late rendering:

Let's have the following template:

```
<div data-c8o-listen=".GetDetails">
        <div data-c8o-render="displayMore">
          <b>Display more for:<b/>__=name__
        </div>
        <div data-c8o-late-render="displayMore">
          <div><b>Details:</b><br/>
          __=details__</div>
        <div>
</div>
```

With the following XML:

```
<document …>
        <name>Monday labors</name>
        <details>This is the details for monday labors.</details>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <div data-c8o-render="displayMore">
          <b>Display more for:<b/>Monday labors
        </div>
        <div data-c8o-late-render="displayMore">
        </div>
</div>
```

When the user clicks on the "Display more for: Monday labors" div, the CTF updates the HTML with the following changes:

```
<div>
        <div data-c8o-render="displayMore">
          <b>Display more for:<b/>Monday labors
        </div>
        <div data-c8o-late-render="displayMore">
          <div><b>Details:</b><br/>
          This is the details for monday labors.</div>
        <div>
</div>
```

- Late rendering inside an iteration:

Let's have the following template:

```
<div data-c8o-each="entries>entry">
        <div data-c8o-render='displayMore___{"mode":"index"}__'>
            <b>Display more for:<b/>__=name__
        </div>
        <div data-c8o-late-render=
                              'displayMore___{"mode":"index"}__'>
          <div><b>Details:</b><br/>
          __=details__</div>
        <div>
</div>
```

The `index` mode is used to generate a unique attribute value per iteration and link the `data-c8o-render` and `data-c8o-late-render` attributes by pairs.

With the following XML:

```
<document …>
    <entries>
       <entry>
         <name>Monday labors</name>
         <details>This is the details for monday labors.</details>
       </entry>
       <entry>
         <name>Tuesday labors</name>
         <details>This is the details for tuesday labors.</details>
       </entry>
       <entry>
         <name>Thursday labors</name>
         <details>This is the details for thursday labors.</details>
       </entry>
    </entries>
</document>
```

The CTF will produce the following HTML:

```
<div>
       <div data-c8o-render="displayMore_1">
         <b>Display more for:<b/>Monday labors
       </div>
       <div data-c8o-late-render="displayMore_1">
       </div>
       <div data-c8o-render="displayMore_2">
         <b>Display more for:<b/>Tuesday labors
       </div>
       <div data-c8o-late-render="displayMore_2">
       </div>
       <div data-c8o-render="displayMore_3">
         <b>Display more for:<b/>Thursday labors
       </div>
       <div data-c8o-late-render="displayMore_3">
       </div>
</div>
```

When the user clicks on the "Display more for: Tuesday labors" div, corresponding to `displayMore_2` attribute value, the CTF updates the HTML with the following changes:

```
<div>
        <div data-c8o-render="displayMore_1">
          <b>Display more for:<b/>Monday labors
        </div>
        <div data-c8o-late-render="displayMore_1">
        </div>
        <div data-c8o-render="displayMore_2">
          <b>Display more for:<b/>Tuesday labors
        </div>
        <div data-c8o-late-render="displayMore_2">
          <div><b>Details:</b><br/>
          This is the details for tuesday labors.</div>
        </div>
        <div data-c8o-render="displayMore_3">
          <b>Display more for:<b/>Thursday labors
        </div>
        <div data-c8o-late-render="displayMore_3">
        </div>
</div>
```

5.4.9 Before rendering callback

The `data-c8o-before-rendering` attribute provides a way to manipulate XML response from C8O call before data rendering. It contains the name of a JavaScript function that is executed at response reception.

The `data-c8o-before-rendering` attribute can only be added on the same element as a `data-c8o-listen` attribute and the function is executed only if the `data-c8o-listen-condition` is validated.

The before rendering function must follow this signature:

```
function onBeforeRendering($doc, c8oData){
        // $doc is the jQuery object of the C8O XML response

        // c8oData is an object containing all parameters sent to the
        request
        // c8oData object contains key/value pairs, key being the
        variable/parameter name, value being the value

        // "this" is the DOM element containing the data-c8o-listen
        attribute
}
```

EXAMPLE

```
<div data-c8o-listen=".GetList"
data-c8o-before-rendering="onBeforeRendering">
        ...
</div>

<script>
        function onBeforeRendering($doc, c8oData) {
          var $this = $(this);
          // work with $this, $doc...
        }
</script>
```

5.4.10 After rendering callback

After the CTF has rendered all the templates, it is sometimes useful to execute some additional code, for instance to refresh a GUI component such as JQuery Mobile ones. Standard "after rendering" callbacks are included in the CTF in order to handle all "standard" JQuery Mobile widgets.

For other purpose, the CTF adds a `data-c8o-after-rendering` attribute that provides a way to manipulate HTML DOM after the CTF has rendered data. It contains the name of a JavaScript function that is executed after rendering.

The after rendering function must follow this signature:

```
function onAfterRendering($doc, c8oData){
        // $doc is the jQuery object of the C8O XML response

        // c8oData is an object containing all parameters sent to the
        request
        // c8oData object contains key/value pairs, key being the
        variable/parameter name, value being the value

        // "this" is the DOM element containing the data-c8o-listen
        attribute
}
```

EXAMPLE

```
<div data-c8o-listen=".GetList"
data-c8o-after-rendering="onAfterRendering">
        ...
</div>

<script>
        function onAfterRendering($doc, c8oData) {
          // work with HTML DOM...
        }
</script>
```

5.4.11 Inline templating

Sometimes HTML attributes have to be rendered with dynamic data: at design time (before rendering), the content will not be valid HTML. Especially in case of HTML templates used in iterations, the HTML nodes of the templates are present only for the CTF, but not for a GUI library such as JQuery mobile.

These attributes should not be really declared at design time, but only declared at run-time, when rendered.

To do so, you can use the `data-c8o-use-<xxx>` attribute, `<xxx>` being the attribute name you want to dynamically declare at run-time.

Once the CTF has rendered elements according to the received XML data, the `data-c8o-use-<xxx>` attributes are rendered but replacing them with the real attribute.

When using `data-c8o-use-<xxx>` attribute, do not forget to add the `data-c8o-use` marker attribute (`data-c8o-use="on"`). This special attribute allows the CTF to quickly find nodes with `data-c8o-use-<xxx>` attributes.

EXAMPLE

- Simple `data-c8o-use`:

```
<div data-c8o-listen=".GetData">
        <div data-c8o-use="on" data-c8o-use-class="surrounded"/>
</div>
```

This will render as:

```
<div data-c8o-listen=".GetData">
        <div class="surrounded"/>
</div>
```

- Rendered value with `data-c8o-use`:

```
<div data-c8o-listen=".GetData">
        <span data-c8o-use="on"
data-c8o-use-<class>='__{"find":"myNum","formatter":"getClass"}__'>
          __=myValue__
        </span>
</div>

<script>
        function getClass(value) {
          if (value > 10)
            return "black";
          else
            return "red";
        }
</script>
```

In this example, a `class` attribute needs to be computed with data received from XML response. Before rendering with data from the `GetData` sequence, the `class` attribute is not valid: it contains a templating pattern. This is why the `data-c8o-use` attribute is used. When the sequence returns, the CTF will render and replace the attribute with a valid class name computed using the `getClass` function.

With the following XML response:

```
<document …>
        <myNum>12.45</myNum>
        <myValue>This is a sample text</myValue>
</document>
```

The CTF will produce the following HTML:

```
<div>
        <span class="black">This is a sample text</span>
</div>
```

5.5 Routing

This section presents the routing module of *Convertigo Templating Framework*:

- Presentation
- Routing table

5.5.1 Presentation

The routing module is responsible for automatically navigate through screens according to C8O call responses.

For instance, let's talk about a `login` screen. A standard behavior would be to:

1. Submit the form on a C8O sequence with relevant parameters,
2. Analyze the sequence's XML response,
3. Route to the relevant screen according to the result: if the login is successful, go to the `main` screen; otherwise, go to the `error` screen

The CTF allows the C8O programmer to describe all these actions, without writing one line of code.

Changing screen means passing from one HTML page to another. It can be considered as a simple HTML page move (in a pure web application context) or also as a jQuery Mobile screen change.

5.5.2 Routing table

The routing table is a JavaScript table that lists the *route definitions* for C8O requestable responses. A *route definition* is a JavaScript object describing what to do (`actions`) when a C8O requestable response returns.

ROUTING TABLE STRUCTURE SPECIFICATION:

The routing table table must follow this structure:

```
[
// routing table : table of "route definition" objects
    {
       calledRequest: "<the called C8O requestables>",
       // indicates on which requestable results the actions occur

       actions: [
       // table of possible actions for this(these) called request(s)
         {
            afterRendering: function ($doc, c8oData) {
            },

            beforeRendering: function ($doc, c8oData) {
            },

            condition: "<jQuery selector>",
       or   condition: <function name>,
       or   condition: function ($doc, c8oData) {
              ...
              return true;
            },

            fromPage: "<pageIds>",

            goToPage: "<pageId>",

            options: {
              ...
            }

         }

       ]

    }
]
```

The `calledRequest` parameter can define a list of requestables and is described using the same format as requestables in `data-c8o-listen` attributes (see *"Listener concept"* on page 5-12).

The following table explains in details the possible parameters of each *action*.

Table 5 - 2: Routing Table Action Parameters

Parameter	Description

`condition`	A condition used to decide whether the action should apply or not.
	The condition can be:
	• a jQuery selector applied on the requestable XML response,
	• or a JavaScript function (function name or inline declared function).
	The condition is considered as validated if the jQuery selector returns a non empty list, or if the JavaScript function returns `true`.
	
	The condition function must have the following signature:
	`function($doc, c8oData) {`
	`  // $doc is the jQuery object of the XML document`
	`  // c8oData is an object containing all parameters sent to the request`
	`  // c8oData object contains key/value pairs, key being the variable/parameter name, value being the value`
	`  // returns true of false`
	`}`
	
	The condition function is called before the page changes.

`fromPage`	A list of comma separated HTML element IDs, defining the page(s) the application came from before calling the C8O requestable. `fromPage : "#page1, #page2, #page3"` The `fromPage` list of pages is used as a condition to decide whether the action should apply or not. This parameter is useful to route to different pages according to the origin page. Technically, the page IDs are tested using the `.is(selector)` method from jQuery.
`goToPage`	An HTML element ID, or an HTML page, to be displayed after the C8O response has arrived. If not present, it means a local rendering (i.e. in the same page). Note: JQueryMobile multipage mode is not currently supported.
`options`	An optional transition information (matching the jQueryMobile `transition` object format) used to display the page targetted in the `goToPage` parameter.

`beforeRendering`	JavaScript function called **before** the rendering process, **after** the page changes.
	
	The beforeRendering function must have the following signature:
	`function($doc, c8oData) {`
	`  // $doc is the jQuery object of the XML document`
	`  // c8oData is an object containing all parameters sent to the request`
	`  // c8oData object contains key/value pairs, key being the variable/parameter name, value being the value`
	`}`

`afterRendering`	JavaScript function called **after** the rendering process, **after** the beforeRendering function.
	
	The afterRendering function must have the following signature:
	`function($doc, c8oData) {`
	`  // $doc is the jQuery object of the XML document`
	`  // c8oData is an object containing all parameters sent to the request`
	`  // c8oData object contains key/value pairs, key being the variable/parameter name, value being the value`
	`}`

EXAMPLE

```
$.extend(true, C8O, {

routingTable: [
   {
       calledRequest: ".Login",
       actions: [
         {
           condition: "status:contains('1')",
           goToPage: "openworkorder.html",
           options: { transition: "none" }
         },
         {
           condition: function ($doc) {
             if ($doc.find("status:contains('true')").length != 0)
               return true;
             else
               return false;
           },
           goToPage: "#page1",
           options: { transition : "none" }
         },
         {
           condition: ">error",
           goToPage: "#Login",
           options: { transition : "slide" }
         }
       ]
    },

    {
       calledRequest: ".OpenWorkOrder",
       actions: [
         {
           condition: "Name",
           goToPage: "tasks.html",
           options: { transition: "none" }
         },
         {
           condition: "status:contains('0')",
           goToPage: "error.html",
           options: { transition: "pop" }
         }
       ]
   }
]});
```

6 Internationalization framework

This chapter introduces the Convertigo *Internationalization Library,* concepts and purpose.

- Convertigo Internationalization Library
- Dictionary content
- Translating
- Enable I18N and language configuration
- Language detection and configuration

6.1 Convertigo Internationalization Library

This section introduces the *Convertigo Internationalization Library*:

6.1.1 Objectives

The *Convertigo Internationalization library* (*I18N lib*) aims at helping Convertigo programmers to display their application in the client language. Thanks to this library, all the strings displayed in an application will appear in a specific language.

6.1.2 Translation

The translation is based on substitution:

- Words and sentences to be translated are declared directly in the HTML source or added dynamically through JavaScript code.
- The *I18N lib* uses a dictionary to translate the given words or sentences into the target language.

A separate dictionary is needed for each different language in order to perform the substitution efficiently. Indeed, the Convertigo server is not the best place to do the translation. The client (browser or mobile) must be able to do the translation for all local strings and dynamic events.

To be efficient, the client only loads the correct dictionary for the currently needed language. It is not necessary to load all the languages and use only one. This is why there is one dictionary file per language.

Strings to translate in the HTML code are handled directly in memory, at document loading. That means the language cannot be changed on the fly.

6.1.3 Project architecture

The dictionary files (one per language) are attached to the Convertigo project responsible for the application user interface. Next to the application HTML code (`index.html`, `app.html`, ...), the dictionnaries are created in an `i18n` folder.

A dictionary is a `JSON` file in the `i18n` folder. The file name should be the language code (2 chars from `ISO 639-1`) or some custom name (but then the *I18N lib* would need a custom management).

EXAMPLE

Content of the resources folder in a Convertigo UI project:

- DisplayObjects/mobile/
  - index.html
  - i18n/

- en.json
- fr.json
- es.json
- …

For desktop or mobile web application, the dictionary is downloaded and may be cached by the browser. For installed mobile application, the dictionary is local and no download is needed (like any other resource).

6.2 Dictionary content

This section presents the dictionary file content.

As already said in *"Translation"* on page 6 - 2, there is one dictionary file by language, each one being a dedicated true `JSON` file.

The `JSON` format is very important so that the *I18N lib* can read the dictionary (the *I18N lib* uses a `JSON` parser, very strict).

The `JSON` format is also very simple: the dictionary content only needs a one level key/value list: **keys** must be the same for each language, **values** should be localized strings.

In addition to the key names, for several language files, the followings are also important to respect in the `JSON` structure:

- start and end braces,
- double quotes for keys and values (no simple quotes),
- escape double quotes in keys and values with a backslash (`"quote\"sample"`),
- use `\n` for new line in keys and values,
- each key/value pair must be comma separated,
- not put a comma after the last key/value pair,
- dictionary must use the same encoding than the application HTML file (often `UTF-8`).

The followings are not important:

- key order,
- blank spaces in the structure,
- new lines in the structure.

EXAMPLE

- for `en.json`:

```
{
        "welcomeMessage":"Welcome in the I18N documentation",
        "nextButton":"Learn more"
}
```

- for `fr.json`:

```
{
        "welcomeMessage":"Bienvenue dans la documentation I18N",
        "nextButton":"En savoir plus"
}
```

6.3 Translating

This section presents the *I18N library* translating behaviors:

- Translating HTML
- Dynamically translating string or fragment in JavaScript

6.3.1 Translating HTML

The *I18N lib* automatically translates the HTML page at document loading time. The HTML source must contain *I18N*-ready strings. These *I18N* strings can be in any text node or any attribute value. If the key is not found in the dictionary, the key text is directly used.

An *I18N* string must follow this format:

`__MSG_<key>__`

The `key` part is the key defined in the dictionary entry.

EXAMPLE

Let's have the following HTML source:

```
<div>
        <p>__MSG_welcomeMessage__</p>
        <button title="__MSG_nextButton__">
          &lt; __MSG_nextButton__ &gt;
        </button>
        <p>__MSG_falseKey__</p>
</div>
```

Using the `en.json` dictionary file previously described, the HTML will render:

```
<div>
        <p>Welcome in the I18N documentation</p>
        <button title="Learn more">
          &lt; Learn more &gt;
        </button>
        <p>falseKey</p>
</div>
```

Using the `fr.json` dictionary file previously described:

```
<div>
        <p>Bienvenue dans la documentation I18N</p>
        <button title="En savoir plus">
          &lt; En savoir plus &gt;
        </button>
        <p>falseKey</p>
</div>
```

6.3.2 Dynamically translating string or fragment in JavaScript

Some strings are not part of the static application and may come from service responses or may be JavaScript messages (wrong password, unreachable server, …). These strings are displayed at runtime, the standard *I18N* translation process cannot appply.

To dynamically translate strings at runtime, the *I18N lib* provides a `translate` function, used in the two following cases:

- Translating string
- Translating HTML fragment

TRANSLATING STRING

The `C8O.translate` function takes a string as input variable, corresponding to a key from the dictionary, and returns the corresponding value for the currently used dictionary.

If the key is not found in the dictionary, the key is returned.

EXAMPLE

```
var translated = C8O.translate("welcomeMessage");
alert(translated);
```

When called in JavaScript, this piece of code would display "`Welcome in the I18N documentation`" when the currently used dictionary is `en.json`.

TRANSLATING HTML FRAGMENT

The `C8O.translate` function can also take a raw HTML fragment in parameter. This would have as effect to:

- walk every text node and attribute value from the HTML fragment,
- and replace in these strings all `__MSG_key__` patterns by the corresponding translated string (or key when it is not found in dictionary).

EXAMPLE

Usage with the *CTF*, the `translate` function can be used to format a fragment in a formatter function:

```
function ctfFormatter(fragment) {
    C8O.translate(fragment);
}
```

6.4 Enable I18N and language configuration

If you decide to use the *I18N library* features in your project, it must be enabled in the project's `custom.js` file.

The following section from `custom.js` must be un-commented and configured:

```
ro_vars: {
        i18n_files: []
},
```

`C8O.ro_vars.i18n_files` is a read-only list of available languages for the application. This table must be filled with data in the source code and cannot be dynamically updated (as it is a read-only variable).

The first language of the list is the default language. It is the language always used when a language is not found in the table.

Once the *I18N lib* is activated, the application resources must have an `i18n` folder with one dictionary file per language (like `i18n/en.json` for english dictionary).

EXAMPLE

The following lines of code allow to declare three languages (english, french and spanish), with english as default language:

```
ro_vars: {
        i18n_files: ["en", "es", "fr"]
},
```

This assumes that `i18n/en.json`, `i18n/es.json` and `i18n/fr.json` files exist in the project and `i18n/en.json` will be used by default if the current language isn't '`es`' or '`fr`'.

We recommand to follow the `ISO 639-1` codes for representing the language, see http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

You can also use any string for identifying the language dictionaries, but the language detection will not work automatically. You will have to use custom code, see "Language detection and configuration" on page 6-9.

6.5 Language detection and configuration

The translation being done as soon as possible after the document is loaded, the current language must be detected before. Find here the two ways of language detection in I18N library:

- Automatic language detection
- Manual language configuration
- General principle of language detection

AUTOMATIC LANGUAGE DETECTION

By default, the language code (from `ISO 639-1`) is extracted from the `user-agent`. This is done by the `C8O.getBrowserLanguage()` function that is automatically called during *I18N* translation process.

In this case, the `C8O.init_vars.i18n` variable should remain empty in the `custom.js` file of the application:

```
init_vars: {
      i18n: ""
},
```

MANUAL LANGUAGE CONFIGURATION

MANUALLY DEFINING A LANGUAGE

The language can be manually configured, using the `C8O.init_vars.i18n` variable.

This variable is empty by default, meaning that the *I18N library* should use the automatic language detection. But it can be customized in the `custom.js` file of the application.

When the `C8O.init_vars.i18n` variable is filled with a non-empty value, the automatic detection of the language is disabled, and the language defined in this variable is used for translation.

EXAMPLE

```
init_vars: {
      i18n: "en"
},
```

CHANGING THE CURRENT LANGUAGE USING A PARAMETER

In both cases, when the automatic detection is activated or disabled, the current language can be passed using the `__i18n` parameter in the query string or in the hash query.

EXAMPLE

- `.../myProject/index.html?param1=v1&__i18n=es&param2=v2`
- `.../myProject/index.html#param1=v1&__i18n=es&param2=v2`

CHANGING THE LANGUAGE USING JAVASCRIPT CODE

In both cases, when the automatic detection is activated or disabled, the current language can be determined by the `get_language` hook, available in the `custom.js` file.

This JavaScript hook is called previously to the translation process. The string returned by this `get_language` hook function determines the current language for translation. This can be helpful if you want to redirect some user-agents/languages to a common language for translation.

If the hook does not return a string usable as language code, the default behavior is executed, depending on the other configurations made in the `custom.js` file.

EXAMPLE

```
C8O.addHook("get_language", function (params) {
        if (localStorage) {
          var lang = localStorage["language"];
          if (typeof lang == "string") {
            return lang;
          }
        }
});
```

This example hook will use a language that is stored in local storage, when available.

GENERAL PRINCIPLE OF LANGUAGE DETECTION

In all cases of language detection/configuration, the detected language should be present in the `C8O.ro_vars.i18n_files` array of languages, as described in *"Enable I18N and language configuration"* on page 6 - 8.

If not present, the first entry, i.e. the default language, is used.

The whole language detection process works in the following order:

- `get_language` hook, can return a string to use as language code,
- `__i18n` parameter value in query string or hash query,
- `C8O.init_vars.i18n` variable value, is filled with a non-empty value,
- `C8O.getBrowserLanguage()` function, that automatically detects the language from the `user-agent`.

Appendixes

This chapter contains all appendixes related to the Reference Manual:

- Keycodes table
- Date format - Usable symbols
- Convertigo paths variables - Usable symbols
- Legacy emulator actions table

A.1 Keycodes table

This appendix contains the list of keycodes Convertigo can send to any HTML object through a *Key* statement.

Table A - 1: Keycode table

Key Pressed	Javascript KeyCode
backspace	8
tab	9
enter	13
shift	16
ctrl	17
alt	18
pause/break	19
caps lock	20
escape	27
page up	33
page down	34
end	35
home	36
left arrow	37
up arrow	38
right arrow	39
down arrow	40
insert	45
delete	46
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
a	65
b	66
c	67

Table A - 1: Keycode table (...)

Key Pressed	Javascript KeyCode
d	68
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89
z	90
left window key	91
right window key	92
select key	93
numpad 0	96
numpad 1	97
numpad 2	98
numpad 3	99
numpad 4	100
numpad 5	101
numpad 6	102
numpad 7	103
numpad 8	104
numpad 9	105

Table A - 1: Keycode table (...)

Key Pressed	Javascript KeyCode
multiply	106
add	107
subtract	109
decimal point	110
divide	111
f1	112
f2	113
f3	114
f4	115
f5	116
f6	117
f7	118
f8	119
f9	120
f10	121
f11	122
f12	123
num lock	144
scroll lock	145
semi-colon	186
equal sign	187
comma	188
dash	189
period	190
forward slash	191
grave accent	192
open bracket	219
back slash	220
close bracket	221
single quote	222

A.2 Date format - Usable symbols

This appendix contains the list of usable symbols for formatting dates.

Date and time formats are specified by *date and time pattern* strings. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string.

Text can be quoted using single quotes (') to avoid interpretation. " ' ' " represents a single quote. All other characters are not interpreted; they're simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

Table A - 2: Date format - Usable Symbols

G	Era designator	Text	AD
y	Year	Year	1996 & 96
M	Month in year	Month	July & Jul & 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday & Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time & PST & GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

A.3 Convertigo paths variables - Usable symbols

This appendix contains the list of Convertigo paths variables available in Site Clipper objects using **replacement strings**.

In replacement string properties, you can use specific symbols, that will allow you to compute in runtime some well known paths.

The following symbols are available:

Table A - 3: Convertigo paths variables - Usable Symbols

Variable name	Retreived value
`project_path`	retrieves the path to your current project
`siteclipper_path`	retrieves the Site Clipper path
`host_path`	retrieves the Site Clipper path and the target host path (without the resource path)
`tail_path`	retrieves the Site Clipper path, the target host path and the resource path (without the resource)

In order to use such symbols, just add `$<symbol>$` into your replacement string value. You can use as many symbols as you want and at any location.

All symbols are computed accordingly to the current project/connector/context.

You can use them in order to make the objects using replacement strings more maintainable and simplier. This is also useful because you do not have to deal with project name (it can change for instance between a development environment and a production one). This makes your string replacements dynamic.

EXAMPLES

Let's say Convertigo receives the following Site Clipper request:

```
http://my_server:8080/convertigo/projects/my_project/
connector=my_connector,context=my_context.siteclipper/http/
remote_host,18080/remote/path/remote_resource
```

The previously described variables are computed with the following values:

- `$project_path$` =

  `"/convertigo/projects/my_project"`

- `$siteclipper_path$` =

  ```
  "/convertigo/projects/my_project/
  connector=my_connector,context=my_context.siteclipper"
  ```

- `$host_path$` =

  ```
  "/convertigo/projects/my_project/
  connector=my_connector,context=my_context.siteclipper/http/
  remote_host,18080"
  ```

- `$tail_path$ =`

  ```
  "/convertigo/projects/my_project/
  connector=my_connector,context=my_context.siteclipper/http/
  remote_host,18080/remote/path"
  ```

All these symbols don't contain the convertigo host ("`my_server:8080`") and never contain trailing slash.

A.4 Legacy emulator actions table

An action is a special keystroke that an emulator can execute as `ENTER` or `SOMMAIRE`. This appendix contains the list of the valid emulator actions Convertigo can send to *Javelin* emulator, depending on the emulator type.

Table A - 4: Videotex emulator - Actions table

Description	Code
Suite	KSuite
Sommaire	KSommaire
Guide	KGuide
Répétition	KRepetition
Annulation	KAnnulation
Correction	KCorrection
Retour	KRetour
Envoi	KEnvoi
Connexion/Fin	KCnxFin

Table A - 5: VT220 emulator - Actions table

Description	Code
F1	F01
F2	F02
F3	F03
F4	F04
F5	F05
F6	F06
F7	F07
F8	F08
F9	F09
F10	F10
F11	F11
F12	F12

Table A - 6: Bull emulator - Actions table

Description	Code
FKCx [1..12]	FKCx
Right arrow	RIGHT
Left arrow	LEFT
Up arrow	UP

Table A - 6: Bull emulator - Actions table (...)

Description	Code
Down arrow	DOWN
Delete active partition	CLEARAP
Delete end of partition	CLEAREP
Initialize active partition	INITAP
Initialize two partitions	INITBP
Partial initialization	INITPA
Insert line	INSLINE
Delete line	SUPLINE
Delete end of line	ERAEOL
Move cursor to begin of line	CURBOL
Move cursor to begin of map	CURHOME
Delete character	DELCHAR
Insert character	INSCHAR
Tabulation	TAB
Back tabulation	BTAB
Insert tabulation	INSTAB
Delete tabulation	CLRTAB
Baskspace	BS
Total transmission	XMITALL
Transmission	XMIT
Break	BREAK

Table A - 7: IBM emulator - Actions table

Description	Code
PAx [1..3]	KEY_PAx [1..3]
SysReq	KEY_SYSREQ
Enter	KEY_ENTER
Attn	KEY_ATTN
PFx [1..24]	KEY_PFx [1..24]
Right arrow	KEY_CURRIGHT
Left arrow	KEY_CURLEFT
Up arrow	KEY_CURUP
Down arrow	KEY_CURDOWN
Backspace	KEY_BACKSP
Tabulation	KEY_TAB
Back tabulation	KEY_BACKTAB

Table A - 7: IBM emulator - Actions table (...)

Description	Code
New line	KEY_NEWLINE
Move cursor to begin of map	KEY_HOME
Insert	KEY_INSERT
Delete	KEY_DELCHAR
Reset	KEY_RESET
Duplication	KEY_DUP
Field mark	KEY_FLDMRK
Delete end of line	KEY_ERASEEOF
Delete entry	KEY_ERASEINPUT
Current selection	KEY_CURSEL
Clear	KEY_CLEAR