convertigo
MASHUP YOUR BUSINESS

Starting with Convertigo Legacy Integrator

Quick Guide

CEMS 5.2.0

TABLE OF CONTENTS

Getting Familiar with Convertigo Legacy Integrator 1

CLI Purpose 1-1

CLI Concepts and Objects 1-2

General Concept 1-2

Block Factory 1-2

CLI Objects 1-3

Definitions 1-4

Connector 1-4

Screen Class 1-5

Transaction 1-10

CLI Studio 1-13

General Presentation 1-13

View Description 1-14

Projects View 1-14

Properties View 1-16

Connector View 1-20

My First Convertigo Legacy Integrator Project 2

Legacy Integrating Project 2-1

Project Description 2-1

Opening the Sample Project from the Studio 2-4

Setting up the project 2-5

Creating a Project and Setting Connector Parameters..... 2-6
Defining Screen Classes 2-10
Defined Screen Classes..... 2-10
Creation of First Screen Class 2-12
Creation of Additional Screen Classes..... 2-16
Creating Extraction Rules 2-22
Defining a Transaction..... 2-39
Executing a Transaction using the Test Platform 2-75

Exposing Projects as Web Services and Presenting Data.....3

Exposing Projects as Web Services 3-1
Exposing a Project as a REST Web Service 3-1
Exposing a Project as a SOAP Web Service..... 3-3
XSL AJAX Presentation..... 3-18

LIST OF FIGURES

Figure 1 - 1: Convertigo Legacy integrator exposes any existing Legacy application as SOAP or REST Web services 1-1

Figure 1 - 2: Block generation on legacy application screen (partial view of screen) 1-3

Figure 1 - 3: CLI objects 1-4

Figure 1 - 4: Connectors folder 1-5

Figure 1 - 5: Screen classes folder 1-5

Figure 1 - 6: Objects inherited from the Default screen class 1-6

Figure 1 - 7: Screen classes and Inherited screen classes folders 1-6

Figure 1 - 8: Legacy application screen identifier 1-7

Figure 1 - 9: "Find string" criterion in screen class Criteria folder 1-7

Figure 1 - 10: Default extraction rules 1-8

Figure 1 - 11: Example of extracted blocks 1-9

Figure 1 - 12: Tag article_status extraction rule 1-9

Figure 1 - 13: XML tag name of extracted data 1-9

Figure 1 - 14: Transactions folder 1-11

Figure 1 - 15: Screen class handler triggered on detection of the LoginScreen screen class .. 1-12

Figure 1 - 16: Screen class handler in transaction Functions folder 1-12

Figure 1 - 17: CLI Studio 1-13

Figure 1 - 18: Formatting of unsaved, inherited and disabled objects in the Projects View 1-16

Figure 1 - 19: Properties View 1-17

Figure 1 - 20: Text property field edition 1-17

Figure 1 - 21: Javascript expression field edition 1-18
Figure 1 - 22: Text property edition via edition window 1-18
Figure 1 - 23: Javascript expression window 1-18
Figure 1 - 24: Edition window 1-19
Figure 1 - 25: Configuration window 1-19
Figure 1 - 26: Text property edition via drop-down menu 1-20
Figure 1 - 27: Selecting a value in the drop-down menu 1-20
Figure 1 - 28: Connector View (Output tab selected by default) 1-21
Figure 1 - 29: Example legacy application screen 1-24
Figure 1 - 30: MMS001E screen class extraction rules 1-24
Figure 1 - 31: XML output resulting from execution of tag name extraction rules 1-24
Figure 2 - 1: Legacy application login screen 2-2
Figure 2 - 2: Input of username and password into login fields 2-2
Figure 2 - 3: Input of article management code (general menu) 2-2
Figure 2 - 4: Extraction of article list (on execution of Table-type extraction rule) 2-3
Figure 2 - 5: Input of article code 2-3
Figure 2 - 6: Extraction of article specific data (on execution of XML tag name-type extraction rules) 2-4
Figure 2 - 7: Launching the New Project wizard 2-5
Figure 2 - 8: Selecting the CLI sample project 2-5
Figure 2 - 9: Sample project in Projects View 2-5
Figure 2 - 10: New Project Wizard 2-6
Figure 2 - 11: Selecting a project type 2-6
Figure 2 - 12: Entering a project name 2-7
Figure 2 - 13: Entering a connector name 2-7
Figure 2 - 14: Setting connection parameters 2-8
Figure 2 - 15: New project summary window 2-8
Figure 2 - 16: sample_doc_CLI project appearing in the Projects View 2-9
Figure 2 - 17: Connector View displaying legacy application login screen 2-9
Figure 2 - 18: "Entrez votre nom" text string (field name) 2-11

Figure 2 - 19: "MNS010" text string (screen identifier) 2-11
Figure 2 - 20: "Gestion Articles" text string (screen title) 2-11
Figure 2 - 21: "MMS001B1" text string (screen identifier) 2-12
Figure 2 - 22: "MMS001E" text string (screen identifier) 2-12
Figure 2 - 23: Sample project login screen of emulated legacy application 2-13
Figure 2 - 24: Selecting the LoginScreen screen class detection criterion on screen 2-13
Figure 2 - 25: Newly created screen class (Entrez votre nom) and associated detection criterion 2-14
Figure 2 - 26: Renaming a screen class 2-14
Figure 2 - 27: Renamed screen class (LoginScreen) - inherited and new objects 2-15
Figure 2 - 28: Clicking Go to current screen class icon 2-15
Figure 2 - 29: Highlighted screen class after Go to current screen class object command 2-16
Figure 2 - 30: Selecting the MNS010 screen class detection criterion on screen 2-16
Figure 2 - 31: Newly created screen class (MNS010) and associated detection criteria 2-17
Figure 2 - 32: Selecting the ArticlesManagementScreen screen class detection criterion on screen 2-17
Figure 2 - 33: Newly created screen class (Gestion Articles) and associated detection criteria 2-18
Figure 2 - 34: "Gestion Articles" screen title 2-18
Figure 2 - 35: Selecting the MMS001B1 screen class detection criterion on screen 2-19
Figure 2 - 36: Newly created screen class (MMS001/B1) and associated detection criteria 2-20
Figure 2 - 37: Selecting the MMS001E screen class detection criterion on screen 2-20
Figure 2 - 38: Newly created screen class (MMS001E) and associated detection criteria 2-21
Figure 2 - 39: Selecting new extraction rule 2-22
Figure 2 - 40: New Extraction Rule wizard 2-23
Figure 2 - 41: Giving a name to the new extraction rule 2-23
Figure 2 - 42: New extraction rule in Extraction rules folder 2-24
Figure 2 - 43: Increasing the ArticlesTable extraction rule priority by one rank 2-24
Figure 2 - 44: Table extraction rule properties 2-25
Figure 2 - 45: Delineating the screen zone containing data to be extracted 2-26
Figure 2 - 46: Linking selection to Screen zone property 2-26

Figure 2 - 47: Automatic setting of Screen zone property value 2-26
Figure 2 - 48: Setting ArticlesTable extraction rule first column 2-27
Figure 2 - 49: First column of ArticlesTable extraction rule marked on screen 2-27
Figure 2 - 50: Second column of ArticlesTable extraction rule marked on screen 2-28
Figure 2 - 51: Third column of ArticlesTable extraction rule marked on screen 2-28
Figure 2 - 52: Column parameters of ArticlesTable extraction rule 2-29
Figure 2 - 53: Changing column labels 2-30
Figure 2 - 54: Columns property values 2-30
Figure 2 - 55: Setting Do not include titles property to true 2-31
Figure 2 - 56: Generating XML output of MMS001B1 screen 2-31
Figure 2 - 57: Generated XML and Last detected screen class 2-32
Figure 2 - 58: Data extracted according to ArticlesTable extraction rule properties 2-32
Figure 2 - 59: Selecting data to be extracted on screen 2-33
Figure 2 - 60: New TagName window 2-33
Figure 2 - 61: New extraction rule (Tag article_status) in Extraction rules folder 2-33
Figure 2 - 62: Increasing the Tag article_status extraction rule priority by one rank 2-34
Figure 2 - 63: Generated XML and Last detected screen class 2-34
Figure 2 - 64: Data extracted according to Tag article_status extraction rule properties 2-35
Figure 2 - 65: Selecting data to be extracted on screen 2-35
Figure 2 - 66: New TagName window 2-36
Figure 2 - 67: New extraction rule (Tag article_rsp) in Extraction rules folder 2-36
Figure 2 - 68: Increasing the Tag article_rsp extraction rule priority by one rank 2-37
Figure 2 - 69: Selecting data to be extracted on screen 2-37
Figure 2 - 70: New TagName window 2-38
Figure 2 - 71: New extraction rule (Tag product_group) in Extraction rules folder 2-38
Figure 2 - 72: Increasing the Tag product_group extraction rule priority by one rank 2-39
Figure 2 - 73: Creating new transaction 2-40
Figure 2 - 74: New Transaction wizard 2-41
Figure 2 - 75: Entering the transaction name 2-41

Figure 2 - 76: New transaction (GetArticleData) in Transactions folder 2-42

Figure 2 - 77: Edit handlers command 2-42

Figure 2 - 78: Transaction handlers editor 2-43

Figure 2 - 79: Creating new handler 2-43

Figure 2 - 80: New transaction handler window 2-44

Figure 2 - 81: onLoginScreenEntry() screen class handler 2-45

Figure 2 - 82: onLoginScreenEntry() screen class handler definition in Editor 2-46

Figure 2 - 83: Document under edition 2-46

Figure 2 - 84: Positioning cursor below comments 2-47

Figure 2 - 85: Javascript method list 2-48

Figure 2 - 86: Selecting "send" method in list 2-48

Figure 2 - 87: Automatic insertion of method 2-48

Figure 2 - 88: First JavaScript statement of onLoginScreenEntry() screen class handler 2-49

Figure 2 - 89: Cursor position "move cursor" statement of onLoginScreenEntry screen class handler 2-49

Figure 2 - 90: Second JavaScript statement of onLoginScreenEntry() screen class handler ... 2-50

Figure 2 - 91: Third JavaScript statement of onLoginScreenEntry() screen class handler 2-50

Figure 2 - 92: Fourth JavaScript statement of onLoginScreenEntry() screen class handler 2-51

Figure 2 - 93: Fifth JavaScript statement of onLoginScreenEntry() screen class handler 2-52

Figure 2 - 94: onLoginScreenEntry() screen class handler definition 2-53

Figure 2 - 95: Executing transaction 2-53

Figure 2 - 96: Accessed screen after execution of transaction 2-54

Figure 2 - 97: Creating new handler 2-54

Figure 2 - 98: onMNS010Entry() screen class handler 2-55

Figure 2 - 99: onMNS010Entry() screen class handler definition in Editor 2-56

Figure 2 - 100: Positioning cursor below comments in MNS010Entry() function 2-57

Figure 2 - 101: Cursor position for "move cursor" statement of onMNS010Entry screen class handler 2-58

Figure 2 - 102: First JavaScript statement of onMNS010Entry() screen class handler 2-58

Figure 2 - 103: Second JavaScript statement of onMNS010Entry() screen class handler 2-59

Figure 2 - 104: Third JavaScript statement of onMNS010Entry() screen class handler 2-59
Figure 2 - 105: Fourth JavaScript statement of onMNS010Entry() screen class handler 2-60
Figure 2 - 106: Fifth JavaScript statement of onMNS010Entry() screen class handler 2-60
Figure 2 - 107: onMNS010Entry() screen class handler definition 2-61
Figure 2 - 108: Executing transaction 2-61
Figure 2 - 109: Accessed screen (MMS001B1) after execution of transaction 2-62
Figure 2 - 110: Creating new handler 2-63
Figure 2 - 111: onMMS001B1Exit() screen class handler 2-63
Figure 2 - 112: onMMS001B1Exit() screen class handler definition in Editor 2-64
Figure 2 - 113: Positioning cursor below comments in MMS001B1Exit() function 2-65
Figure 2 - 114: Cursor position for "move cursor" statement of onMMS001B1Exit screen class handler 2-66
Figure 2 - 115: First JavaScript statement of onMMS001B1Exit() screen class handler 2-67
Figure 2 - 116: Second JavaScript statement of onMMS001B1Exit() screen class handler 2-67
Figure 2 - 117: Third JavaScript statement of onMMS001B1Exit() screen class handler 2-68
Figure 2 - 118: Fourth JavaScript statement of onMMS001B1Exit() screen class handler 2-68
Figure 2 - 119: Fifth JavaScript statement of onMMS001B1Exit() screen class handler 2-69
Figure 2 - 120: onMMS001B1Exit() screen class handler definition 2-69
Figure 2 - 121: Executing transaction 2-70
Figure 2 - 122: Accessed screen (MMS001E) after execution of transaction 2-71
Figure 2 - 123: Creating a new transaction variable 2-72
Figure 2 - 124: New Variable wizard 2-72
Figure 2 - 125: Giving a name to the new variable 2-73
Figure 2 - 126: New variable in transaction Variables folder 2-73
Figure 2 - 127: Variable properties 2-74
Figure 2 - 128: Description variable property 2-74
Figure 2 - 129: Default value variable property 2-74
Figure 2 - 130: Variable in onMMS001B1Exit() Javascript function 2-75
Figure 2 - 131: Convertigo test platform URL 2-76
Figure 2 - 132: Test platform 2-76

Figure 2 - 133: XML output in the Execution result window of the test platform 2-77
Figure 3 - 1: URL syntax for exposing transactions as REST Web services 3-2
Figure 3 - 2: Transaction exposed as a REST Web service 3-3
Figure 3 - 3: GetArticleData transaction properties 3-4
Figure 3 - 4: Setting the Public Method property to true 3-4
Figure 3 - 5: Update transaction schema from current generated XML 3-5
Figure 3 - 6: Update transaction schema - Confirmation window 3-5
Figure 3 - 7: Generate XML first 3-6
Figure 3 - 8: Project folder in Project Explorer 3-6
Figure 3 - 9: Project schema in Design tab of Editor 3-7
Figure 3 - 10: Project schema in Source tab of Editor 3-8
Figure 3 - 11: Transaction data types in schema 3-9
Figure 3 - 12: URL syntax for displaying the SOAP Web service WSDL 3-10
Figure 3 - 13: SOAP Web service WSDL 3-10
Figure 3 - 14: Launching the Web Services Explorer 3-11
Figure 3 - 15: Web Services Explorer 3-11
Figure 3 - 16: Web Services Explorer 3-13
Figure 3 - 17: URL of the SOAP service in the Web Services Explorer 3-14
Figure 3 - 18: Web Services Explorer panes after a WSDL URL is reached 3-14
Figure 3 - 19: WSDL Binding Details in Actions pane of the Web Services Explorer 3-15
Figure 3 - 20: List of methods displayed in the Web Services Explorer 3-15
Figure 3 - 21: Web service elements displayed in the Navigator pane 3-15
Figure 3 - 22: Invoke a web service method view in the Actions pane 3-16
Figure 3 - 23: Invoking a web service method 3-16
Figure 3 - 24: Status pane after web service method invocation 3-17
Figure 3 - 25: SOAP Request envelope detail 3-17
Figure 3 - 26: SOAP response envelope detail 3-18
Figure 3 - 27: New Sheet wizard 3-19
Figure 3 - 28: Entering a name for the new style sheet 3-20

Figure 3 - 29: New style sheet object in the Projects View 3-20
Figure 3 - 30: New File wizard 3-21
Figure 3 - 31: New file in Project Explorer 3-21
Figure 3 - 32: XSL style sheet 3-22
Figure 3 - 33: Template rule on document 3-22
Figure 3 - 34: Template rule on articles 3-23
Figure 3 - 35: Template rule on article_status, article_rsp and product_group 3-23
Figure 3 - 36: Sheet properties 3-24
Figure 3 - 37: Transaction properties 3-25
Figure 3 - 38: Transaction Style Sheet property 3-25
Figure 3 - 39: XML output after XSL transformation 3-26

LIST OF TABLES

Table 1 - 1: CLI objects 1-3

Table 1 - 2: Criterion types 1-7

Table 1 - 3: CLI Default extraction rules 1-8

Table 1 - 4: Block standard XML attributes 1-10

Table 1 - 5: Extraction rule common properties 1-10

Table 1 - 6: Screen class handler types 1-12

Table 1 - 7: CLI views 1-14

Table 1 - 8: Standard Eclipse buttons 1-14

Table 1 - 9: Projects View tabs and icons 1-15

Table 1 - 10: Properties View icons 1-17

Table 1 - 11: Connector View parts 1-20

Table 1 - 12: Connector View icons 1-21

Table 2 - 1: Defined screen classes 2-10

Table 2 - 2: Sample project extraction rules 2-22

Table 2 - 3: Table parameters 2-29

Table 2 - 4: Table extraction rule properties 2-30

Table 2 - 5: Transaction actions 2-40

Table 2 - 6: New transaction handler window description 2-44

Table 2 - 7: LoginScreen screen class - Actions to be performed and methods 2-47

Table 2 - 8: doAction method codes 2-50

Table 2 - 9: MNS010 screen class - Actions to be performed and methods 2-57

Table 2 - 10: MMS001B1 screen class - Actions to be performed and methodsB 2-65

Table 3 - 1: Web Services Explorer 3-12

Table 3 - 2: Style Sheet transaction property 3-24

1 Getting Familiar with Convertigo Legacy Integrator

This chapter presents the purpose, concepts and objects of **Convertigo Legacy Integrator** (**CLI**), one of Convertigo Enterprise Mashup Studio's components. It also describes the four views of the Eclipse-based CLI Studio and their elements (tabs, icons, menu).

- CLI Purpose
- CLI Concepts and Objects
- CLI Studio

1.1 CLI Purpose

The purpose of Convertigo Legacy Integrator is to expose the internal business logic of legacy applications as modern SOA-oriented Web services (SOAP and REST).

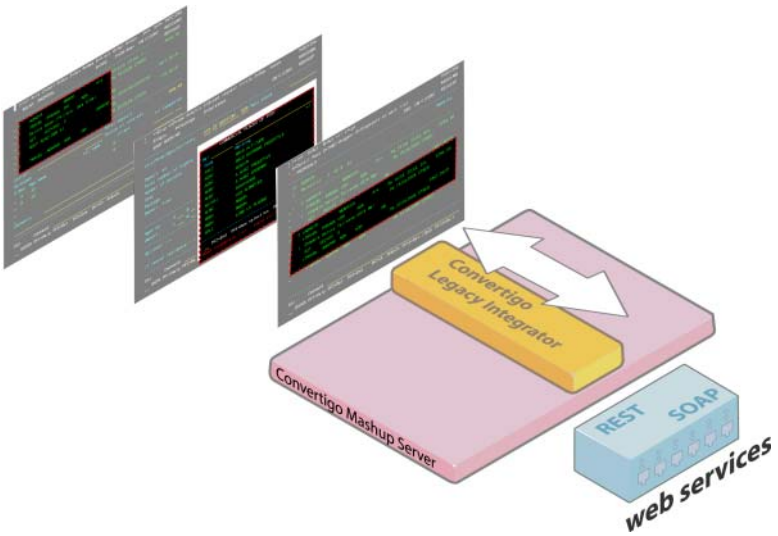


Figure 1 - 1: Convertigo Legacy integrator exposes any existing Legacy application as SOAP or REST Web services

CLI accesses legacy applications in a non intrusive way (without changing existing applications) through their existing interface (3270, 5250, VT, DKU) user screens or IBM's CICS COMMAREA.

CLI encompasses a large number of applications:

- *Integrating* existing "green screen" legacy applications into SOA architectures.

- *Creating* APIs (Application Programming Interfaces) to any "green screen"-based legacy application.
- *Automating* complex user tasks performed on multiple "old-fashioned" legacy application screens.
- *Inserting* into or *pulling* from databases information stored in legacy applications, even without knowledge of their programming language or database technology.

1.2 CLI Concepts and Objects

This section presents the general concept and notions of the application. It then relates notions to real CLI objects, which are defined in detail in a dedicated sub-section.

- General Concept
- CLI Objects
- Definitions

1.2.1 General Concept

CLI accesses a legacy application (for example IBM 5250, IBM 3270, BULL DKU) via a *connector*. It then navigates through the application "green" *screens* by detecting to which *screen class* a screen belongs (based on detection *criteria*) and executes associated *extraction rules*. Extracted data is structured in XML format. It can then be turned into HTML for display purpose using XSL style sheets or exposed as SOAP or REST Web services.

Navigation is automated through the definition of a variable-based *transaction*, which includes *screen class handlers* based on JavaScript functions. Screen class handlers trigger actions (execution of JavaScript methods) on detection of *screen classes*.

BLOCK FACTORY

For extraction purpose, legacy application screens are broken into elementary entities called *blocks*. The process in charge of generating blocks is called *block factory*. The block generation process can be divided into three major steps:

1 The block factory analyses screen lines.

2 For each line, the block factory extracts character strings having same attributes (text color, background color, bold, blinking…).

3 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.

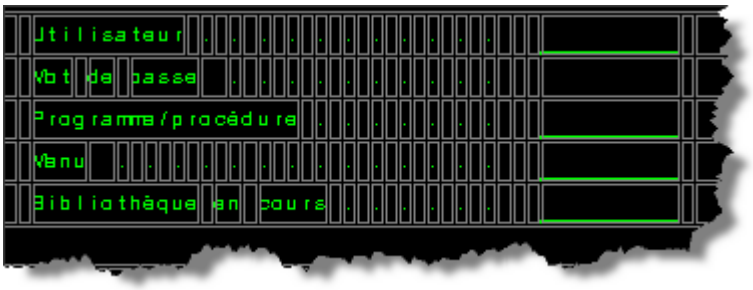


Figure 1 - 2: Block generation on legacy application screen (partial view of screen)

1.2.2 CLI Objects

All of the previously described concepts correspond to as many CLI objects, that are set as part of a Legacy integrating project (except the *block factory*).

The following table describes CLI objects:

Table 1 - 1: CLI objects

Object[a]	Description
Connector	Defines the type of legacy application to which CLI connects.
Screen class	Defines a family of legacy application screens with common characteristics.
Criterion	Is a characteristic element defining a specific screen class.
Extraction rule	Is a rule defining how information is extracted from a legacy application screen.
Block factory	See *"Block Factory"* on page 1 - 2.
Transaction	Defines the actions Convertigo must perform on legacy application screens.
Screen class handler	Is a set of JavaScript functions executed on detection of a specific screen class.
Variable	Is the transaction input variable.

a. *Objects are indented in the **Object** column according to their indentation level in the Project tree of the Projects View (see Figure 1 - 3)*

All of these objects are set when developing a Legacy integrating project. Once the project has been created, these objects appear in the **Projects View** of the CLI Studio (*"CLI Studio"* on page 1 - 13):

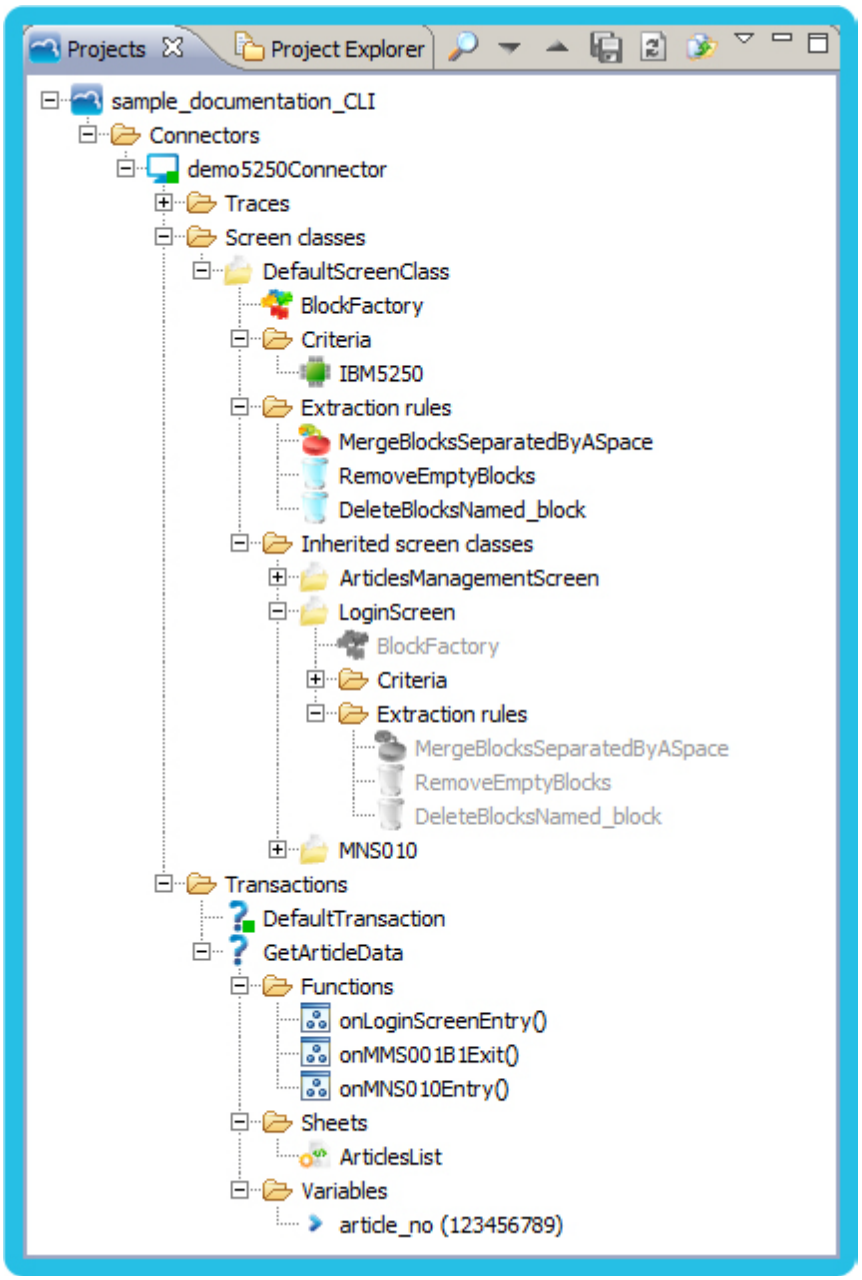


Figure 1 - 3: CLI objects

To get started with your first Legacy integrating project, see *"My First Convertigo Legacy Integrator Project" on page 2-1*.

1.2.3 Definitions

CONNECTOR

A connector defines the type of legacy application CLI is able to connect to.

One type of connector is available in CLI: the *Javelin* connector, which represents a connection to a legacy system. The connector is based on a terminal emulation session which represents the connection to a legacy system type (IBM5250, Bull DKU...).

Connector parameters are set in the **New Project** wizard which opens when creating a Legacy integrating project (see *"Creating a Project and Setting Connector Parameters"* on page 2-6).

Once a new project has been created, the corresponding connector appears in the *Connectors* folder of the project, in the **Projects View** of the CLI Studio (*"CLI Studio"* on page 1 - 13):

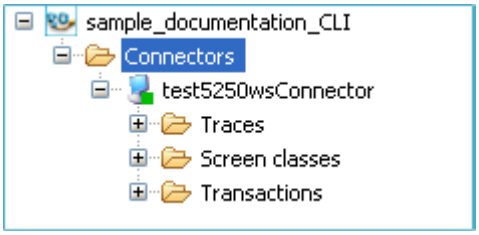


Figure 1 - 4: Connectors folder

Screen Class

Screen classes define families of legacy application screens with common characteristics (called *criteria*). Screen classes are:

- uniquely identified by at least one *criterion*;
- associated to *extraction rules*.

A single root screen class, called ***Default screen class***, is created by default when creating a project. The default screen class is defined by only one - specific, preset and un-editable - criterion. A basic set of common extraction rules is also associated to the default screen class upon project creation. It is parent to all other screen classes.

Screen classes appear in the **Screen Classes** folder of the project:

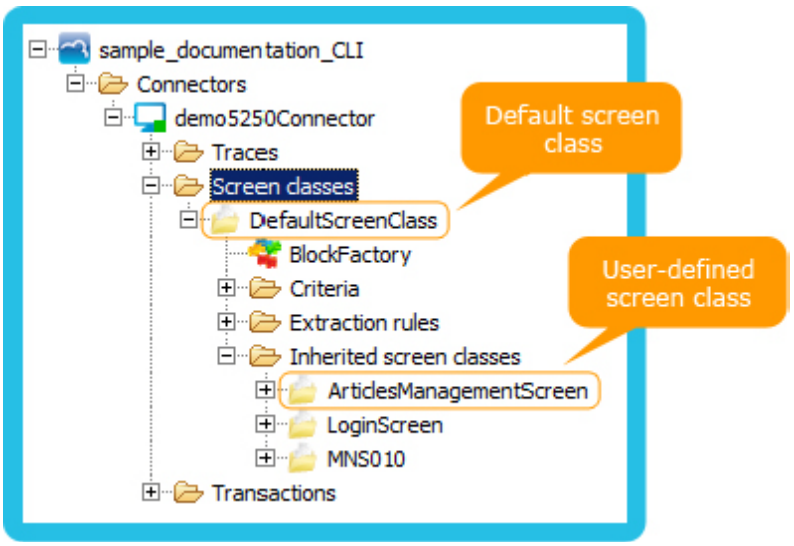


Figure 1 - 5: Screen classes folder

A screen class based on a parent screen class (defined with extra criteria and extraction rules for instance) is called ***Inherited screen class***. It inherits criteria, extraction rules and style sheets from its parent screen class.

Inherited objects appear greyed (see Figure 1 - 18) in the **Projects View**:

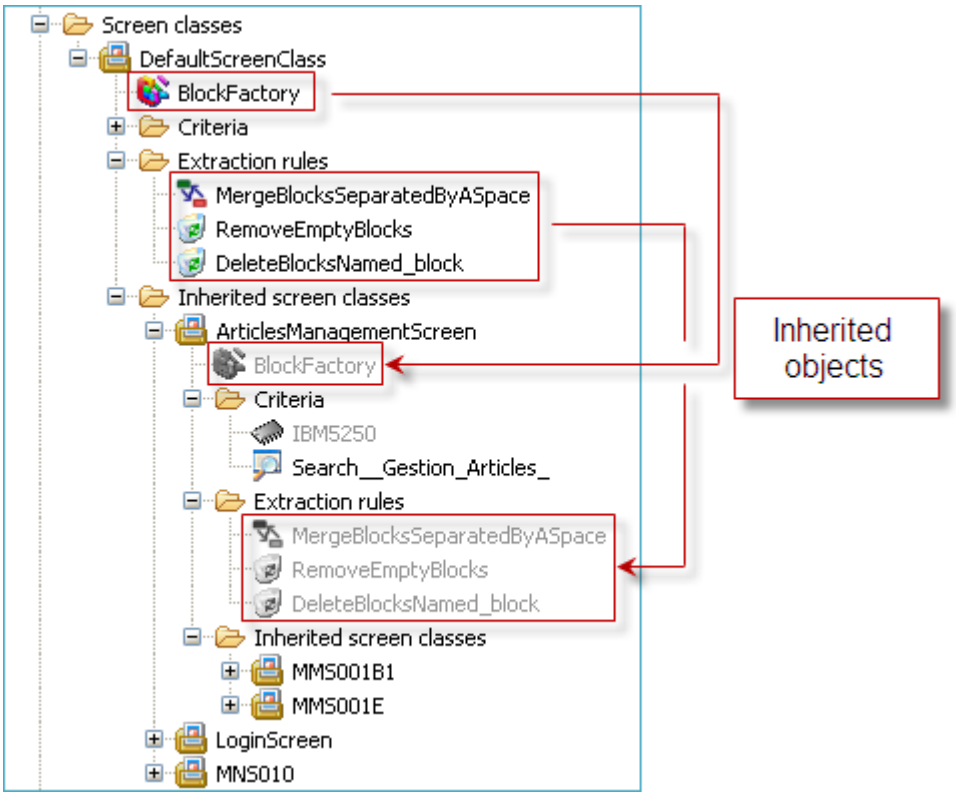


Figure 1 - 6: Objects inherited from the Default screen class

Inherited screen classes appear in subfolders named *Inherited screen classes* in the **Projects View** of the CLI Studio (*"CLI Studio"* on page 1 - 13):

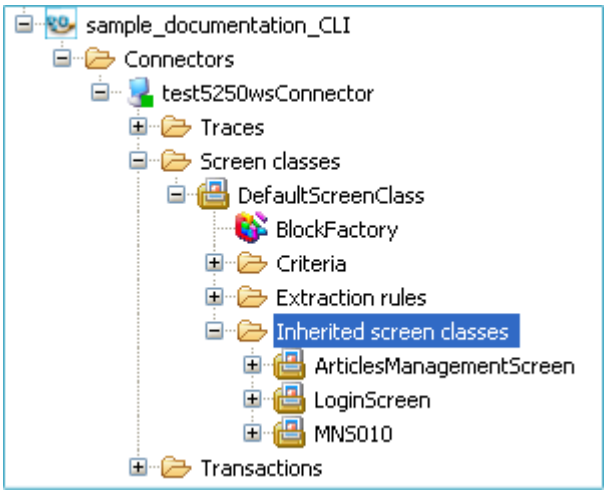


Figure 1 - 7: Screen classes and Inherited screen classes folders

CRITERION

A legacy application screen is considered belonging to a screen class when it matches all criteria defined by the user for this screen class. If no criterion matches, the screen is considered belonging to the default screen class (provided that the default screen class is not associated with any other detection criterion than the default criterion, see below).

When creating a project (and defining a connector), a criterion is automatically created in the Criteria folder of the `DefaultScreenClass`. This default criterion corresponds to the *emulator technology*. For example, if the connector set for the project emulates a connection

to an IBM5250 terminal, an `IBM5250` criterion is created by default (see Figure 1 - 6).

Three other types of criteria can be defined in CLI:

Table 1 - 2: Criterion types

Type	Icon	Description	Matching condition
Empty screen		Detects empty screens.	Matches when all screen map lines are empty.
Find string		Defines a specific string to find in the screen map based on presentation attributes, position (column, line) and string value. Easily set by selecting the required string on screen with the mouse.	Matches when text starting at given position in screen map is the searched string and has same presentation attributes
Regular expression		Defines a regular expression to find into the screen map based on presentation attributes, position (column, line) and regular expression value.	Matches when text starting at given position in screen map verifies the regular expression and has same presentation attributes

Developers of legacy application screens sometimes add a screen identifier in the upper left corner of the screen. If present, this identifier should always be selected as detection criteria of a defined screen class:

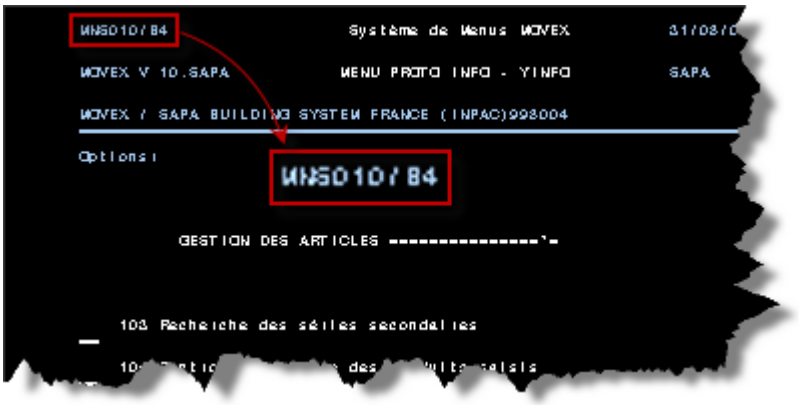


Figure 1 - 8: Legacy application screen identifier

Once a new criterion has been created, it appears in the Criteria folder of the relevant screen class:

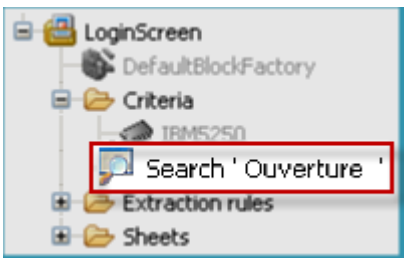


Figure 1 - 9: "Find string" criterion in screen class Criteria folder

EXTRACTION RULE

Extraction is based on the block structure of legacy screens (see *"Block Factory"* on page 1-2). Extraction rules define which data must be extracted from legacy application screens and

their properties, such as the XML tag name of extracted data in the final XML output, the screen position of data to be extracted, etc.

During a *transaction* process, when a screen class is detected (i.e. all criteria defined for this screen class match), all screen class-specific extraction rules - including those of inherited screen classes - are executed.

CLI includes a number of default extraction rules belonging to the `DefaultScreenClass` screen class. Default extraction rules are inherited by any screen class created for a given project.

Default and user-defined extraction rules are all and always executed one after another according to their order in the **Extraction rules** folder of the currently detected screen class:

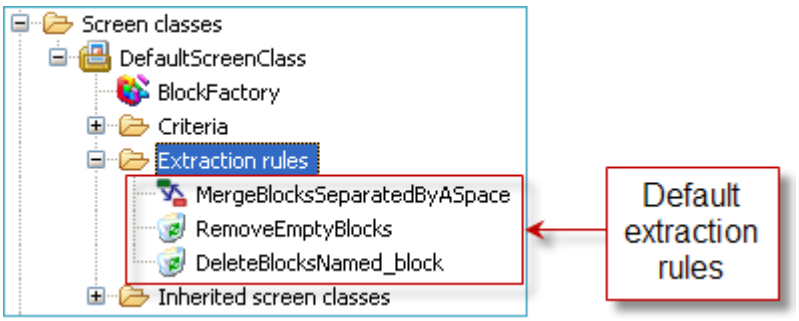


Figure 1 - 10: Default extraction rules

The following table describes CLI default extraction rules:

Table 1 - 3: CLI Default extraction rules

Default extraction rule	Description
`MergeBlocksSeparatedByASpace`	Groups together blocks separated by a space. Example: "Part number" is one block.
`RemoveEmptyBlocks`	Removes blocks not containing any information (text, field, etc.).
`DeleteBlocksNamed_block`	Deletes blocks with XML tag name "block", that is, extracted blocks not renamed through user-defined extraction rules.

When extraction rules are executed, blocks are extracted in XML format and an XML document is generated (XML representation of the legacy application screen).

Extracted blocks are given the default XML tag name "`block`". When creating extraction rules, such as *Table* or *XML tag name* extraction rules for example, extracted blocks are renamed through their **XML tag name** and **Label policy** (if applicable) properties.

The `DeleteBlocksNamed_block` extraction rule deletes all blocks called "`block`". As a consequence, after all extraction rules have been executed, only blocks renamed through extraction rules remain.

The `DeleteBlocksNamed_block` must always be positioned last in the Extraction rules folder of screen classes.

For example, if we consider the following data to be extracted (*article status*):

Figure 1 - 11: Example of extracted blocks

To extract this specific data, an *XML tag name* extraction rule called `Tag article_status` is created:

Figure 1 - 12: Tag article_status extraction rule

The **XML tag name** property is given, for example, the `article_status` value:

XML tag name	article_status

.

After all extraction rules have been executed, data appearing in the final XML output contains the block renamed as `article_status`:

Figure 1 - 13: XML tag name of extracted data

*Tag names can also be extracted from the screen through the **Label policy** property.*

In addition to their XML tag name, blocks are described in XML format using ***XML standard attributes***:

Table 1 - 4: Block standard XML attributes

Attribute	Description
background	Defines the name of the block background color.
blink	Defines if the block has the « blinking » presentation attribute (`false` if the XML attribute is not specified).
column	Defines the block starting column on emulation screen.
foreground	Defines the name of the block text color.
intense	Defines if the block has the « intense » presentation attribute (`false` if the XML attribute is not specified)
line	Defines the block starting line on the emulation screen.
name	Defines the block name.
reverse	Defines if the block has the « inverse video » presentation attribute (`false` if the XML attribute is not specified)
type	Defines the block type (several block families can be created).
underlined	Defines if the block has the « underlined » presentation attribute (false if the XML attribute is not specified).

Extraction rules share three common properties:

Table 1 - 5: Extraction rule common properties

Property	Description
Attributes	Specifies the expected formatting attributes of a block for the rule to apply (for example green text on black background).
Screen zone	Defines the position and size of the screen area to which the rule applies. Any block found in the specified perimeter is affected by the rule. Screen area is defined by four values: . *x*: area left corner; . *y*: area upper corner; . *width*: area width; . *height*: area height. "-1" represents an undefined value. *[x=-1, y=-1, width=-1, height=-1]* represents an undefined area in which all blocks, whatever their coordinates, are affected.
Type	Specifies which types of block are affected by the rule using a regular expression. There are three types of block: . *static*:non editable static text area; . *field*: editable text area; . *filler*: filler area.

TRANSACTION

Navigating through legacy application screen requires using screen transactions, that is to say a series of actions (represented by JavaScript statements) performed from one legacy screen to another. These actions are triggered on detection of defined screen classes, using *screen class handlers*.

A transaction can be summed up as a process:

- taking *variables* in input;
- producing an XML document in output.

When executing the transaction, transaction variables are automatically added as variables of the JavaScript scope. Transaction variables can be given a fixed (default) value, or their value can be set as input parameter prior to executing a transaction.

In CLI, a transaction is defined as a set of Javascript *functions* containing Javascript *methods*:

- A Javascript function is a *screen class handler*.
- A Javascript method is a *statement*, that is an action to be performed on the emulator screen.

Once a transaction has been defined, it appears in the *Transactions* folder of the connector, in the **Projects View** of the CLI Studio (*"CLI Studio"* on page 1 - 13):

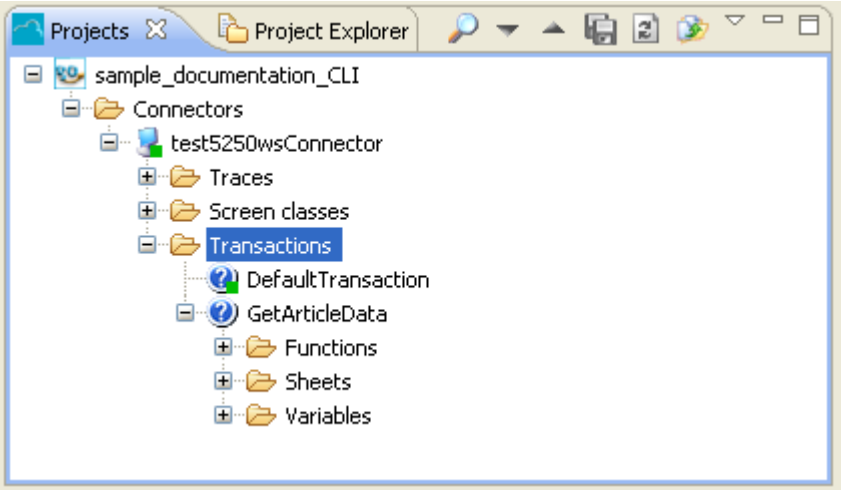


Figure 1 - 14: Transactions folder

SCREEN CLASS HANDLER

Basically, a screen class handler is meant to answer the following question : "Now that I have accessed this screen, what am I supposed to do?".

In other words, once a given screen class has been accessed and detected (depending on user-defined criteria), the corresponding screen class handler is triggered and performs user-defined actions (fill in a field, set cursor in a specific location of the screen, press `Enter` or a function key, etc.) on the legacy screen.

In CLI, screen class handlers are defined as JavaScript *functions*. JavaScript functions include *statements* (or *methods*) applied on the *javelin* object, which corresponds to the emulator. A JavaScript statement represents an action performed in the terminal emulator:

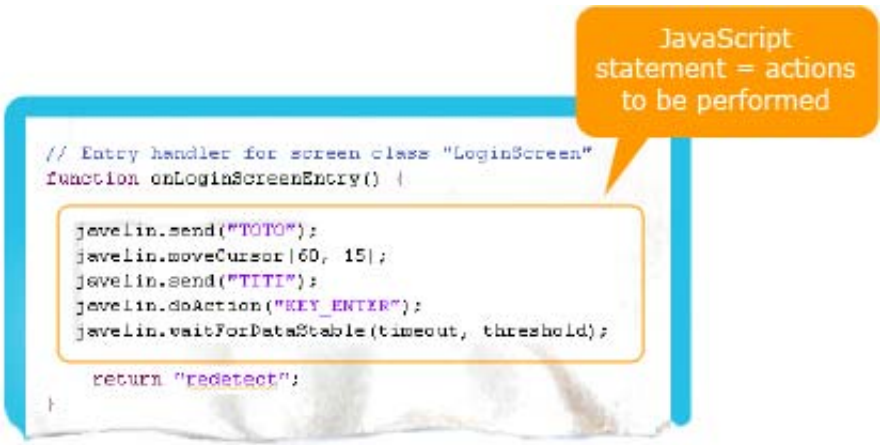


Figure 1 - 15: Screen class handler triggered on detection of the LoginScreen screen class

*For more information on the methods of the **javelin** object, see the complete Javelin Object Reference here.*

In CLI, two types of screen class handler can be defined for a given screen class:

Table 1 - 6: Screen class handler types

Screen class handler	Description
on<*ScreenClass*>Entry	Triggered when CWI detects a given <***ScreenClass***>.
on<*ScreenClass*>Exit	Triggered after extraction rules have been executed for a given <***ScreenClass***>.

Other handlers exist, but their use falls out of the scope of this guide.

Once a screen class handler has been created, it appears in the *Functions* folder of the transaction, in the **Projects View** of the CLI Studio (*"CLI Studio"* on page 1 - 13):

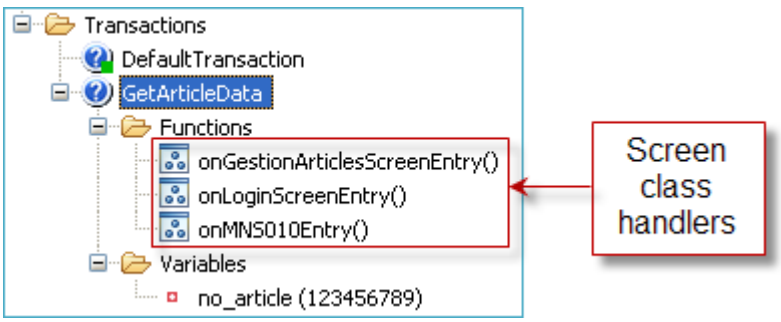


Figure 1 - 16: Screen class handler in transaction Functions folder

A screen class handler is a Javascript function. It can contain one or several statements (method calls).

1.3 CLI Studio

This section describes the CLI Studio, which comprises four distinct panes: the **Projects View**, the **Connector View**, the **Properties View** and the **Console View**.

- General Presentation
- View Description

1.3.1 General Presentation

Based on an Eclipse platform, the CLI Studio is divided into four views that all include:

- one or several tabs, located either on the upper or on the lower part of the view;
- icons;
- three standard Eclipse buttons (**Menu** , **Minimize** and **Maximize**)

The four views are illustrated and named in the figure below:

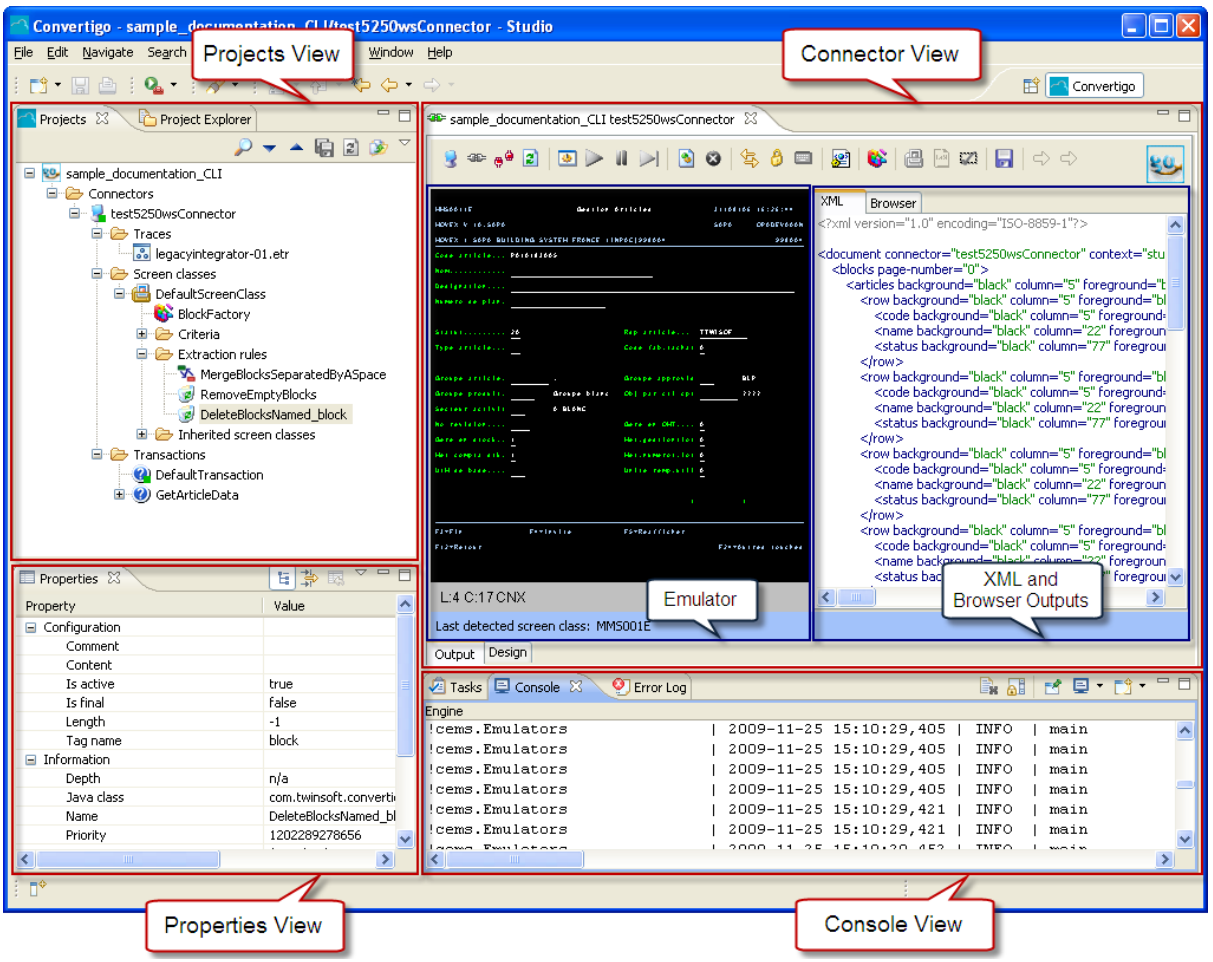


Figure 1 - 17: CLI Studio

The following table describes the views of the Convertigo Legacy Integrator studio:

Table 1 - 7: CLI views

View Name	Description
Projects View	Displays in a tree structure all CLI projects and their related objects: connectors, screen classes (including criteria, extraction rules, inherited screen classes and sheets) and transactions (including screen class handlers).
Properties View	Displays the properties of the CLI object selected in the **Projects View**. Used to modify a given object's properties.
Connector View / Editor	When loading a project, the **Connector View** automatically displays information related to the default connector set for the project. The **Connector View** is divided into three parts: • the **Emulator** - connects to the legacy application and displays legacy application screens; • the **XML output** - active only when **XML** tab is selected. Displays the XML output resulting from extraction; • the **Browser output** - active only when **Browser** tab is selected. Displays extracted data after XSL transformation. It is automatically updated as an **Editor** whenever files (in editable format such as .xml, .css or .xsl) are edited. **Document Editors** open editable documents such as transaction scripts in tabs.
Console View	Standard Eclipse console. Displays information about running tasks, errors, etc. Different consoles are available (standard output, Engine, Studio, etc.).

1.3.2 View Description

All four views contain common standard Eclipse buttons in the upper right corner of the view:

Table 1 - 8: Standard Eclipse buttons

Button	Icon	Description
Menu		Opens a drop-down menu whose content is identical to the view toolbar.
Minimize		Minimizes the current view without closing it. The view is still visible in the form of an icon that can be clicked to be enlarged again.
Maximize		Maximizes the current view to full screen. Maximizing a view can also be achieved by double clicking the upper bar of the view.

The following section describes the four panes of the CLI Studio in further detail.

PROJECTS VIEW

The **Projects View** displays all information (objects, folders and files) associated with a project. Information appears in distinct tree structures depending on the selected tab.

In addition to project tree structures and standard Eclipse buttons (see Table 1 - 8 on page 1-14), the **Projects View** includes tabs and icons:

Table 1 - 9: Projects View tabs and icons

View Element		Description
Tabs	Project	Convertigo view. Contains all projects and related objects. Objects are presented in a tree structure and contained in folders.
	Project Explorer	Standard Eclipse view. Contains all projects and related files. Files are represented in a tree structure.
Icons		*Find* icon. Search project tree for a specific object
		Decrease priority icon. Available only if a criterion or an extraction rule is selected. Decreases the priority of the selected object, meaning that the order according to which screen classes are detected or extraction rules are executed changes.
		Increase priority icon. Available only if a criterion or an extraction rule is selected. Increases the priority of the selected object, meaning that the order according to which screen classes are detected or extraction rules are executed changes.
		Save icon. Available only when at least one object has been modified. Saves all the project. All objects will be saved, and project xml file will be updated. **Remember to save projects on a regular basis.**
		Refresh icon. Available only if a project is selected. Refreshes Projects View by reloading project content from the hard drive. Identical to closing and re-opening project.
		Import icon. Imports a new project (in CAR or XML format) in the Studio.

In the **Projects View**, objects are formatted depending on their status:

- plain text - unchanged object;
- bold - object changed and unsaved;
- red - disabled object;
- grey - inherited object.

The figure below shows the different types of formatting in the **Projects View**.

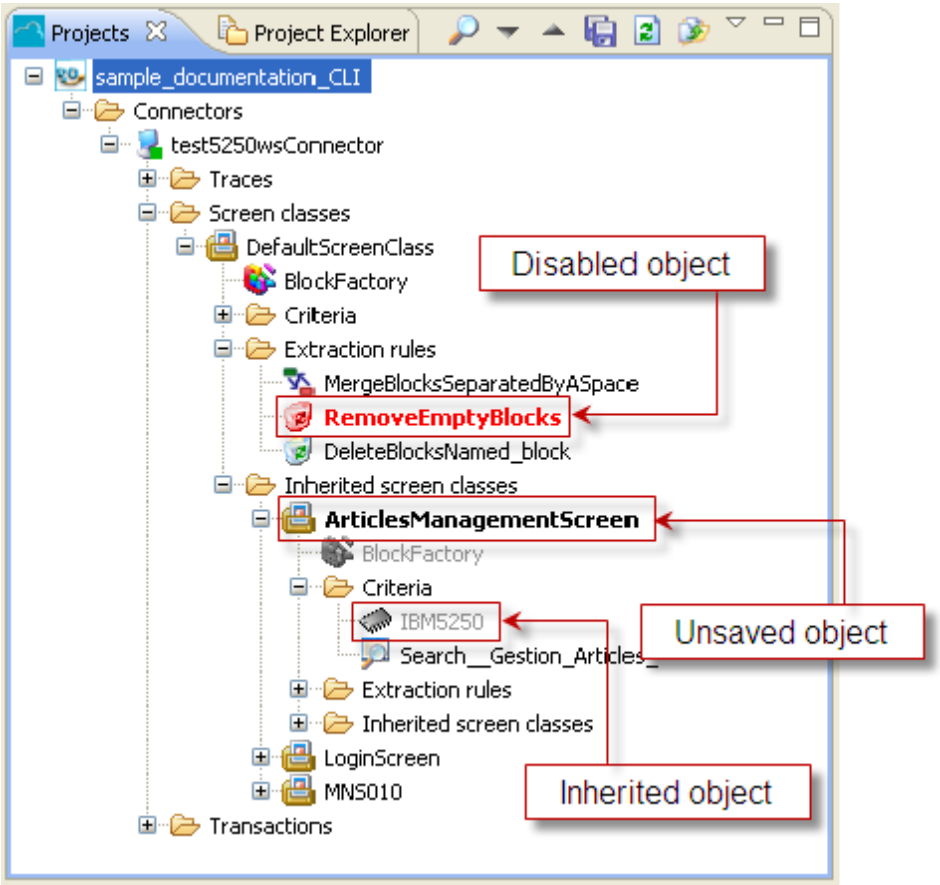


Figure 1 - 18: Formatting of unsaved, inherited and disabled objects in the Projects View

Once an object has been selected in the **Projects View**, its properties appear in the **Properties View**.

Properties View

The **Properties View** displays the properties of the object selected in the **Projects View**.

Properties depend on the type of the selected object. They are presented by categories (*Expert*, *Configuration*, *Information*, *Selection*, *Properties*, etc. - categories can be hidden by clicking on) that can be either expanded or collapsed by clicking respectively on or .

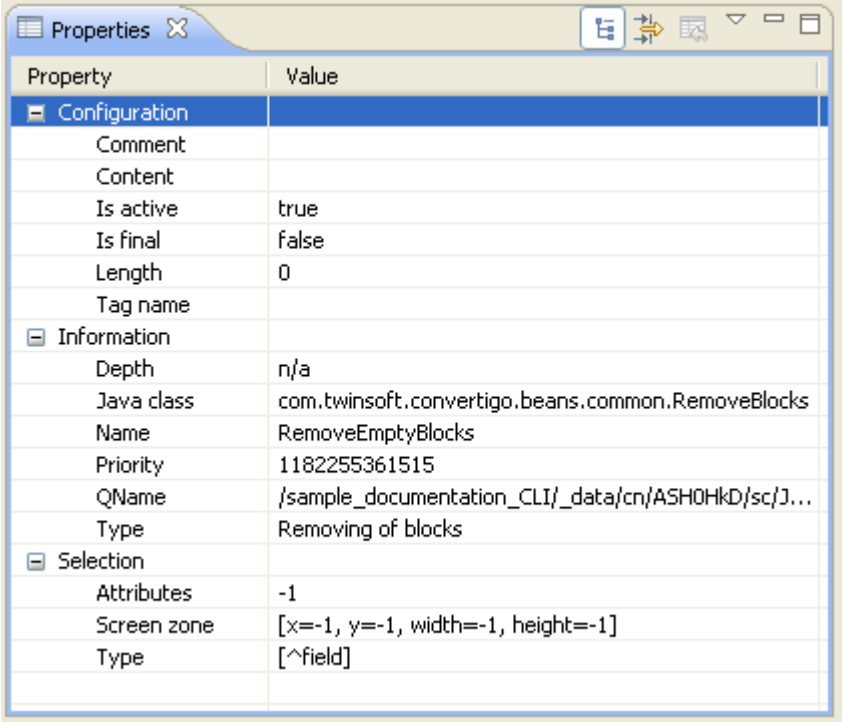


Figure 1 - 19: Properties View

In addition to object properties and standard Eclipse buttons (see Table 1 - 8 on page 1-14), the **Properties View** includes one tab (*Properties*) and icons:

Table 1 - 10: Properties View icons

Icon	Description
	Show Categories icon. Displays or hide property categories. When categories are hidden, properties appear in alphabetical order.
	Show Advanced Properties icon. Displays or hides advanced properties, if relevant. Not used for Convertigo objects properties.
	Restore Default Value icon. Available only when a property has been modified. Restores it to default value. Not used for Convertigo objects properties.

To edit properties

1 Left-click on the property value in the **Properties View**.

- the value is highlighted with white background; edit it by typing a new value in the field then pressing `Enter`:

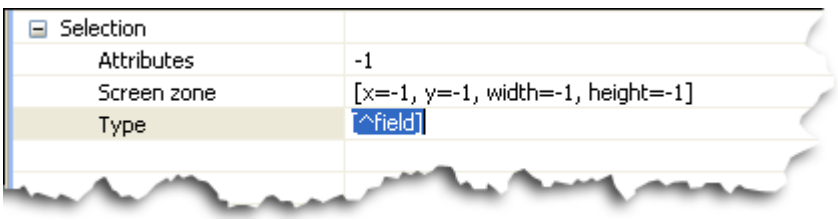


Figure 1 - 20: Text property field edition

- the value is highlighted with clear blue background; edit it by typing a new javascript

expression in the field then pressing `Enter`:

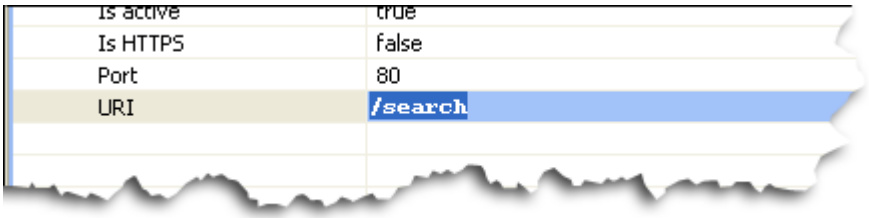


Figure 1 - 21: Javascript expression field edition

- the value is higlighted and a symbol appears on the right of the field:

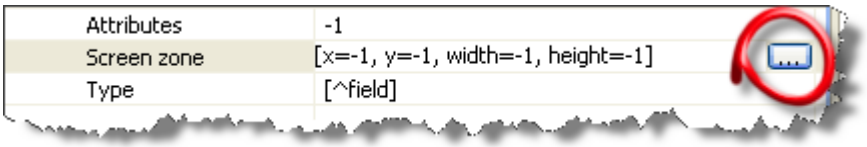


Figure 1 - 22: Text property edition via edition window

Click on . Depending on the property type:

A. A Javascript expression window or an edition window opens:

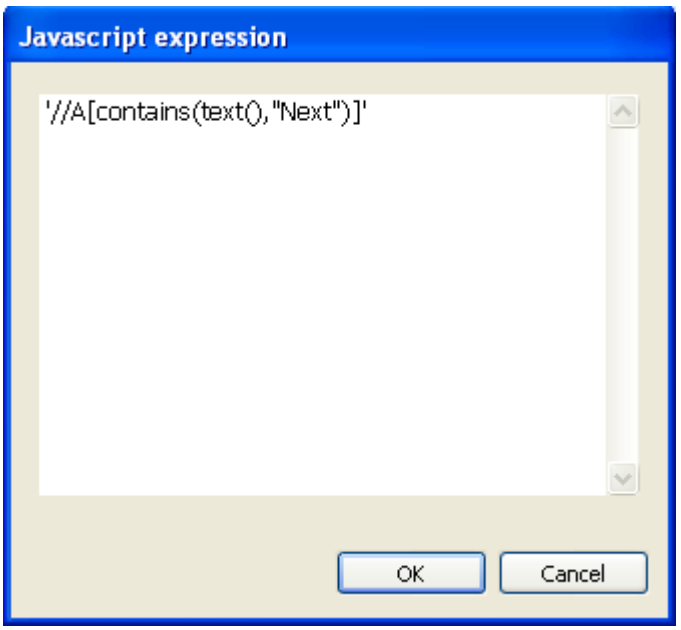


Figure 1 - 23: Javascript expression window

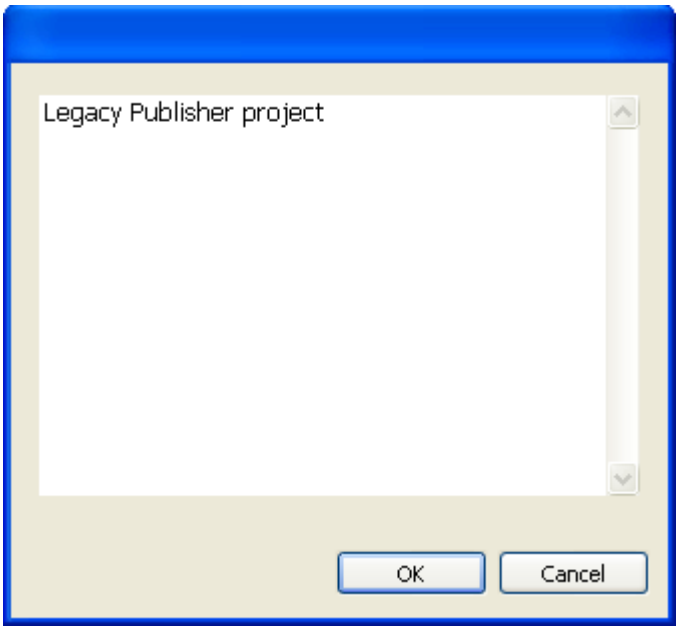


Figure 1 - 24: Edition window

1. Enter the new javascript expression or the text comment in the window.
2. Click on **OK**.

B. A configuration window opens (in this example, the **Attributes** window for extraction rules):

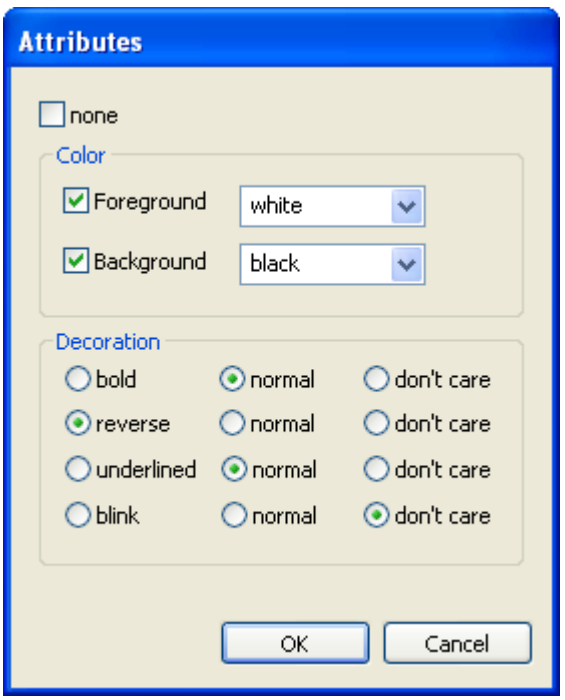


Figure 1 - 25: Configuration window

1. Set the parameters in the different fields and menus.
2. Click on **OK**.

- the value is higlighted and a symbol appears on the right of the field:

Figure 1 - 26: Text property edition via drop-down menu

1 Click on . A drop-down menu appears:

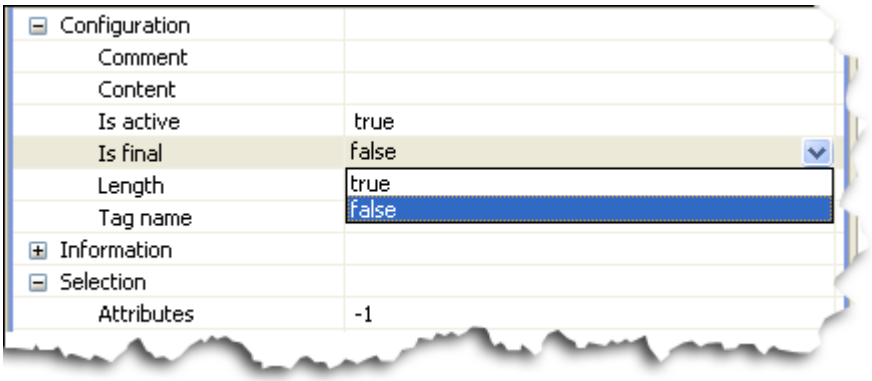


Figure 1 - 27: Selecting a value in the drop-down menu

2 Select a value in the drop-down menu.

3 Press `Enter`.

Connector View

The **Connector View** displays the screens of the emulated legacy application. Information displayed on screens depends on the emulator technology set for the project (IBM 5250, IBM 3270, BULL DKU) and on the accessed legacy application.

The **Connector View** is divided into three parts:

Table 1 - 11: Connector View parts

Name	Description
Emulator	Connects to the legacy application. Displays legacy application screens, which are divided into blocks for extraction purpose (see *"Block Factory"* on page 1-2).
XML output	Displays XML code resulting from extraction, after all extraction rules have been executed.
Browser output	Not applicable to Convertigo Legacy Integrator.

As other views, the **Connector View** also includes tabs and icons:

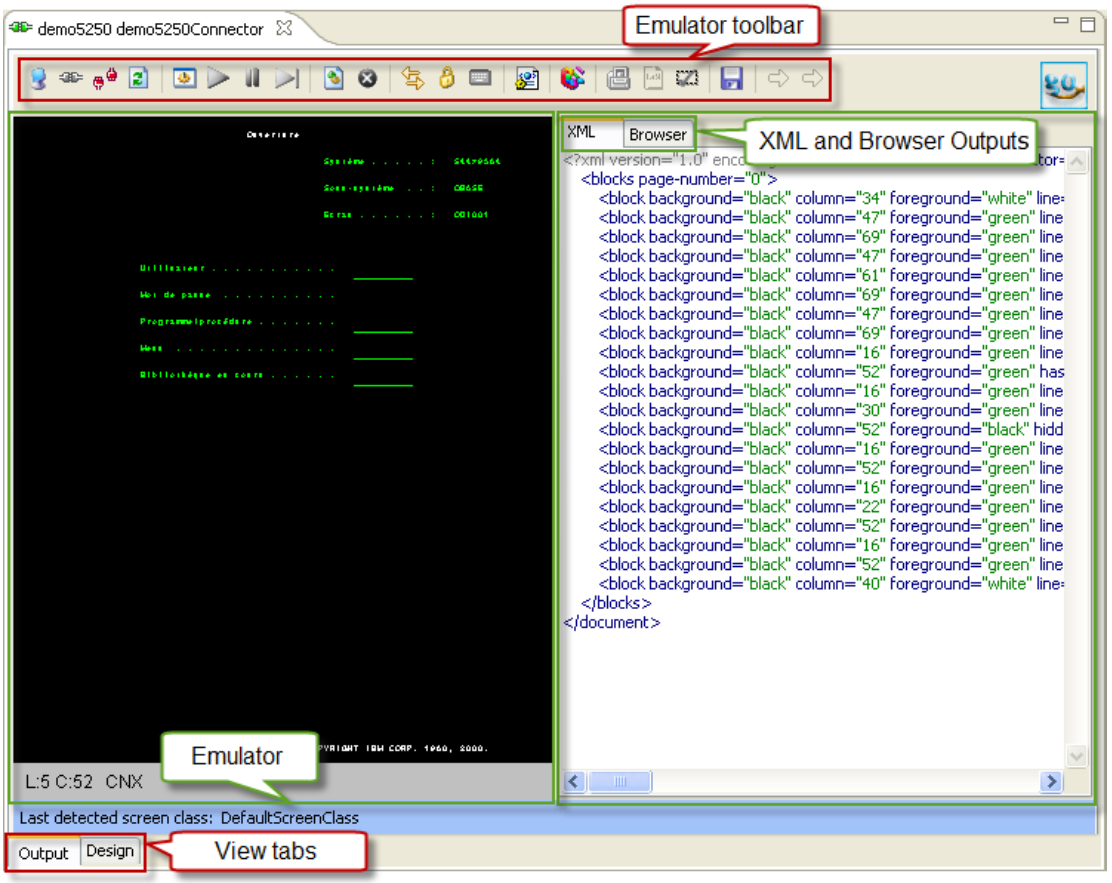


Figure 1 - 28: Connector View (Output tab selected by default)

The **Output** tab is selected by default. The **Design** tab does not apply to this module of the Convertigo Enterprise Mashup Studio.

The table below describes the **Connector View** icons that are available when the **Output** tab is selected:

Table 1 - 12: Connector View icons

Icon	Description
	Renew the connector icon. Renews the connection according to the connector parameters set for the project.
	Connect the connector icon. Connects the emulator to the application defined in the connector of the project.
	Disconnect the connector icon. Disconnects the emulator from the application defined in the connector of the project.
	Refresh connector content icon. Refreshes the content of current legacy application screen.
	Debug mode icon. Activates debug mode. In Debug mode, the XML generation process (mainly through execution of extraction rules) pauses after each step.

Table 1 - 12: Connector View icons (...)

Icon	Description
	Continuous debug mode icon. Activates continuous debug mode. When in debug mode, CLI proceeds with XML extraction by executing remaining steps without pause until XML is generated or process is manually paused again.
	Pause the debug process icon. Pauses XML extraction. Available only when continuous debug mode is activated.
	Step-by-step debug mode icon. Activates step-by-step debug mode.
	Generate XML icon. Generates the XML code of the Web page currently displayed in the **Emulator** by executing the related extraction rules.
	Stop transaction icon. Stops the current transaction (while a transaction is in process).
	Show current screen class icon. Highlights in light grey in the **Projects View** the screen class detected for the legacy application screen currently displayed in the **Emulator**.
	Link current selection to highlighted property in Properties panel icon. Shortcut command for automatically setting screen zone property (**Screen zone**, **Image layout**, etc.) coordinates.
	Add element from current selection to highlighted property in Properties panel icon. Shortcut command for automatically adding a column to a table when setting a *Table* extraction rule.
	Generate XSL icon. Generates an XSL style sheet for a given screen class. The XSL style sheet is based on the XML output generated for this screen class, and ready for edition using any editor.
	Show blocks icon. Show the block structure of currently displayed legacy screen.
	New ScreenClass icon. Creates a new screen class based on detection criterion selected on screen.
	New TagName icon. Creates a new tag name extraction rule based on selected screen element....
	New WaitAt icon. Available for Videotex, VT connectors only. Shortcut command for generating the code of a synchronisation based on appearance of an element on screen.
	Record trace icon. When activated, CLI starts recording a trace. Stops when icon is clicked on a second time. The trace is stored in the **Traces** folder of the connector in the following format: `YYYY-MM-DD_trace.etr`.
	Learn icon. Available only when a transaction is selected. Activates the *Learn* mode - any action performed in the emulator will be recorded and screen classes entry handlers generated. For advanced users only.
	Accumulate learning mode icon. Available only when *Learn* mode is on. Accumulate data on visited legacy application screens. Exit screen class handlers will automatically be created in the transactions. For advanced users only.

EMULATOR

The (terminal) **Emulator** displays the screens of the legacy application to which CLI connects according to connector parameters set when creating a project.

In CLI, a number of actions are directly performed in the **Emulator** screen, including selecting elements on screen with the mouse to create, for example:

. new *screen classes*, through the **New ScreenClass** icon ;

. new *tag name* extraction rules, through the **New TagName icon** .

As mentioned in Table 1 - 12 on page 1-21, specific screen-related properties of objects selected in the **Projects View** can be directly set from the **Emulator** using the **Link current selection to highlighted property in Properties panel** icon .

Fo example, when setting the **Screen zone** property of an extraction rule (see Table 1 - 5 on page 1-10), select the screen zone in the **Emulator** then click on . The **Screen zone** property value is automatically set.

When the terminal emulation session is active, all keys associated with the emulated technology are available from the CLI application. For example, if the emulated terminal `F3` function key is associated with a "display general menu" command", pressing `F3` on the keyboard from CLI will display the legacy application general menu in the **Emulator** (see *"Connector View"* on page 1-20).

*The prerequisite for enabling keyboard activation in the terminal emulator is to set the focus in the **Emulator** view first by double-clicking in it.*

XML OUTPUT

The **XML Output** displays the XML document resulting from the execution of all extraction rules associated with a legacy application screen class.

For example, if we consider the following legacy application screen, corresponding to a given screen class (`MMS001E` screen class):

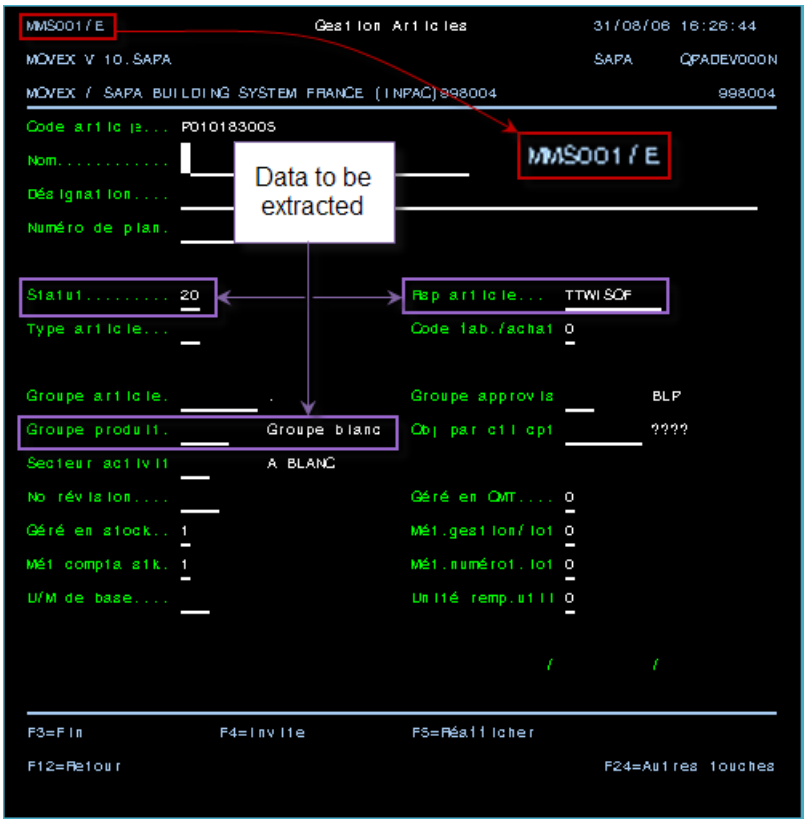


Figure 1 - 29: Example legacy application screen

The `MMS001E` screen class is associated with three *XML tag name* extraction rules:

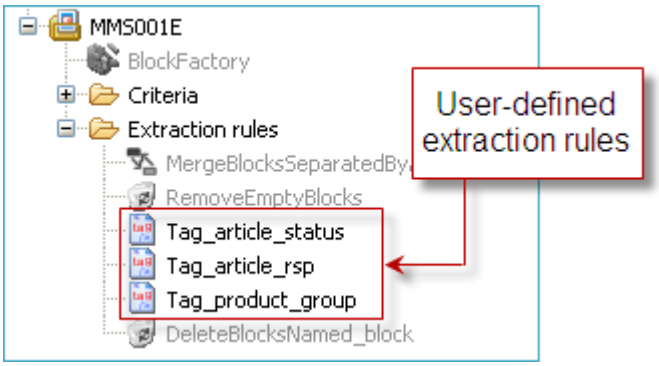


Figure 1 - 30: MMS001E screen class extraction rules

After the `MMS001E` screen class has been detected and extraction rules have been executed, the XML representation of extracted data appears in the **XML tab** of the **Connector View**:

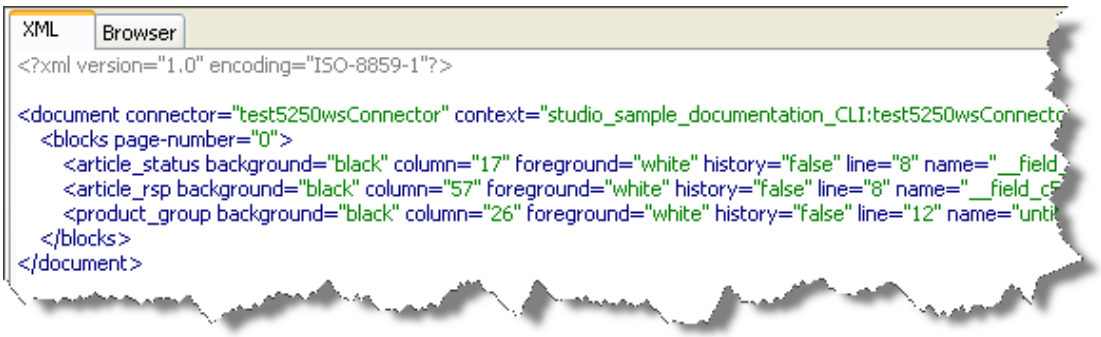


Figure 1 - 31: XML output resulting from execution of tag name extraction rules

2 My First Convertigo Legacy Integrator Project

This chapter gives instructions on how to set up a simple CLI project.

- Legacy Integrating Project
- Setting up the project

2.1 Legacy Integrating Project

This section describes the sample Legacy Publishing project and how to open the completed sample project from the Studio:

- Project Description
- Opening the Sample Project from the Studio

2.1.1 Project Description

This Legacy integrating project (called `sample_doc_CLI`) is based on an article management legacy application. Its purpose is to:

- *Emulate* a connection to an IBM 5250 mainframe hosting the application.
- *Log in* to the emulated legacy application (by entering user name and password).
- *Navigate* through legacy application screens by inputing data (menu choice, article code).
- *Extract* selected data using various extraction rules (*XML tag name* and *Table*).

A *transaction* (see *"Transaction"* on page 1-10) using as input variable an *article code* is created at the end of the project to automate these actions.

The process can be summed up as follows:

1 Connect to legacy application login screen:

Figure 2 - 1: Legacy application login screen

2 Enter name and password:

Figure 2 - 2: Input of username and password into login fields

3 A general menu opens. Type in the code for accessing article management pages ("110"):

Figure 2 - 3: Input of article management code (general menu)

4 The article management screen is acessed. Extract the list of referenced articles in a

three-column table:

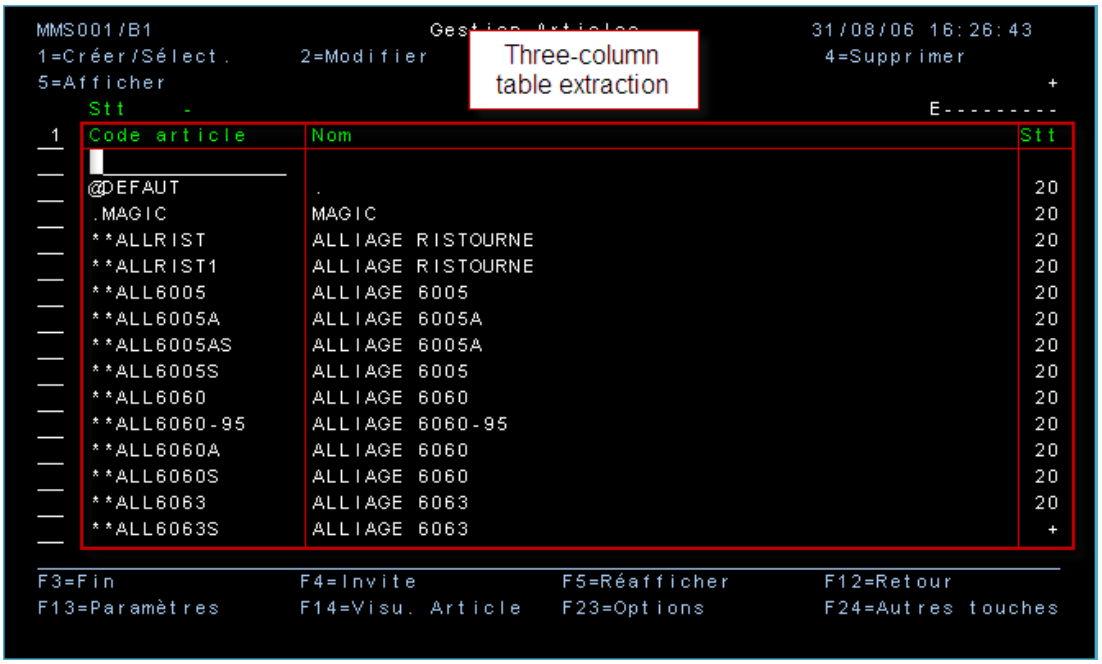


Figure 2 - 4: Extraction of article list (on execution of Table-type extraction rule)

5 In the same page, enter an article code (which is set as input variable of the transaction at the end of the project):

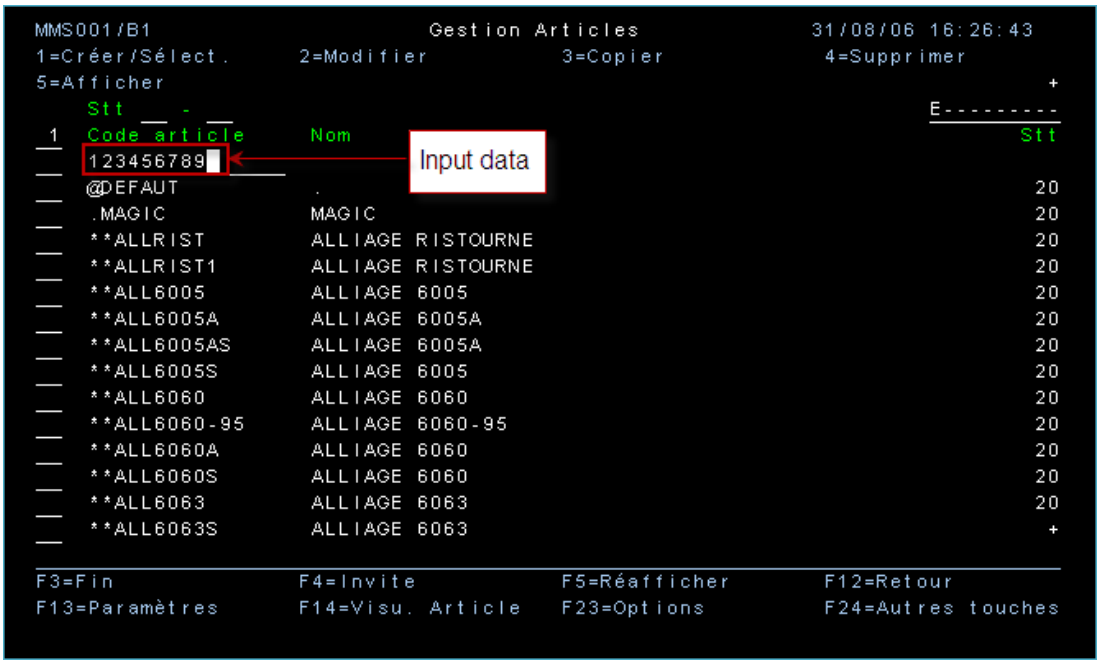


Figure 2 - 5: Input of article code

6 The article description screen is accessed. Extract *article status*, *article rsp* and *product group* data:

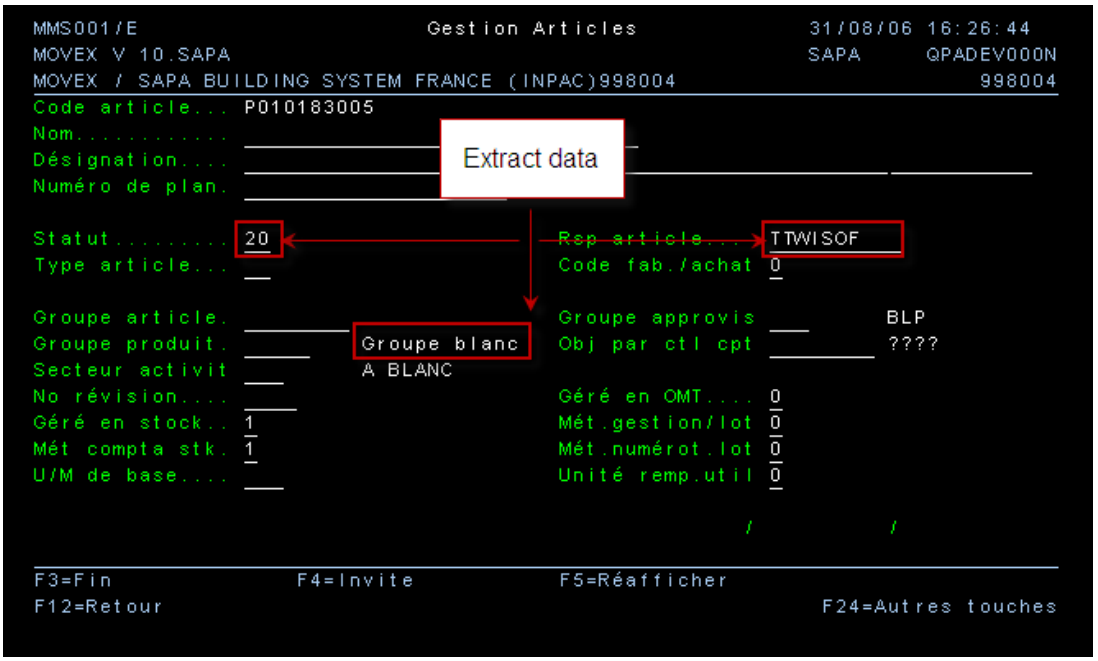


Figure 2 - 6: Extraction of article specific data (on execution of XML tag name-type extraction rules)

Input, navigation and extraction tasks will be automated through the setting of:

- a *connector* in charge of emulating a connection to the legacy application;
- *criterion-based screen classes* identifying each legacy screen accessed throughout the navigation;
- *extraction rules* associated with and executed on detection of screen classes;
- a *variable-based transaction* containing *screen class handlers* triggering actions on screens (move cursor, press `Enter`, type in username, etc.).

Procedures for setting mentioned objects are given in the section 2.2.

2.1.2 Opening the Sample Project from the Studio

The completed sample project is stored in the application installation folder.

The following procedure describes how to access it from the Studio using the **New Project** wizard.

To open the sample project from the Studio

1 In the **Studio** toolbar, click on ***New > Project***:

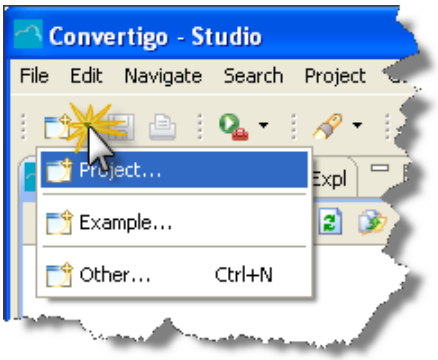


Figure 2 - 7: Launching the New Project wizard

A **New Project** wizard opens.

2 Expand **Convertigo Projects**, then **Documentation samples**, and select **Legacy integrator**:

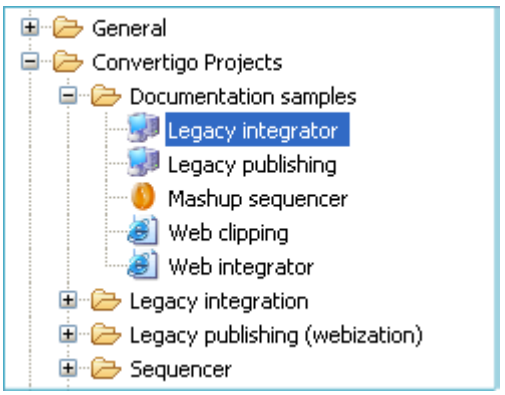


Figure 2 - 8: Selecting the CLI sample project

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

The sample project opens in the **Projects View**:

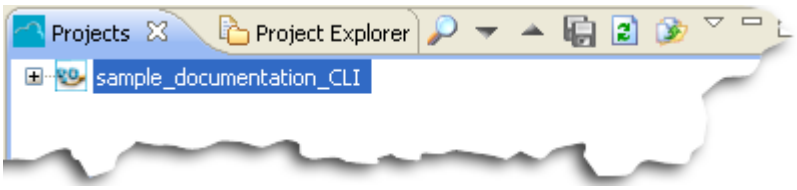


Figure 2 - 9: Sample project in Projects View

You can now start setting up your own CLI project, `sample_doc_CLI`.

2.2 Setting up the project

The following sections explain step by step how to set up your first Legacy integrating project:

- Creating a Project and Setting Connector Parameters
- Defining Screen Classes
- Defining a Transaction
- Executing a Transaction using the Test Platform

2.2.1 Creating a Project and Setting Connector Parameters

This section describes how to create a project and set connector parameters.

To create a project and set a connector

1. Launch the **Convertigo Studio**.

2. Select ***File > New > Project*** or click on in the toolbar then select *Project.*

   A **New Project** wizard opens:

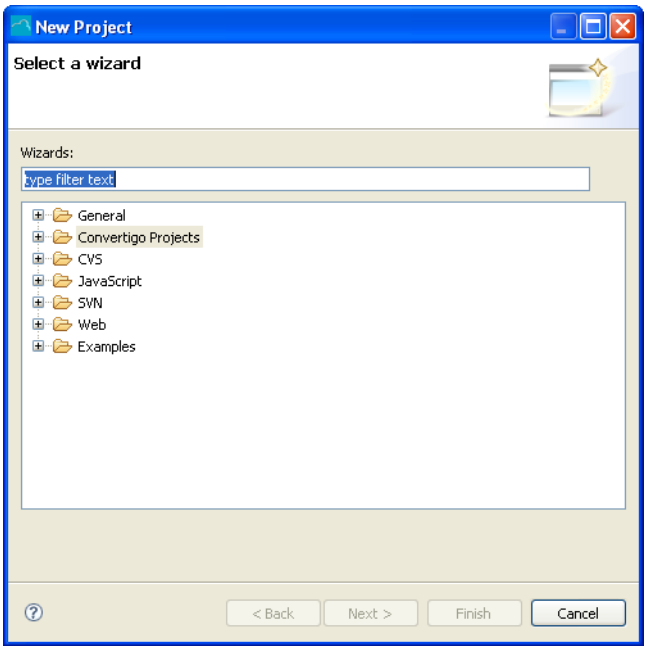


Figure 2 - 10: New Project Wizard

3. Expand **Convertigo Projects**, then **Legacy integration**, and select **5250 Screen**:

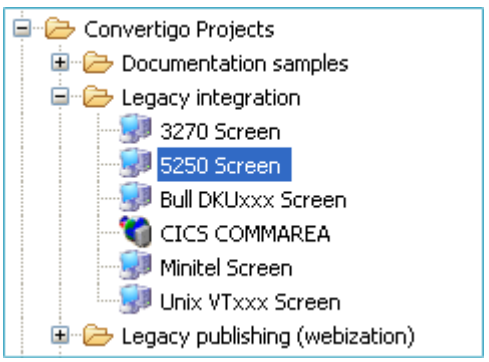


Figure 2 - 11: Selecting a project type

4. Click on **Next**:

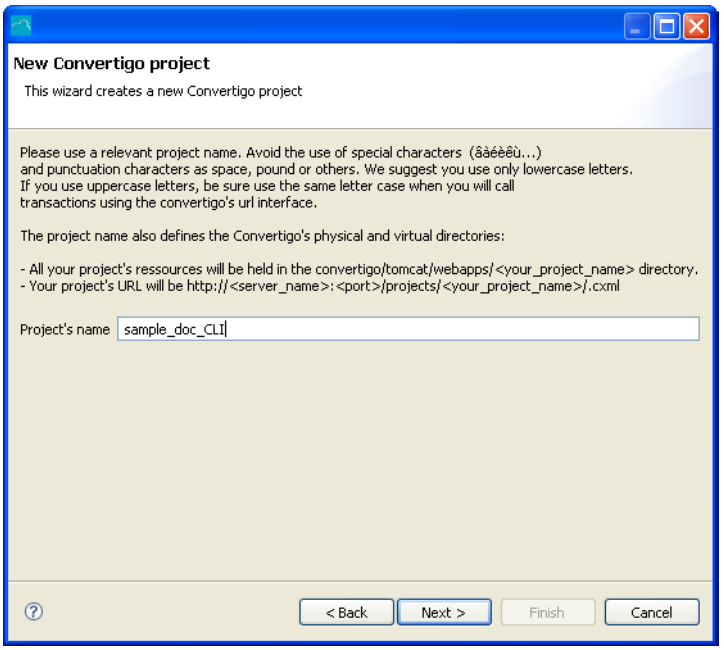


Figure 2 - 12: Entering a project name

5 In the **Project's name** field, type in the project name (for example `sample_doc_CLI`).

Project names are part of the URL adress used for calling a project from a Web browser. It is therefore recommended to avoid project names with blank spaces, special characters, lower and upper case letters, etc.

6 Click on **Next**:

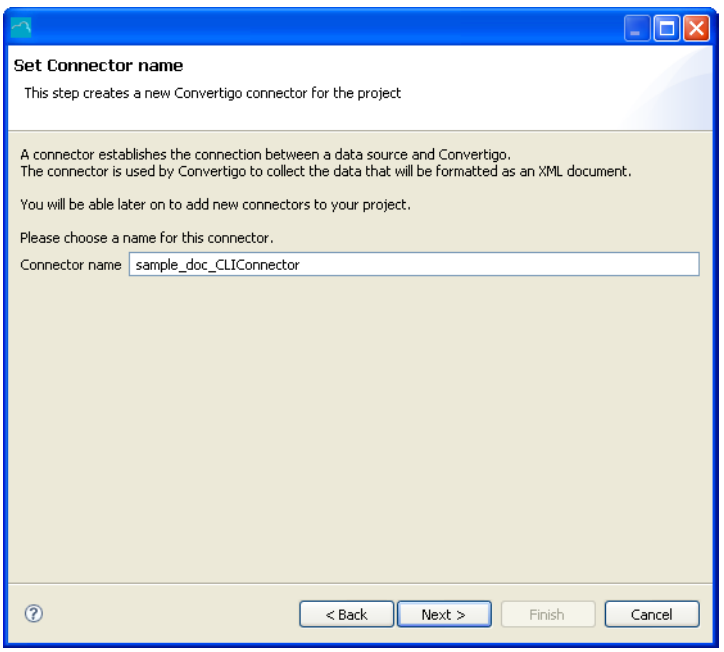


Figure 2 - 13: Entering a connector name

The **Connector name** field is filled by default with the connector name `sample_doc_CLIConnector`. You can choose another name, (for example `demo5250Connector`).

7 Click on **Next**:

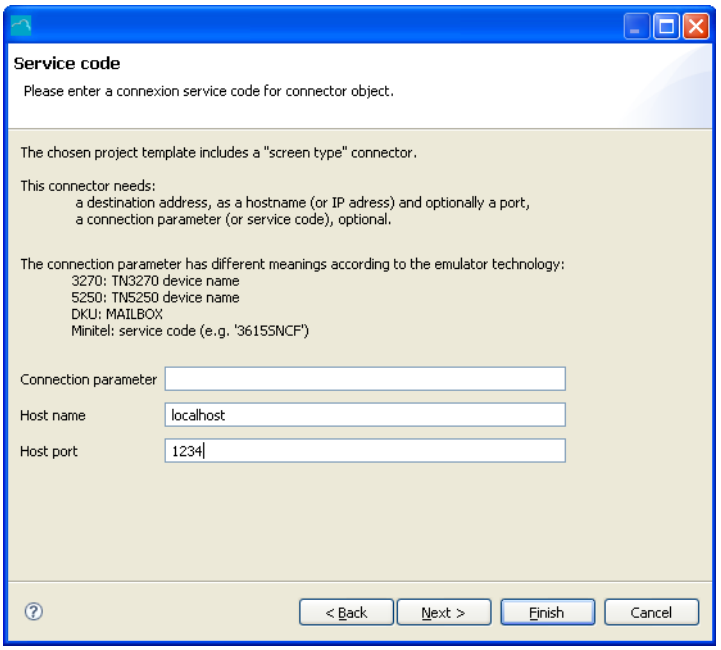


Figure 2 - 14: Setting connection parameters

8 Enter required information:

- in the **Connection parameter** field, type in the name of the device, mailbox or service code to which CLI will connect (optional parameter).
- in the **Host name** field, type in the host name or IP address of the terminal.
- in the **Host port**, type in the port of the terminal.

9 Click on **Next**.

The **New project summary** window opens:

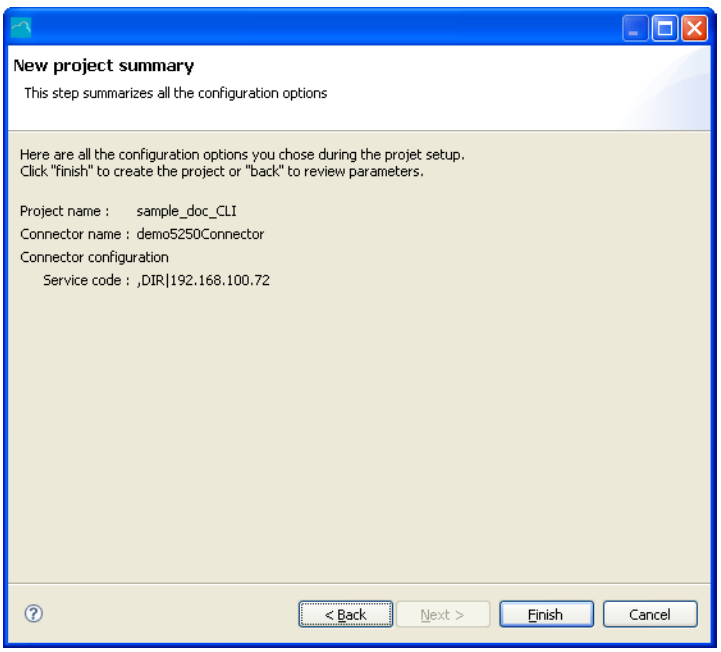


Figure 2 - 15: New project summary window

10 Check that all settings are correct, then click on **Finish**. The project now appears in the **Projects View**:

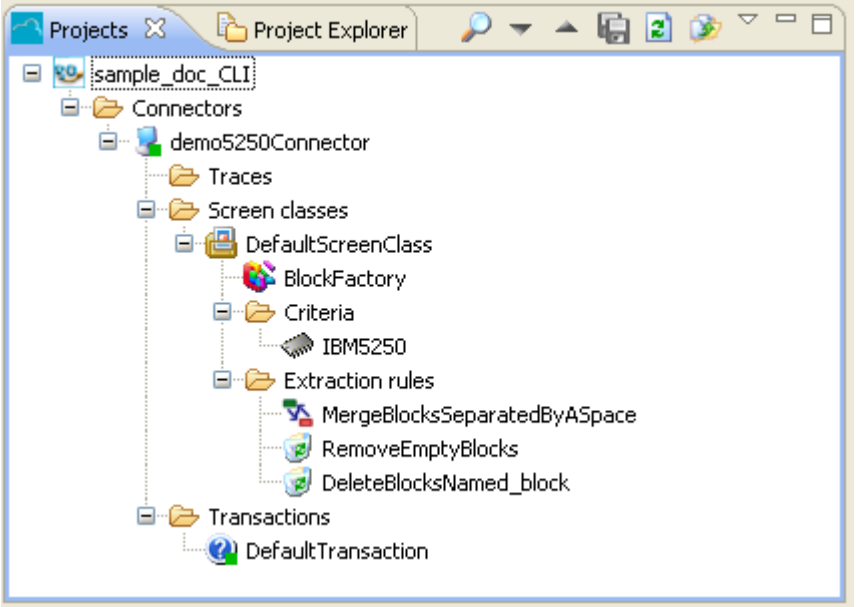


Figure 2 - 16: sample_doc_CLI project appearing in the Projects View

The legacy application login screen opens in the **Emulator** of the **Connector View**:

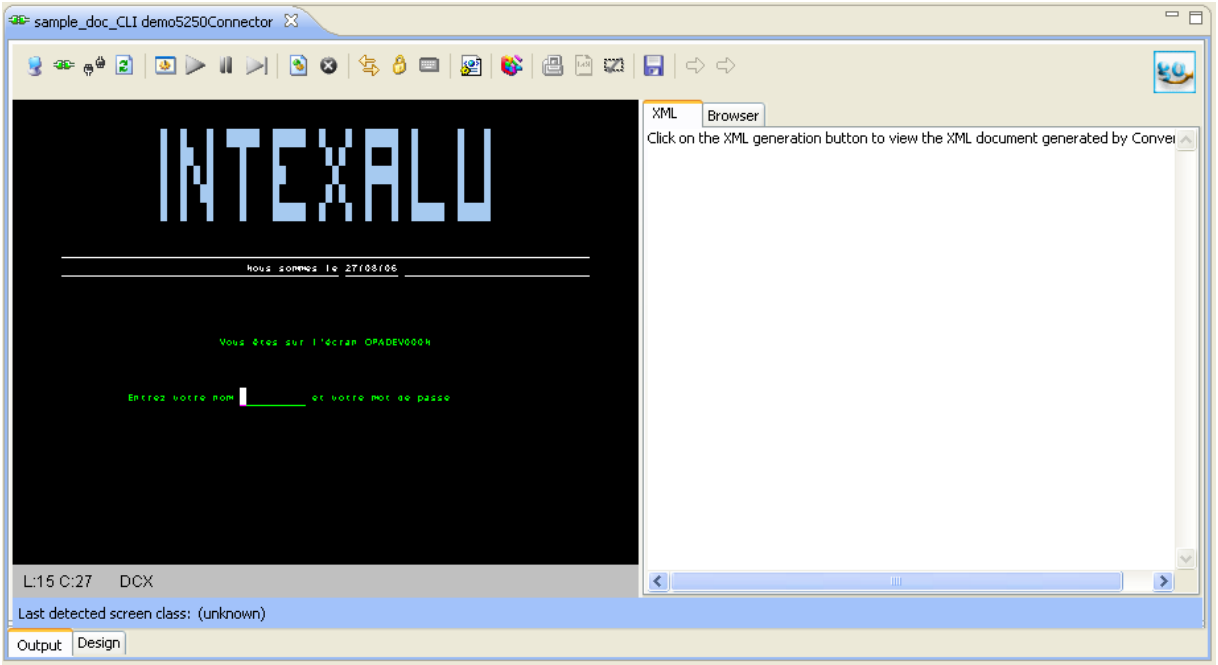


Figure 2 - 17: Connector View displaying legacy application login screen

This is the starting point of the project.

WHAT COMES NEXT?

The following step will consist in defining all screen classes corresponding to legacy application screens accessed throughout the project, using *detection criteria* (in other words, elements in screens - for example text strings or screen identifiers - defining legacy application screens in a unique manner).

When CLI accesses a screen, detection criteria are checked; if they match predefined criteria, then CLI "knows" which screen class has been accessed. *Extraction rules* associated with the detected screen class are executed. In parallel, screen class handlers trigger actions defined for this screen class.

2.2.2 Defining Screen Classes

This section shows how to create and define screen classes.

To define a screen class, you must first find, on an accessed legacy screen, which element can be considered sufficiently relevant to serve as detection criterion. Using a screen element as detection criterion for a screen class means:

1 *Selecting* the element on the screen by defining a capture zone with the mouse.

2 *Creating* the screen class based on this element.

DEFINED SCREEN CLASSES

In our sample project, five screen classes are needed. The following table describes these screen classes:

Table 2 - 1: Defined screen classes

Name	Description	Detection criteria
`LoginScreen`	Login screen class. Corresponds to first legacy application screen accessed in this project.	**Find string** type. Text string: "`Entrez votre nom`".
`MNS010`	Article management general menu screen class. Corresponds to second screen accessed in this project	**Find string** type. Text string: "`MNS010`".
`ArticlesManagementScreen`	Article management generic screen class. Corresponds to any article management screen.	**Find string** type. Text string: "`Gestion Articles`".
`MMS001B1`	Article list screen class. Child to article management generic screen class. Corresponds to third screen accessed in this project.	**Find string** type. Text string: "`MMS001/B1`".
`MMS001E`	Article description screen class. Child to article management generic screen class. Corresponds to fourth screen accessed in this project.	**Find string** type. Text string: "`MMS001/E`".

All five screen classes are children screen classes of the `DefaultScreenClass` screen class (see *"Screen Class"* on page 1-5). As such, they inherit its default extraction rules and criterion (the *emulator technology*).

The following figure shows, for each legacy application screen, which element will be selected on screen as detection criterion prior to creating a given screen class.

LOGINSCREEN SCREEN CLASS

Figure 2 - 18: "Entrez votre nom" text string (field name)

MNS010 SCREEN CLASS

Figure 2 - 19: "MNS010" text string (screen identifier)

ARTICLESMANAGEMENTSCREEN SCREEN CLASS

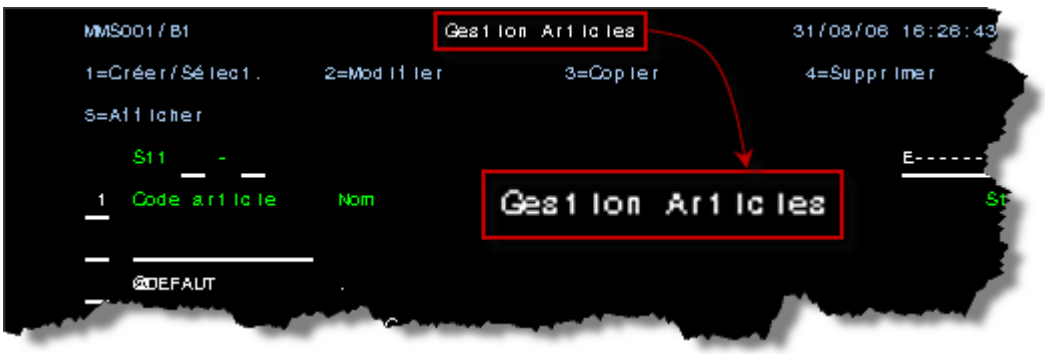


Figure 2 - 20: "Gestion Articles" text string (screen title)

MMS001B1 SCREEN CLASS

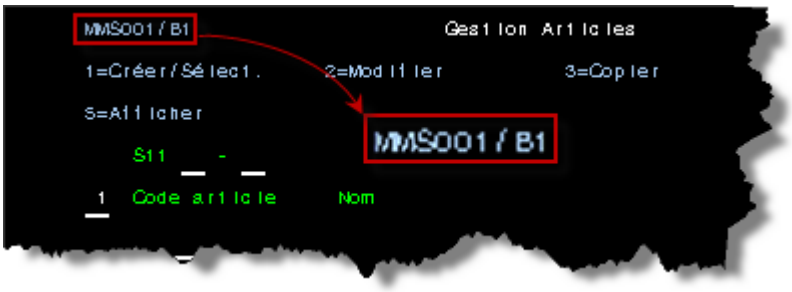


Figure 2 - 21: "MMS001B1" text string (screen identifier)

MMS001E SCREEN CLASS

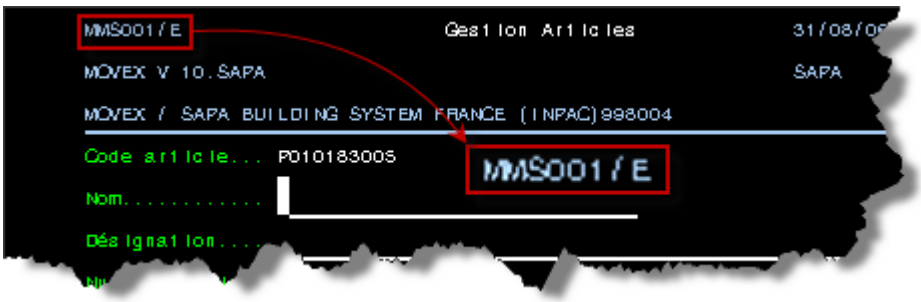


Figure 2 - 22: "MMS001E" text string (screen identifier)

CREATION OF FIRST SCREEN CLASS

The first screen class to be created in this project is the `LoginScreen` screen class. It corresponds to the first legacy application screen to be accessed in the project. The basic steps for creating this screen class are the following:

1 Assuming that the legacy application login screen is displayed in the **Emulator**, *define* a zone including the "Entrez votre nom" text string with the mouse.

2 *Create* a new screen class based on this text string by clicking on the **New ScreenClass** icon in the **Emulator** toolbar.

To create the LoginScreen screen class

1 Connect to the legacy application by clicking on the **Connect the connector** icon .

A legacy application screen is displayed in the **Emulator** (in our project, a login screen):

Figure 2 - 23: Sample project login screen of emulated legacy application

2 Using the mouse, define a screen zone including the "`Entrez votre nom`" text string. The screen zone is outlined in white:

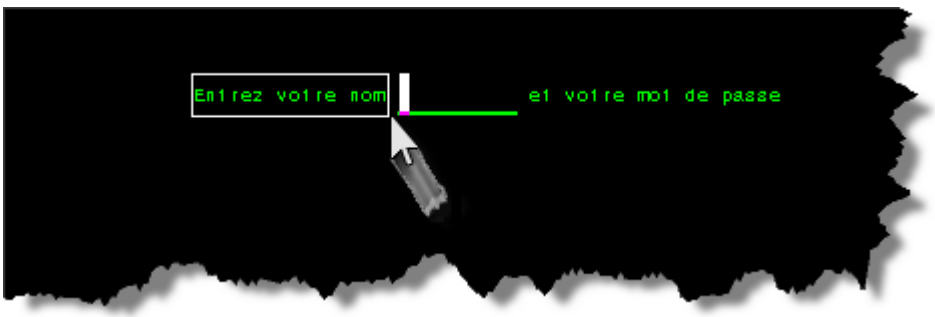


Figure 2 - 24: Selecting the LoginScreen screen class detection criterion on screen

In the **Emulator** toolbar, the **New ScreenClass** and **New TagName** icons become active.

3 Click on the **New ScreenClass** icon .

A new screen class is automatically generated in the **Projects View** tree structure. It is called after the text string selected as detection citerion: `Entrez votre nom`. The selected criterion of **Find string** type (see *"Criterion"* on page 1-6) appears in the **Criteria** folder of the newly created screen class:

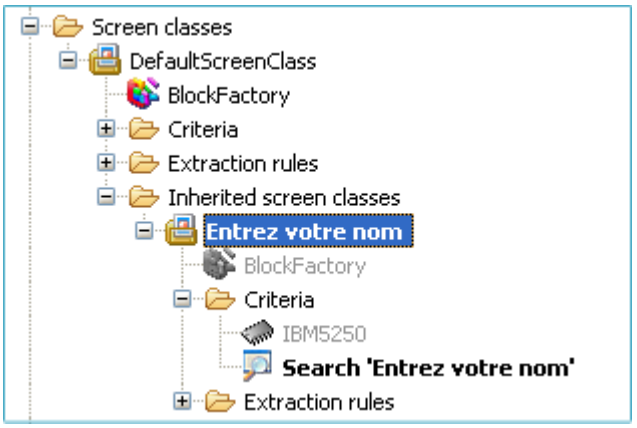


Figure 2 - 25: Newly created screen class (Entrez votre nom) and associated detection criterion

Default screen class names can be changed to reflect more clearly the screen class that they represent. For example, in our project, we can decide to rename the `Entrez votre nom` screen class, which identifies login screens, into `LoginScreen`.

To rename a screen class

1. Select the screen class with a left-click and press `F2` or right-click on the screen class and select **Rename** in the drop-down menu:

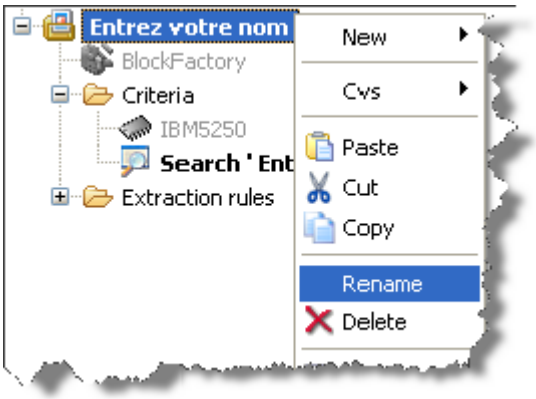
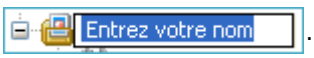


Figure 2 - 26: Renaming a screen class

The screen class name is highlighted in blue: 

.

2. Type in a new name (for example `LoginScreen`) and press `Enter`.

3. Save the project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

Remember to save your project on a regular basis.

The newly created screen class is now renamed.

This screen class inherits its parent screen class (`DefaultScreenClass`) criterion and extraction rules, which appear greyed in the **Criteria** and **Extraction rules** folders of the `LoginScreen` screen class:

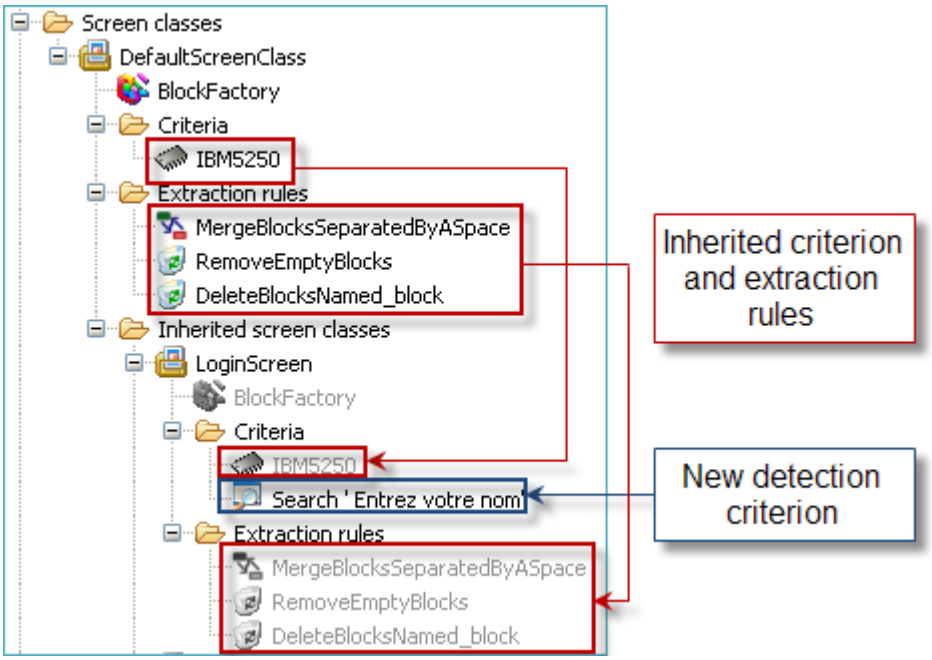


Figure 2 - 27: Renamed screen class (LoginScreen) - inherited and new objects

Once a screen class has been created, check that the screen from which it has been created is detected as belonging to the correct screen class, by clicking on .

To check that a screen is detected as belonging to the defined screen class

Assuming that the **Emulator** displays the login screen of our sample project, click on the **Go to current screen class object** icon in the **Emulator** toolbar:

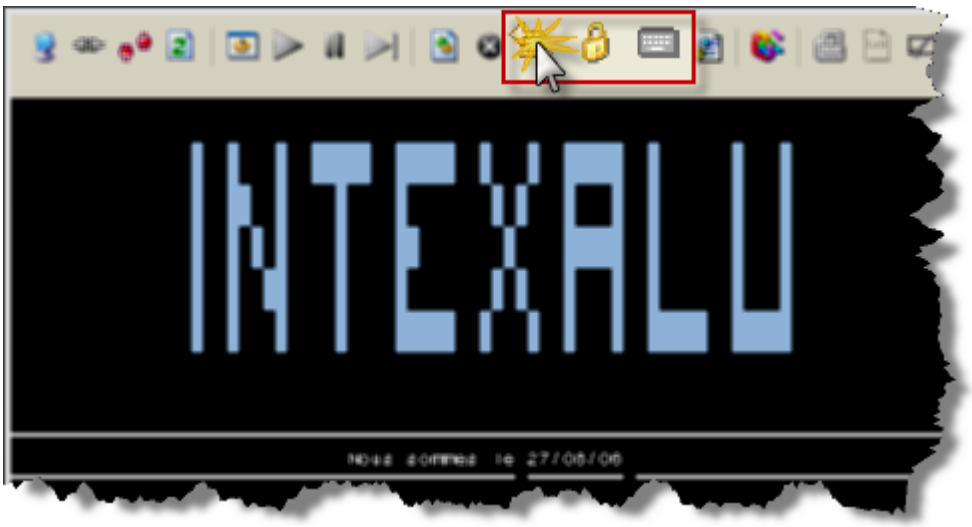


Figure 2 - 28: Clicking Go to current screen class icon

Expected result: The `LoginScreen` screen class object is highlighted in the **Projects View**.

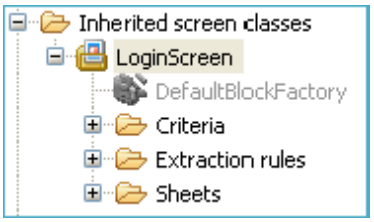


Figure 2 - 29: Highlighted screen class after Go to current screen class object command

WHAT COMES NEXT ?

The first screen class is now created, renamed and properly set.

The process for creating the other four screen classes is identical to the process for creating the `LoginScreen` screen class:

1 When accessing a new screen, *delineate* a screen zone including the detection criterion.

2 *Click* on to create the new screen class.

3 If needed, *rename* the screen class.

It is recommended to avoid special characters and blank spaces when naming a screen class.

CREATION OF ADDITIONAL SCREEN CLASSES

As mentioned in *"Defined Screen Classes"* on page 2 - 10, four other screen classes are to be created. These are, in appearing order in the project: `MNS010`, `ArticlesManagementScreen`, `MMS001B1` and `MMS001E`.

The following procedures describe how to create, one after another and based on the same logical process, those four additional screen classes.

To create the MNS010 screen class

1 Assuming that the **Emulator** displays the proper screen (in our project, the second accessed screen, i.e. the application general menu screen identified by the "`MNS010/B4`" screen identifier), define a screen zone including the required text string (for example "`MNS010`") using the mouse. The screen zone is outlined in white:

Figure 2 - 30: Selecting the MNS010 screen class detection criterion on screen

The **New ScreenClass** and **New TagName** icons become active.

2 Click on the **New ScreenClass** icon .

A new screen class is automatically generated in the **Projects View** tree structure. It is

called after the text string selected as detection citerion: `MNS010`. The selected criterion of **Find string** type (see *"Criterion"* on page 1-6) appears in the **Criteria** folder of the newly created screen class:

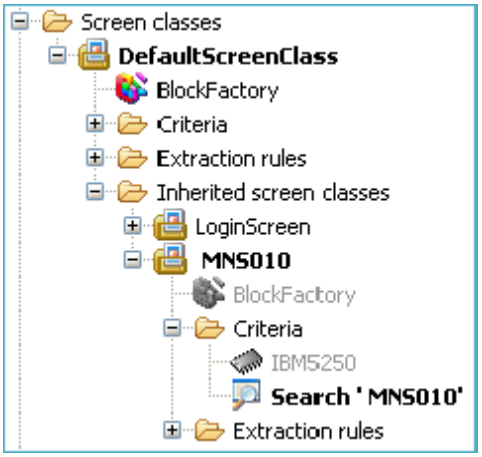


Figure 2 - 31: Newly created screen class (MNS010) and associated detection criteria

The screen class name (`MNS010`) corresponds to a screen identifier. It speaks for itself and does not need to be changed.

3 Save the project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

The screen class identifying the second screen (application general menu) accessed in the project is now created.

After the article management code is entered on this page, the article management screens are accessed. The following procedure describes how to create the screen class corresponding to generic article management screens: `ArticlesManagementScreen`.

To create the ArticlesManagementScreen screen class

1 Assuming that the **Emulator** displays the proper screen (in our project, the third accessed screen, i.e. the article list screen including a "`Gestion Articles`" title), define a screen zone including the required text string (for example "`Gestion Articles`") using the mouse. The screen zone is outlined in white:

Figure 2 - 32: Selecting the ArticlesManagementScreen screen class detection criterion on screen

The **New ScreenClass** and **New TagName** icons become active.

2 Click on the **New ScreenClass** icon .

A new screen class is automatically generated in the **Projects View** tree structure. It is called after the text string selected as detection citerion: `Gestion Articles`. The

selected criterion of **Find string** type (see *"Criterion"* on page 1-6) appears in the **Criteria** folder of the newly created screen class:

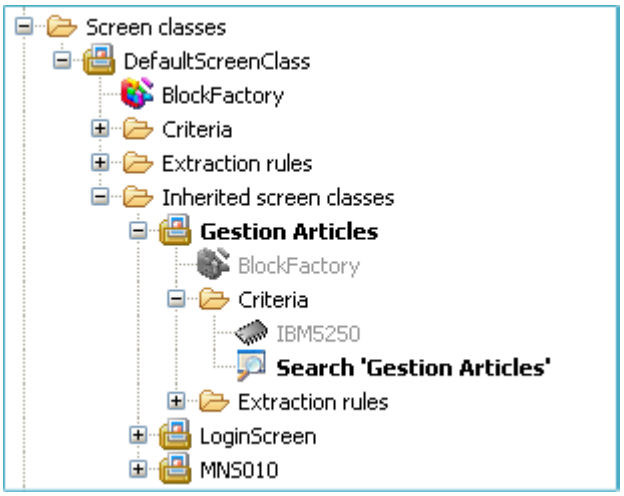


Figure 2 - 33: Newly created screen class (Gestion Articles) and associated detection criteria

The screen class name (`Gestion Articles`) can be renamed to be consistent with the naming policy chosen for the project (either screen identifiers or names ending in "`Screen`").

3 Rename the `Gestion Articles` screen class into `ArticlesManagementScreen` by following the renaming procedure described in *"To rename a screen class"* on page 2 - 14.

The screen class identifying article management screens is now created.

A problem is now emerging: If we pass on to the next screen, we notice that the "`Gestion Articles`" text string also appears in the next screen, which is an article management screen as well:

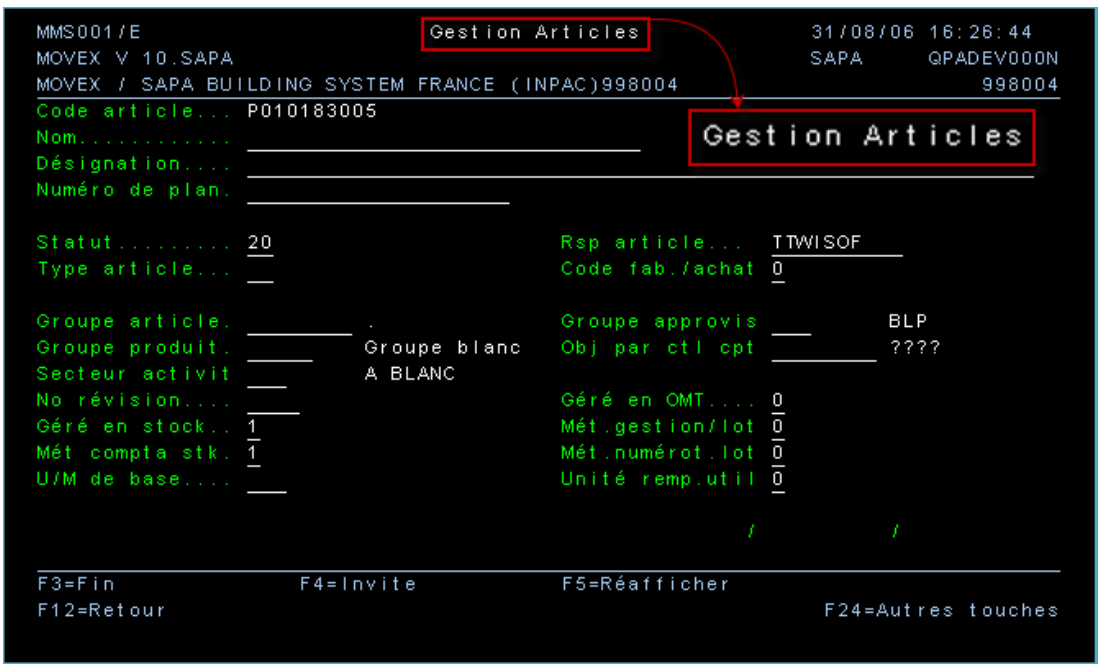


Figure 2 - 34: "Gestion Articles" screen title

This can be confusing for CLI, since both screens (article list screen and article description screen) will be detected as belonging to the same `ArticlesManagementScreen` screen class during screen navigation whereas different actions must be performed on each of them.

To solve this problem, we will create two children screen classes of the `ArticlesManagementScreen` screen class, using the screen identifiers of the third and fourth screen accessed in the project: "`MMS001/B1`" and "`MMSOO1/E`".

The following procedure describes how to create the first child screen class of the `ArticlesManagementScreen` screen class: `MMS001B1`.

To create the MMS001B1 screen class

1 Assuming that the **Emulator** displays the proper screen (in our project, the third accessed screen, i.e. the article list screen including a "`Gestion Articles`" title and identified by the `MMS001/B1` screen identifier), define a screen zone including the required text string (for example "`MMS001/B1`") using the mouse. The screen zone is outlined in white:

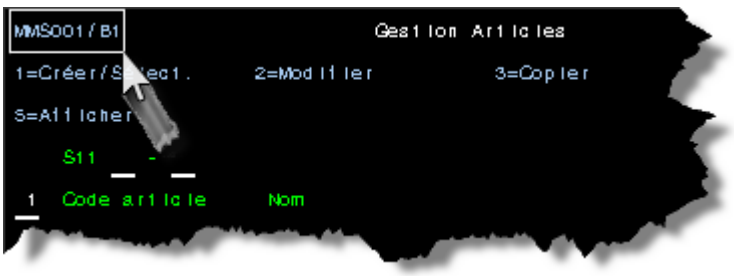


Figure 2 - 35: Selecting the MMS001B1 screen class detection criterion on screen

The **New ScreenClass** and **New TagName** icons become active.

2 Click on the **New ScreenClass** icon .

A new **Inherited screen classes** folder is automatically created under the **ArticlesManagementScreen** screen class in the **Projects View**. The newly created screen class is child to the `ArticlesManagementScreen` screen class.

The new screen class is called after the text string selected as detection citerion: `MMS001/B1`. The selected criterion of **Find string** type (see *"Criterion"* on page 1-6) appears in the **Criteria** folder of the newly created screen class, in addition to inherited parent criteria (greyed):

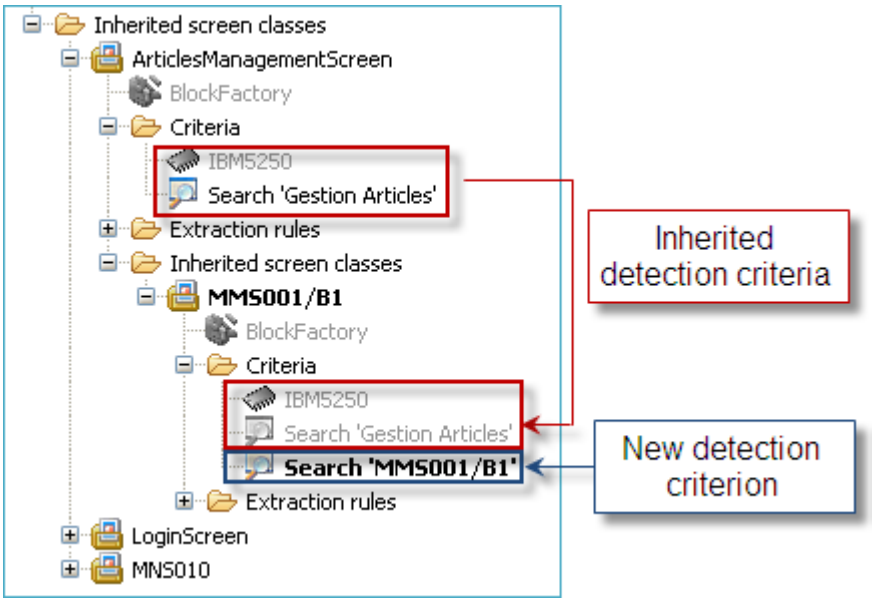


Figure 2 - 36: Newly created screen class (MMS001/B1) and associated detection criteria

It it recommended to avoid special characters in object names. The screen class name (`MMS001/B1`) should therefore be renamed into `MMS001B1` following the renaming procedure described in *"To rename a screen class"* on page 2 - 14.

3 Save the project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

The screen class identifying the third screen (article list), with which a *Table* extraction rule will be associated later in the project, is now created.

After an article code is entered on this page, the corresponding article description screen is accessed. As mentioned before, this screen also contains the "`Gestion Articles`" text string. It will therefore be automatically created as a child screen class of the `ArticlesManagementScreen` screen class.

The following procedure describes how to create the screen class corresponding to the article description screen: `MMS001E`.

To create the MMS001E screen class

1 Assuming that the **Emulator** displays the proper screen (in our project, the fourth accessed screen, i.e. the article description screen including a "`Gestion Articles`" title and identified by the "`MMS001/E`" screen identifier), define a screen zone including the required text string (for example "`MMS001/E`") using the mouse. The screen zone is outlined in white:

Figure 2 - 37: Selecting the MMS001E screen class detection criterion on screen

The **New ScreenClass** and **New TagName** icons become active.

2 Click on the **New ScreenClass** icon .

A new screen class is automatically created in the **Inherited screen classes** folder of the `ArticlesManagementScreen` screen class tree structure.

The new screen class is called after the text string selected as detection citerion: `MMS001/E`. The selected criterion of **Find string** type (see *"Criterion"* on page 1-6) appears in the **Criteria** folder of the newly created screen class, in addition to inherited parent criteria (greyed):

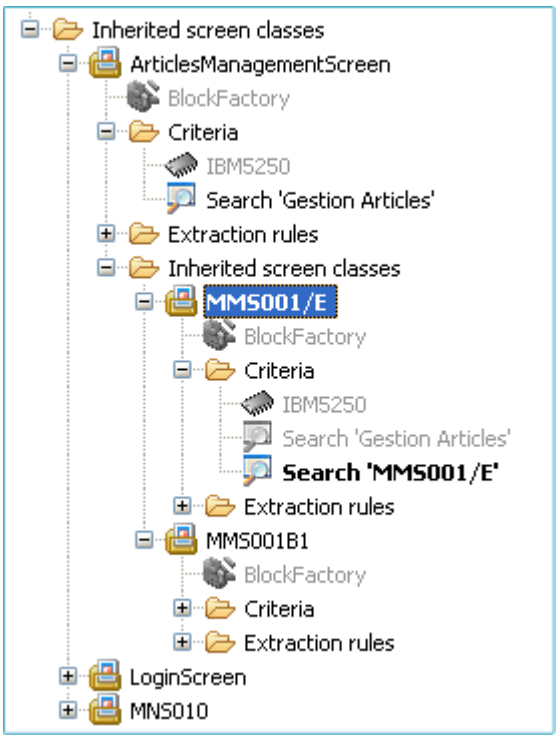


Figure 2 - 38: Newly created screen class (MMS001E) and associated detection criteria

It it recommended to avoid special characters in object names. The screen class name (`MMS001/E`) should therefore be renamed into `MMS001E` following the renaming procedure described in *"To rename a screen class"* on page 2 - 14.

3 Save the project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

All screen classes are now defined and properly set.

WHAT COMES NEXT?

As mentioned in *"Screen Class"* on page 1 - 5, screen classes are:

- *identified* by detection criteria;
- *associated* with extraction rules (if need be).

In the context of our sample project, two types of extraction rules are created: one *Table* and three *XML tag name* extraction rules. They are executed on detection of two of the previously set screen classes: `MMS001B1` and `MMS001E`.

2.2.3 Creating Extraction Rules

The following table describes the extraction rules created in our sample project:

Table 2 - 2: Sample project extraction rules

Extraction rule	Template	Description
`ArticlesTable`	*Table*	Executed on detection of `MMS001B1` screen class (article list). Extracts article list in a three-column (`code article`, `nom`, `stt`) table.
`Tag article_status`	*XML tag name*	Executed on detection of `MMS001E` screen class (article description). Extracts article status.
`Tag article_rsp`	*XML tag name*	Executed on detection of `MMS001E` screen class (article description). Extracts article rsp.
`Tag product_group`	*XML tag name*	Executed on detection of `MMS001E` screen class (article description). Extracts product group.

The following procedures describe how to create and set these four extraction rules.

The first extraction rule to be described is a *Table* extraction rule: `ArticlesTable`. Its purpose is to extract a list of articles into a three-column table.

To create a Table extraction rule

1. In the **Projects View**, right-click on the required screen class (`MMS001B1`).
2. Select ***New > Extraction rule*** in the drop-down menu:

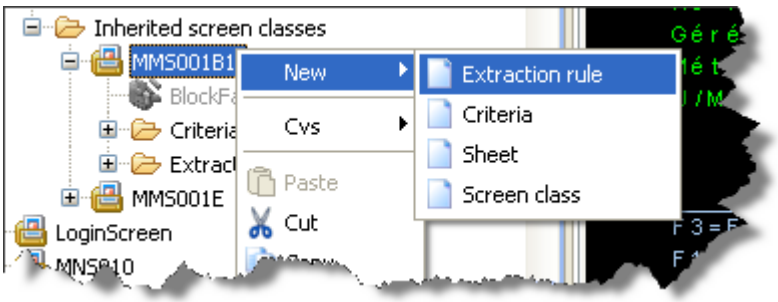


Figure 2 - 39: Selecting new extraction rule

A **New Extraction Rule** wizard opens:

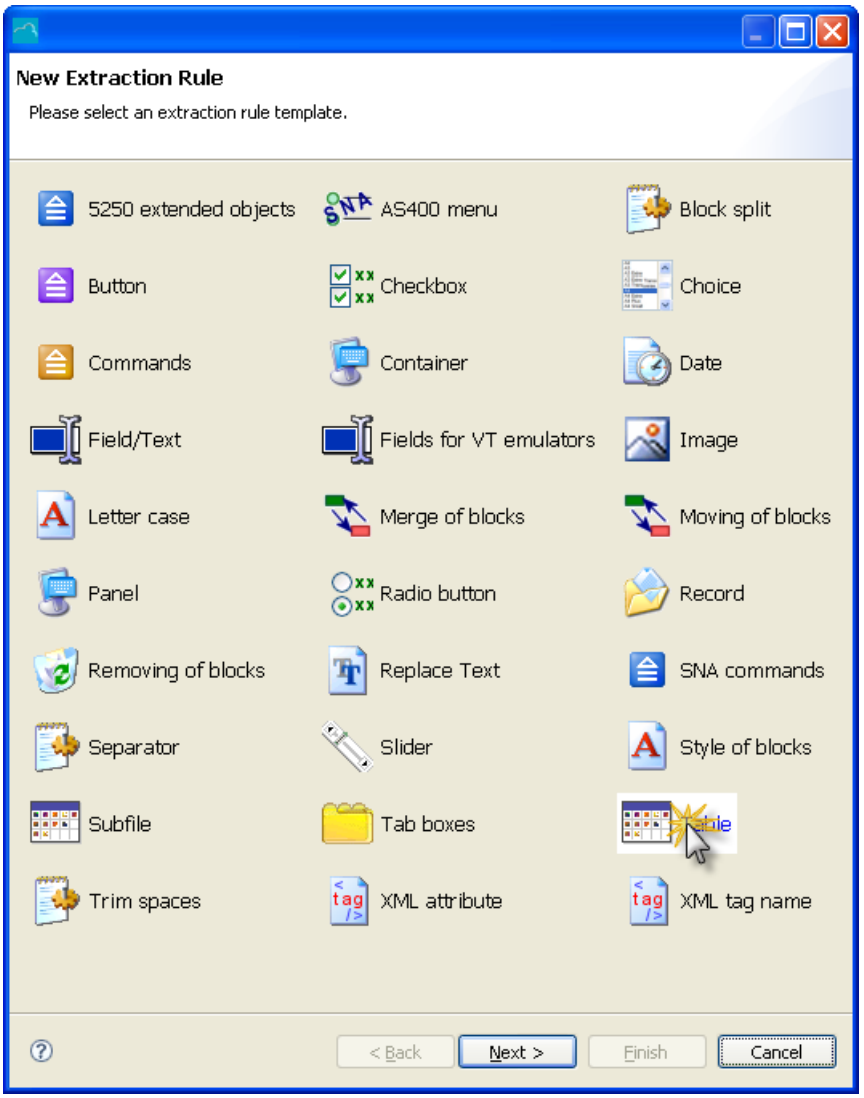


Figure 2 - 40: New Extraction Rule wizard

3 Click on **Table**, which is the template for extracting data into tables.

4 Click on **Next**.

5 In the **Name** field, enter a name (for example `ArticlesTable`):

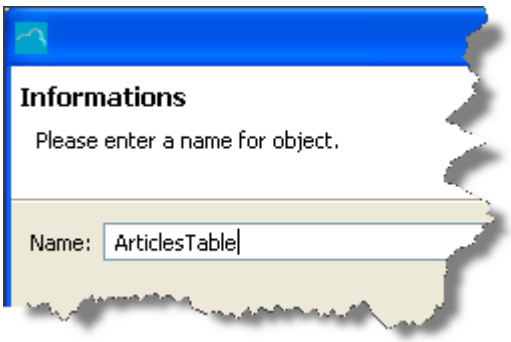


Figure 2 - 41: Giving a name to the new extraction rule

6 Click on **Finish**.

The `ArticlesTable` extraction rule appears in the **Extraction rules** folder of the

screen class in last position after the `DeleteBlocksNamed_block` default extraction rule:

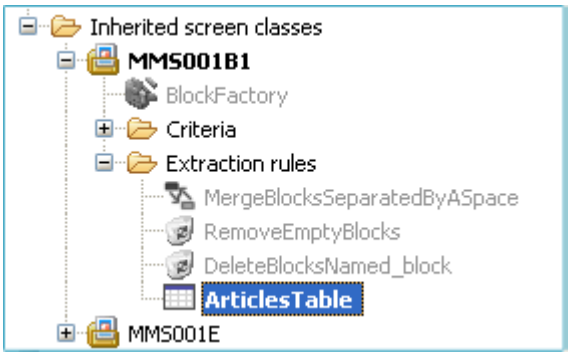


Figure 2 - 42: New extraction rule in Extraction rules folder

As mentioned in *"Extraction rule"* on page 1 - 7, the `DeleteBlocksNamed_block` default extraction rule must always be executed last. To this end, it must always be positioned last in all **Extraction rules** folders.

We will therefore increase the `ArticlesTable` priority by one rank in the folder, to position it before `DeleteBlocksNamed_block` in the extraction rule hierarchy.

7 Select the `ArticlesTable` extraction rule with a left click.

8 In the **Projects View** toolbar, click on the **Increase priority of selected object(s)** icon .

The `ArticlesTable` extraction rule now appears before the `DeleteBlocksNamed_block` default extraction rule:

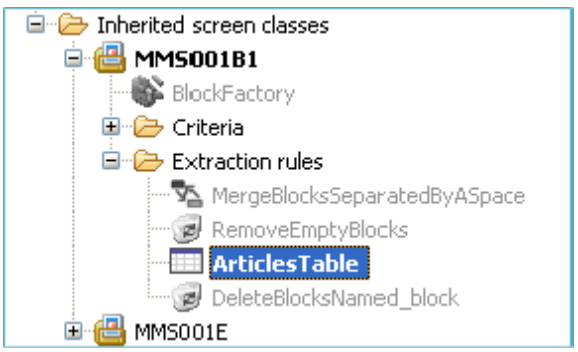


Figure 2 - 43: Increasing the ArticlesTable extraction rule priority by one rank

9 Save the project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

The extraction rule is created and properly positioned in the folder.

We must now:

- set the screen zone of the legacy application screen to which the extraction rule applies, through the setting of the **Screen zone** property;
- set each column name and width, through the setting of the **Columns** property.

Both properties can be automatically and graphically set on screen using the **Link current**

selection to highlighted property in Properties panel icon and the **Add element from current selection to highlighted property in Properties panel** icon .

To set the properties of a Table extraction rule

10 Select the *Table* extraction rule (for example `ArticlesTable`) in the **Projects View**.

The **Properties View** is automatically updated:

Properties

Property	Value
Configuration	
Action separators	=
Action table	[]
Auto Validate	false
Columns	[]
Comment	
Do not accumulate	false
Do not include titles	false
End pattern	
Height resize	0
Index of the selection column	-1
Is active	true
Is final	false
Keep empty lines	false
Offset	0
property.mashupEventName.display_name	
Relative index of the line of actions	0
Remove action lines	true
Separators	
Start pattern	
XML tag name	table
Information	
Depth	n/a
Java class	com.twinsoft.convertigo.beans.common.Table
Name	TableArticles
Priority	1253095598043
QName	/test5250ws3/_data/cn/ASH0HkD/sc/JT1hNjUB/sc/_Wy0d5...
Type	Table
Selection	
Attributes	-1
Screen zone	[x=-1, y=-1, width=-1, height=-1]
Type	

Figure 2 - 44: Table extraction rule properties

11 Click on the **Screen zone** property line. The line is highlighted and a button appears in the **Value** column.

12 In the **Emulator**, delineate with the mouse the screen zone containing the data to be extracted:

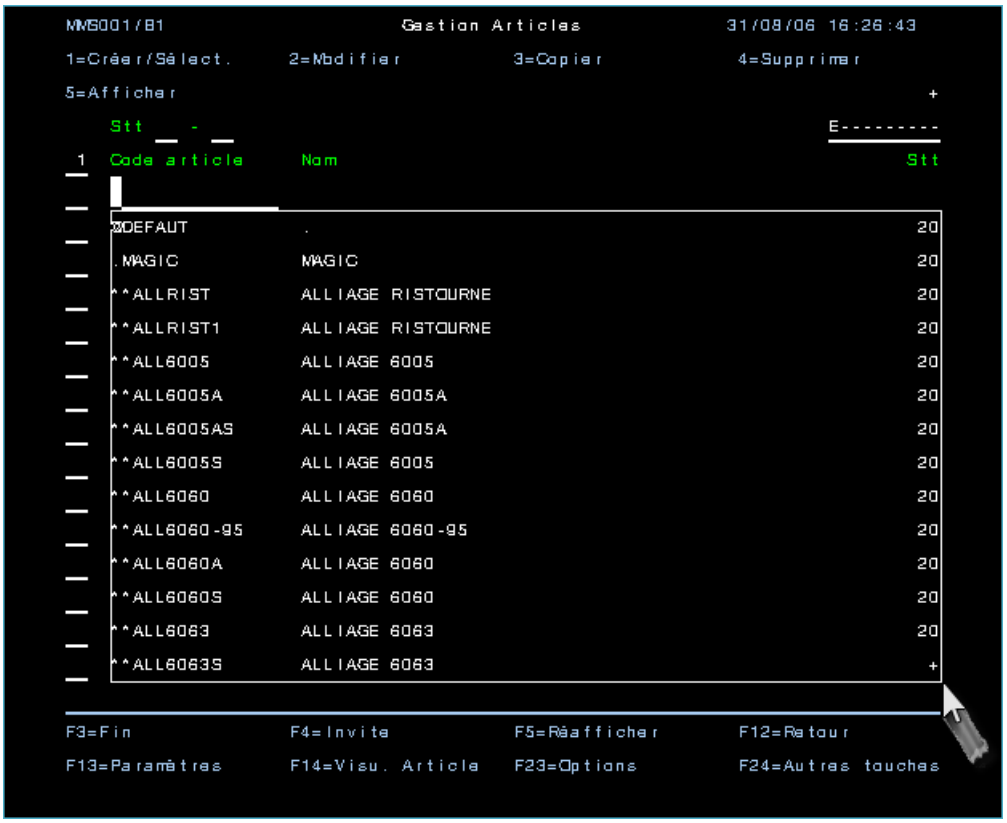


Figure 2 - 45: Delineating the screen zone containing data to be extracted

13 Click on the **Link current selection to highlighted property in Properties panel** icon in the **Emulator** toolbar to link the outlined screen zone with the **Screen zone** property:

Figure 2 - 46: Linking selection to Screen zone property

14 Save your project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

Screen zone values are automatically linked to the highlighted property (**Screen zone**):

Screen zone	[x=5, y=6, width=74, height=14]

Figure 2 - 47: Automatic setting of Screen zone property value

Now that the screen zone containing data to be extracted is set, each table column (`code`, `name` and `status`) must be created and named, and its width set.

15 Click on the **Columns** property line. The line is highlighted and a button appears in the **Value** column.

16 In the **Emulator**, select the first column (`code`) with the mouse, up to the second column (`name`) in the table title line:

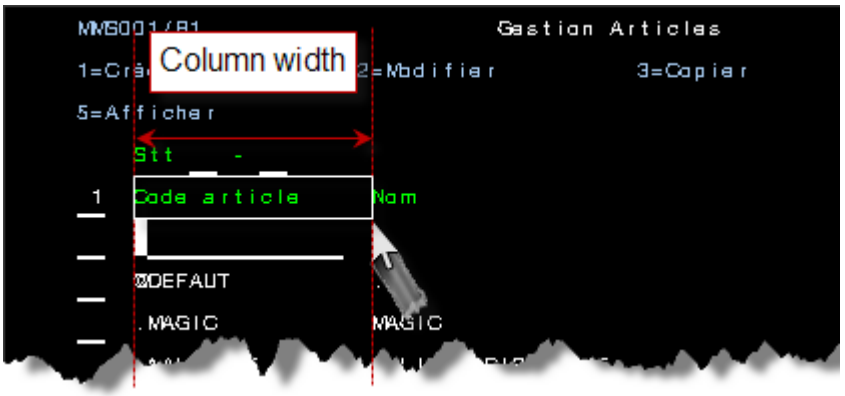


Figure 2 - 48: Setting ArticlesTable extraction rule first column

17 Click on the **Add element from current selection to highlighted property in Properties panel** icon to automatically set the column name and width.

The column is marked on the emulator screen:

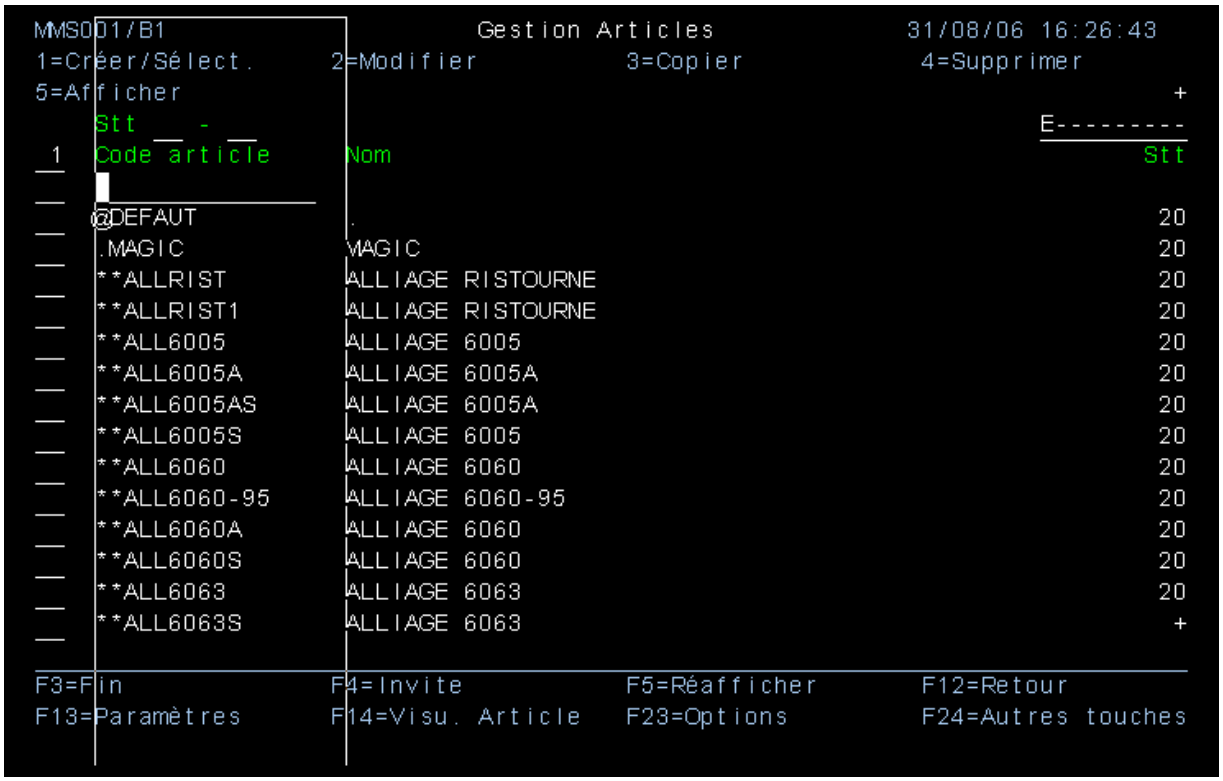


Figure 2 - 49: First column of ArticlesTable extraction rule marked on screen

18 Repeat the last two steps to set the names and widths of second (`name`) and third columns (`status`).

```
MMS001/B1                    Gestion Articles                31/08/06 16:26:43
1=Créer/Sélect.      2=Modifier          3=Copier            4=Supprimer
5=Afficher                                                                +
    Stt __ - __                                              E---------
_1  Code article      Nom                                            Stt
__  ______________
__  @DEFAUT           .                                               20
__  .MAGIC            MAGIC                                           20
__  **ALLRIST         ALLIAGE RISTOURNE                               20
__  **ALLRIST1        ALLIAGE RISTOURNE                               20
__  **ALL6005         ALLIAGE 6005                                    20
__  **ALL6005A        ALLIAGE 6005A                                   20
__  **ALL6005AS       ALLIAGE 6005A                                   20
__  **ALL6005S        ALLIAGE 6005                                    20
__  **ALL6060         ALLIAGE 6060                                    20
__  **ALL6060-95      ALLIAGE 6060-95                                 20
__  **ALL6060A        ALLIAGE 6060                                    20
__  **ALL6060S        ALLIAGE 6060                                    20
__  **ALL6063         ALLIAGE 6063                                    20
__  **ALL6063S        ALLIAGE 6063                                     +
__
F3=Fin               F4=Invite           F5=Réafficher       F12=Retour
F13=Paramètres       F14=Visu. Article   F23=Options         F24=Autres touches
```

Figure 2 - 50: Second column of ArticlesTable extraction rule marked on screen

```
MMS001/B1                    Gestion Articles                31/08/06 16:26:43
1=Créer/Sélect.      2=Modifier          3=Copier            4=Supprimer
5=Afficher                                                                +
    Stt __ - __                                              E---------
_1  Code article      Nom                                            Stt
__  ______________
__  @DEFAUT           .                                               20
__  .MAGIC            MAGIC                                           20
__  **ALLRIST         ALLIAGE RISTOURNE                               20
__  **ALLRIST1        ALLIAGE RISTOURNE                               20
__  **ALL6005         ALLIAGE 6005                                    20
__  **ALL6005A        ALLIAGE 6005A                                   20
__  **ALL6005AS       ALLIAGE 6005A                                   20
__  **ALL6005S        ALLIAGE 6005                                    20
__  **ALL6060         ALLIAGE 6060                                    20
__  **ALL6060-95      ALLIAGE 6060-95                                 20
__  **ALL6060A        ALLIAGE 6060                                    20
__  **ALL6060S        ALLIAGE 6060                                    20
__  **ALL6063         ALLIAGE 6063                                    20
__  **ALL6063S        ALLIAGE 6063                                     +
__
F3=Fin               F4=Invite           F5=Réafficher       F12=Retour
F13=Paramètres       F14=Visu. Article   F23=Options         F24=Autres touches
```

Figure 2 - 51: Third column of ArticlesTable extraction rule marked on screen

19 Save your project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

20 Click again on the **Columns** property line. The line is highlighted and a button

appears.

21 Click on . A window opens:

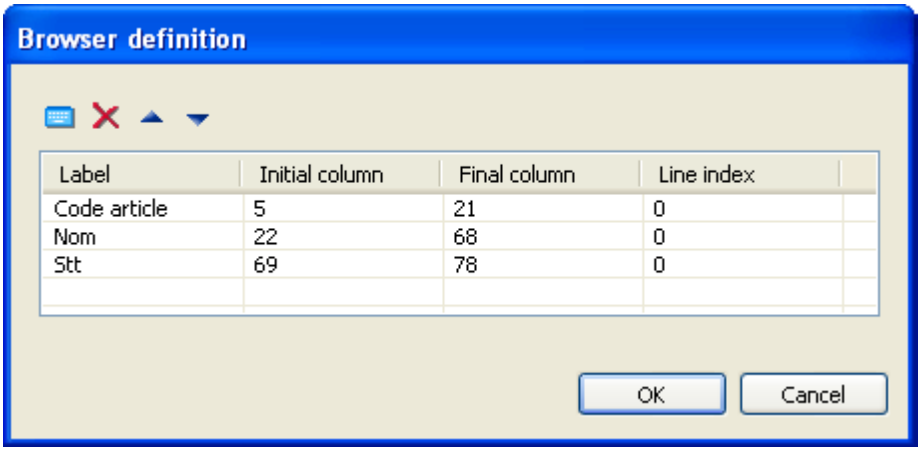


Figure 2 - 52: Column parameters of ArticlesTable extraction rule

This window includes a table showing all column parameters:

Table 2 - 3: Table parameters

Column	Description
Label	Table column XML tag name as it appears in XML output after extraction rule has been executed. The values automatically set have been extracted from the selected screen zones (table title line).
Initial column	Position of column left side in legacy application screen (see Figure 2 - 48).
Final column	Position of column right side in legacy application screen (see Figure 2 - 48).
Line index	Parameter used for multi-lined tables: defines the line number of the column relatively to the first column line (0).

When extracted in XML format (after execution of the extraction rule), each data is given as XML tag name the value of the **Label** column. It is therefore recommended to replace all blank spaces with underscores and to avoid using upper case letters.

We will change:

- "`Code article`" into "`code`";
- "`Nom`" into "`name`";
- "`Stt`" into "`status`".

22 Click in the first line of the **Label** column.

"`Code article`" is highlighted:

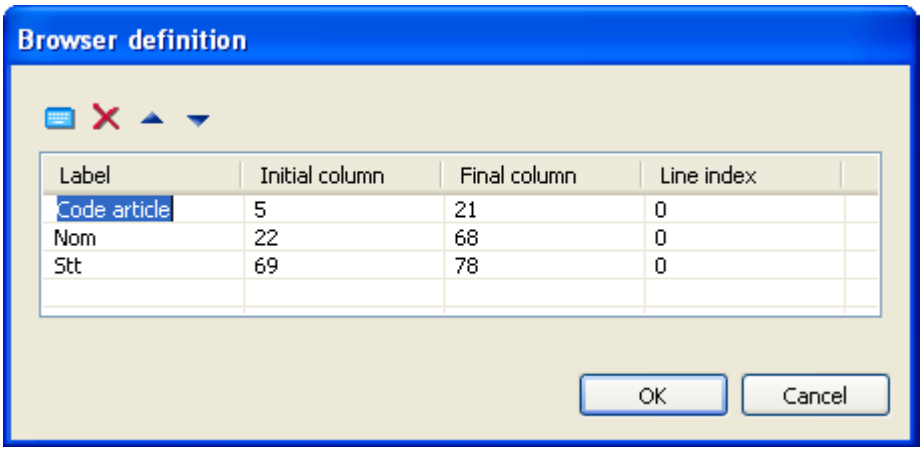


Figure 2 - 53: Changing column labels

23 Type in `code` and press `Enter`.

24 Repeat the last two steps for the other two labels, by changing "`Nom`" into "`name`" and "`Stt`" into "`status`".

25 Click **OK**.

The **Columns** property values are now properly set:

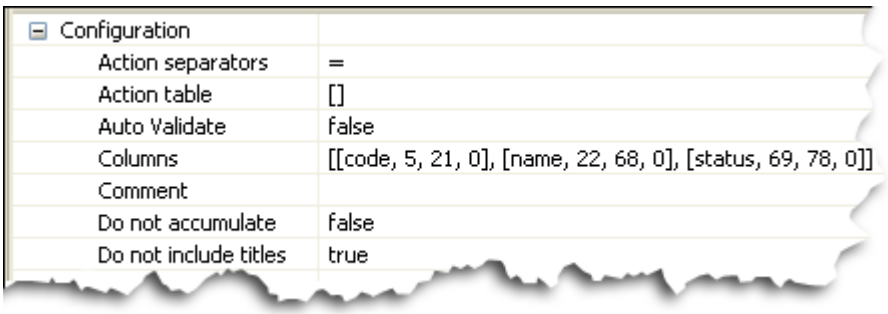


Figure 2 - 54: Columns property values

We will now set two other *Table* extraction rule properties: **Do not include titles** and **XML tag name**. The following table describes these properties:

Table 2 - 4: Table extraction rule properties

Property	Description
Do not include titles	Default value: `false`. When generating the XML table, the first row contains titles. This is useful for webization projects, but not for integration projects. In this case, place the setting to `true`.
XML tag name	Default value: `table`. When generated, the XML table is given the tag name "`table`" by default. You can change this tag name by setting this property value. In our project, the XML table is given the name "`articles`" (see Figure 2 - 57).

26 Click on the value of the **Do not include titles** property. The property is highlighted and a drop-down menu appears.

27 Click on and select `true`:

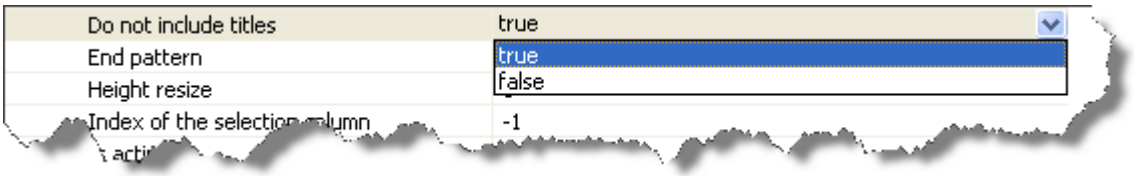


Figure 2 - 55: Setting Do not include titles property to true

28 Press `Enter`.

29 Click on the value of the **XML tag name** property. The value is highlighted:

.

30 Enter a new name (for example `articles`) then press `Enter`.

31 Save your project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

The `ArticlesTable` extraction rule is now properly set. We can execute this extraction rule and see if the XML generation gives expected results.

To check that the extraction rule returns expected results

Assuming that the **Emulator** displays the `MMS001/B1` screen, click on the **Generate XML** icon in the **Connector View** toolbar:

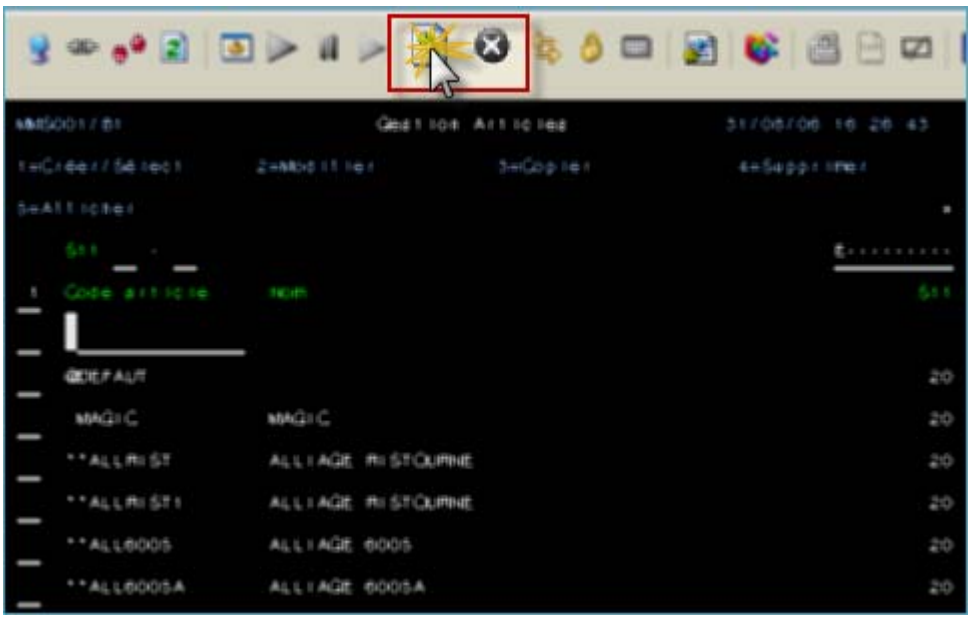


Figure 2 - 56: Generating XML output of MMS001B1 screen

Expected results: The screen is detected as belonging to the `MMS001B1` screen class and data is extracted according to the associated `ArticlesTable` extraction rule:

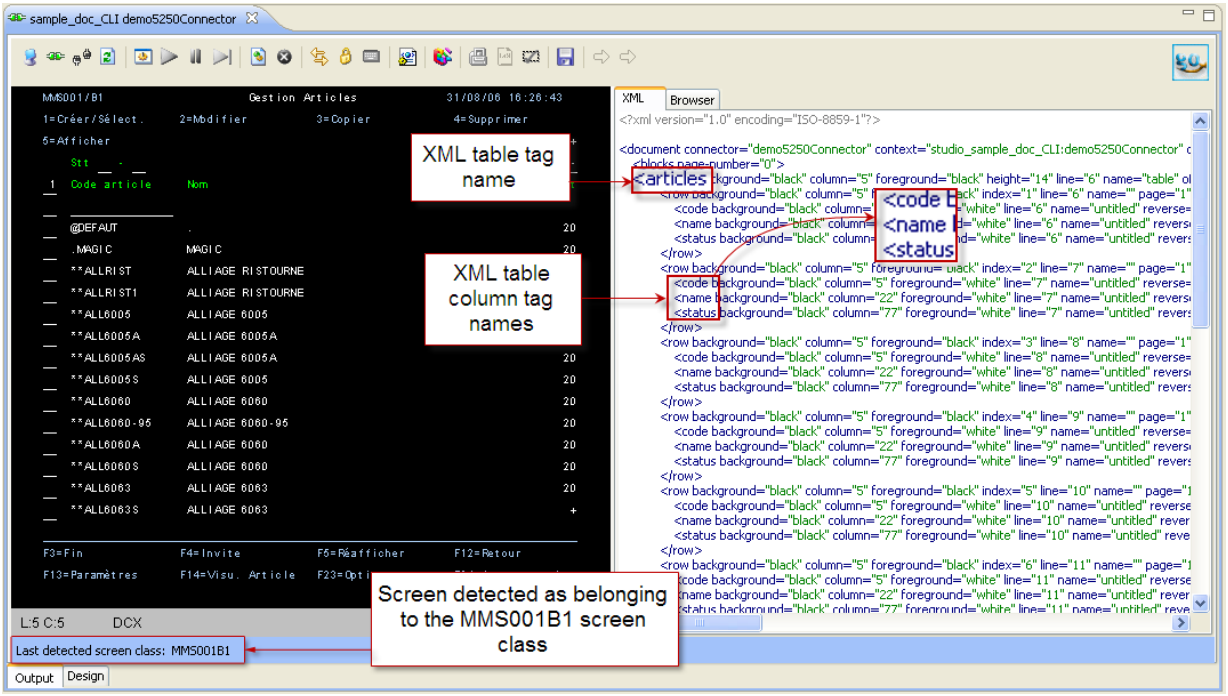


Figure 2 - 57: Generated XML and Last detected screen class

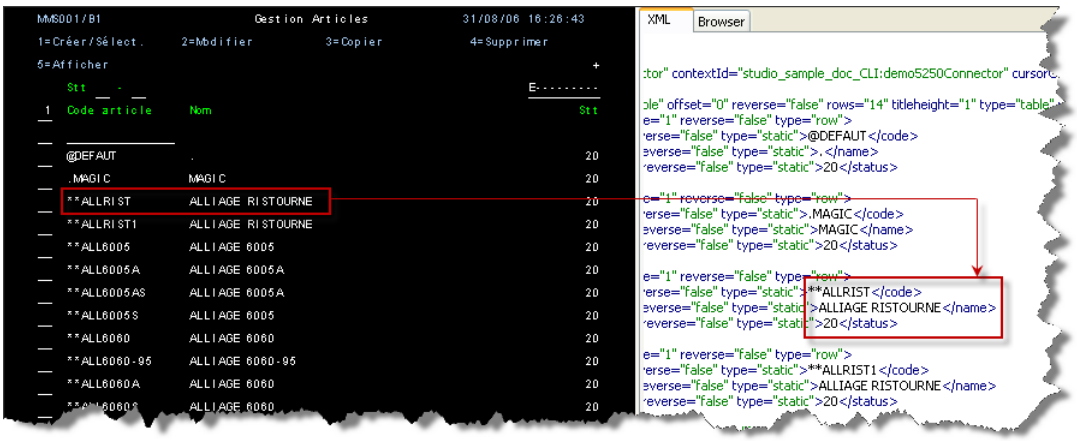


Figure 2 - 58: Data extracted according to ArticlesTable extraction rule properties

The XML generation gives expected results - the test is conclusive.

We will now create three *XML tag name* extraction rules, all associated with the `MMS001E` screen class.

XML tag name extraction rules change the tag names of extracted blocks, tagged by default as "`block`".

The first *XML tag name* extraction rule to be created is called `Tag article_status`. Its purpose is to extract the article status value of a given article from the article description screen.

To create an XML tag name extraction rule

1. Assuming that the **Emulator** displays the proper screen (in our project, the fourth accessed screen, i.e. the article description screen identified by the "`MMS001/E`" screen identifier), define a screen zone including the data to be extracted (for example

"20") using the mouse. The screen zone is outlined in white:

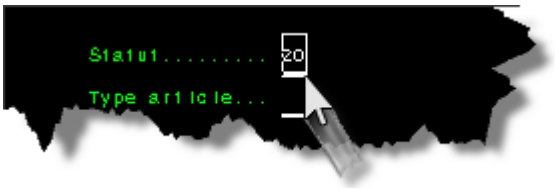


Figure 2 - 59: Selecting data to be extracted on screen

The **New ScreenClass** and **New TagName** icons become active.

2 Click on the **New TagName** icon .

A **New TagName** window opens:

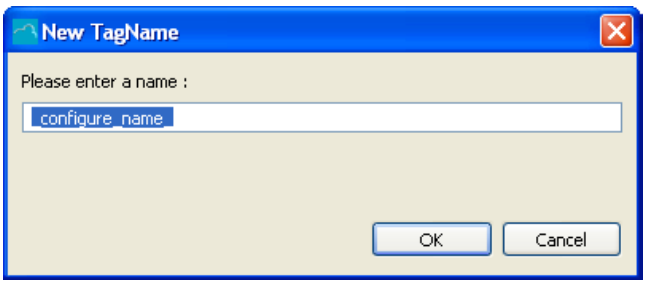


Figure 2 - 60: New TagName window

3 Type an XML tag name representing accurately the data to be extracted (for example `article_status`).

4 Click on **OK**.

The new XML tag name extraction rule is automatically created in the **Extraction rules** folder of the corresponding screen class:

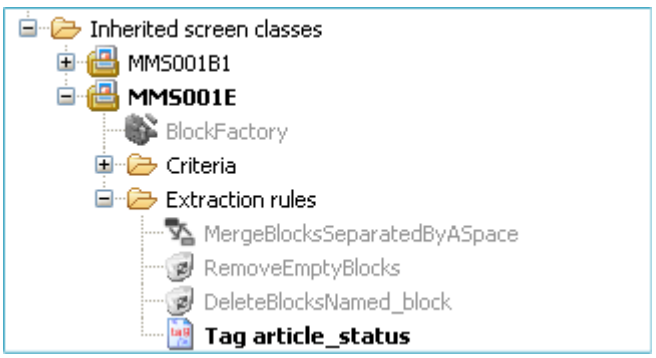


Figure 2 - 61: New extraction rule (Tag article_status) in Extraction rules folder

As mentioned when creating the *Table* extraction rule, the `DeleteBlocksNamed_block` extraction rule must always be positioned last in the folder, so we must increase the priority of the `Tag article_status` extraction rule by one rank.

5 Select the `Tag article_status` extraction rule with a left click.

6 In the **Projects View** toolbar, click on the **Increase priority of selected object(s)** icon .

The `Tag article_status` extraction rule now appears above the `DeleteBlocksNamed_block` default extraction rule:

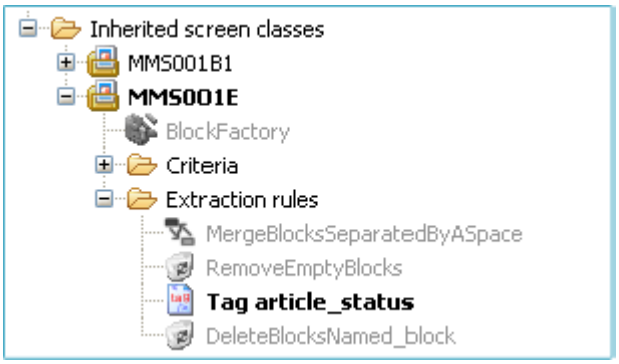


Figure 2 - 62: Increasing the Tag article_status extraction rule priority by one rank

The extraction rule is created and properly positioned in the folder.

7 Save your project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

We can now execute this extraction rule and see if the XML generation gives expected results.

To check that the extraction rule returns expected results

Assuming that the **Emulator** displays the `MMS001/E` screen, click on the **Generate XML** icon in the **Connector View** toolbar.

Expected results: The screen is detected as belonging to the `MMS001E` screen class and the article status ("20") is extracted according to the associated `Tag article_status` extraction rule:

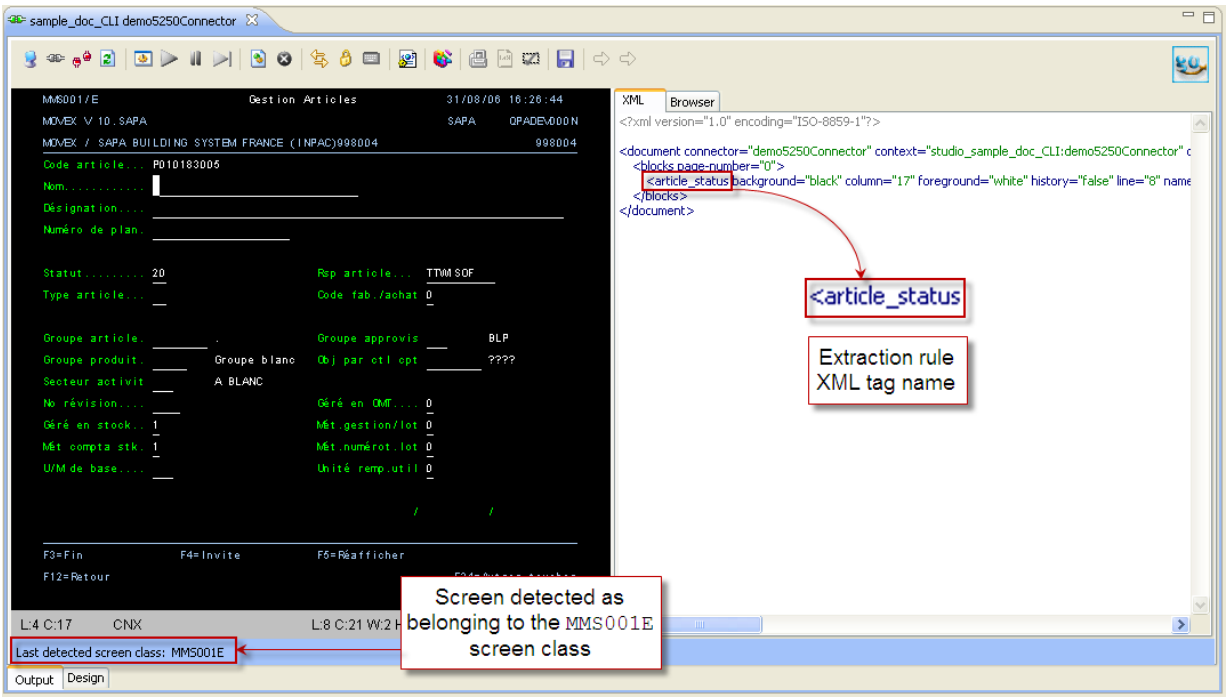


Figure 2 - 63: Generated XML and Last detected screen class

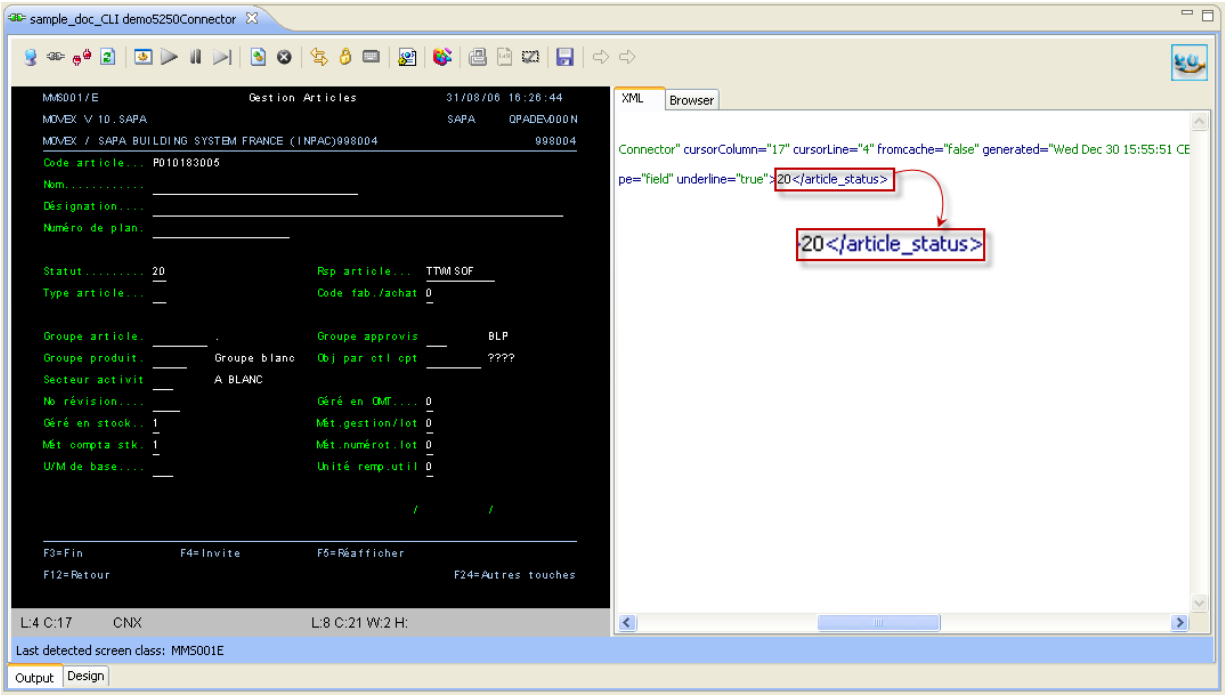


Figure 2 - 64: Data extracted according to Tag article_status extraction rule properties

The XML generation gives expected results - the test is conclusive.

We will now create the other two *XML tag name* extraction rules (`Tag article_rsp` and `Tag product_group`).

The following two procedures, which are identical to the procedure for creating the first *XML tag name* extraction rule, describe how to create the two remaining extraction rules.

To create the Tag article_rsp extraction rule

1. Assuming that the **Emulator** displays the proper screen (in our project, the fourth accessed screen, i.e. the article description screen identified by the "`MMS001/E`" screen identifier), define a screen zone including the data to be extracted (for example "`TTWISOF`") using the mouse. The screen zone is outlined in white:

Figure 2 - 65: Selecting data to be extracted on screen

> ***Although the current text doesn't fill the entire field, the white selected area should cover the whole field in order to be able to extract any data present in this field.***

The **New ScreenClass** and **New TagName** icons become active.

2. Click on the **New TagName** icon .

A **New TagName** window opens:

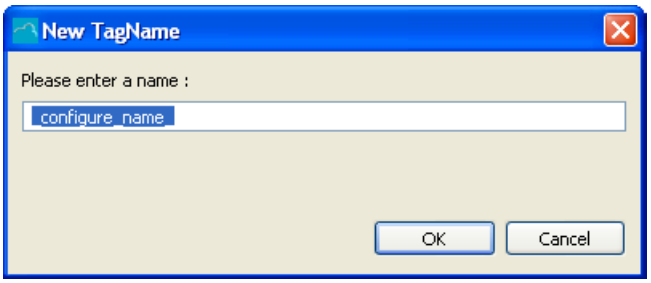


Figure 2 - 66: New TagName window

3 Type an XML tag name representing accurately the data to be extracted (for example `article_rsp`).

4 Click on **OK**.

The new XML tag name extraction rule is automatically created in the **Extraction rules** folder of the corresponding screen class:

Figure 2 - 67: New extraction rule (Tag article_rsp) in Extraction rules folder

As mentioned when creating the *Table* extraction rule, the `DeleteBlocksNamed_block` extraction rule must always be positioned last in the folder, so we must increase the priority of the `Tag article_rsp` extraction rule by one rank.

5 Select the `Tag article_rsp` extraction rule with a left click.

6 In the **Projects View** toolbar, click on the **Increase priority of selected object(s)** icon .

The `Tag article_rsp` extraction rule now appears before the `DeleteBlocksNamed_block` default extraction rule:

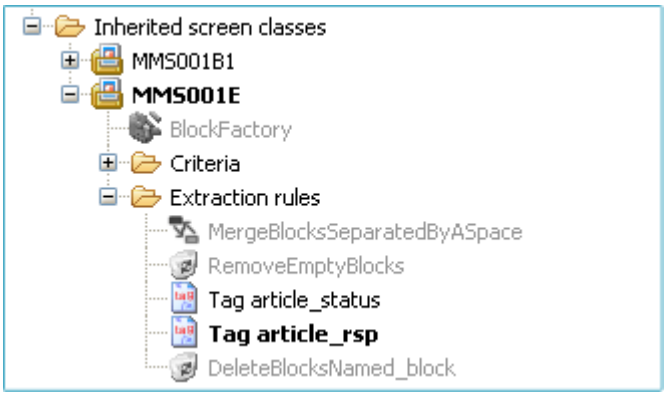


Figure 2 - 68: Increasing the Tag article_rsp extraction rule priority by one rank

The extraction rule is created and properly positioned in the folder.

7 Save your project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

We can now execute this extraction rule and see if the XML generation gives expected results. To do so, follow the instructions of *"To check that the extraction rule returns expected results"* on page 2 - 34.

To create the Tag product_group extraction rule

1 Assuming that the **Emulator** displays the proper screen (in our project, the fourth accessed screen, i.e. the article description screen identified by the "`MMS001/E`" screen identifier), define a screen zone including the data to be extracted (for example "`Groupe blanc`") using the mouse. The screen zone is outlined in white:

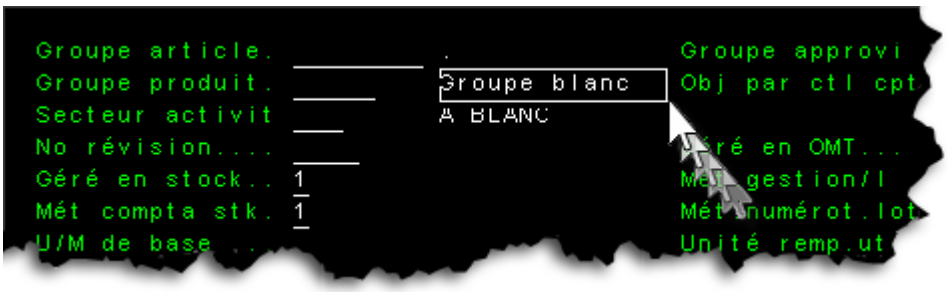


Figure 2 - 69: Selecting data to be extracted on screen

The **New ScreenClass** and **New TagName** icons become active.

2 Click on the **New TagName** icon .

A **New TagName** window opens:

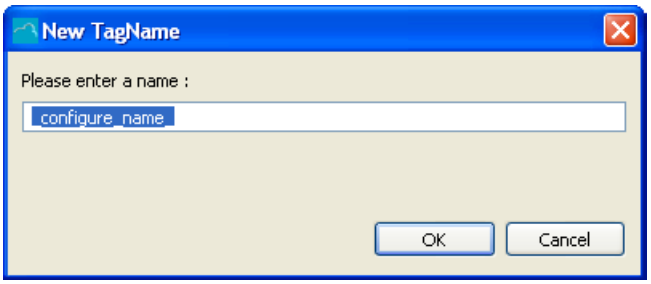


Figure 2 - 70: New TagName window

3 Type an XML tag name representing accurately the data to be extracted (for example `product_group`).

4 Click on **OK**.

The new XML tag name extraction rule is automatically created in the **Extraction rules** folder of the corresponding screen class:

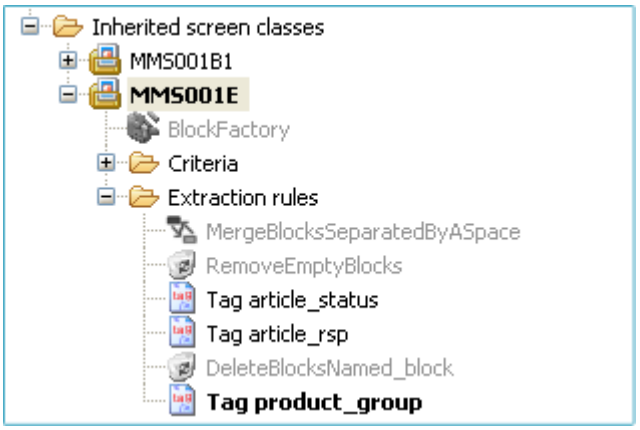


Figure 2 - 71: New extraction rule (Tag product_group) in Extraction rules folder

As mentioned when creating the *Table* extraction rule, the `DeleteBlocksNamed_block` extraction rule must always be positioned last in the folder, so we must increase the priority of the `Tag product_group` extraction rule by one rank.

5 Select the `Tag product_group` extraction rule with a left click.

6 In the **Projects View** toolbar, click on the **Increase priority of selected object(s)** icon .

The `Tag product_group` extraction rule now appears before the `DeleteBlocksNamed_block` default extraction rule:

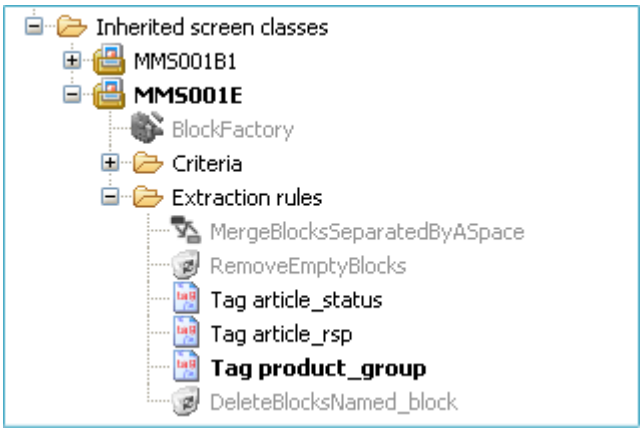


Figure 2 - 72: Increasing the Tag product_group extraction rule priority by one rank

The extraction rule is created and properly positioned in the folder.

7 Save your project by clicking on the **Save** icon or by pressing the `Ctrl+S` key combination.

We can now execute this extraction rule and see if the XML generation gives expected results. To do so, follow the instructions of *"To check that the extraction rule returns expected results"* on page 2 - 34.

WHAT COMES NEXT?

All screen classes and associated extraction rules are now created and properly set.

The next step consists in defining a transaction automating the navigation process described in *"Legacy Integrating Project"* on page 2 - 1.

For the automation to be possible, screen class handlers must be defined for each accessed screen (and detected screen class). Screen class handlers are triggered on detection of screen classes, and perform actions such as "move cursor", "input data", "press Enter", etc. in accessed and detected screens of the emulated application.

The following section shows how to create a transaction, screen class handlers and how to set these objects.

2.2.4 Defining a Transaction

A transaction can be seen as an intelligent script performing automated actions on the screens of the emulated application. Automated actions are performed by screen class handlers (see *"Screen class handler"* on page 1-11) when screens are accessed and detected as belonging to given user-defined screen classes.

As many screen class handlers as screens needing actions to be performed must therefore be defined. In our project, three screen class handlers are needed, corresponding to the LoginScreen, `MNS010` and `MMS001B1` screen classes (see Table 2 - 5 on page 2-40). These screen classes define the screens on which we need to automate actions.

The following table describes screen class handlers and actions to be performed on each detected screen class:

Table 2 - 5: Transaction actions

Screen class	Screen class handler type	Set of actions
`LoginScreen`	Entry screen class handler	1 Enter username "`TOTO`". 2 Position cursor in "`mot de passe`" field. 3 Enter password "`TITI`". 4 Press Enter.
`MNS010`	Entry screen class handler	1 Position cursor in "`Option ou commande`" field; 2 Enter command ("`110`"); 3 Press Enter.
`MMS001B1`	Exit screen class handler	**Extract article list**, then: 1 Position cursor in "`Code article`" field; 2 Enter article code ("`123456789`"); 3 Press Enter.
`MMS001E`	-	No action needed, only extract article data.

For more infomation on transactions, see "Transaction" on page 1-10.

The following procedures describe how to create a transaction and set screen class handlers.

To create a transaction

1. In the **Projects View**, right-click on the connector (for example `demo5250Connector`) or on the **Transactions** folder.
2. Select ***New > Transaction***:

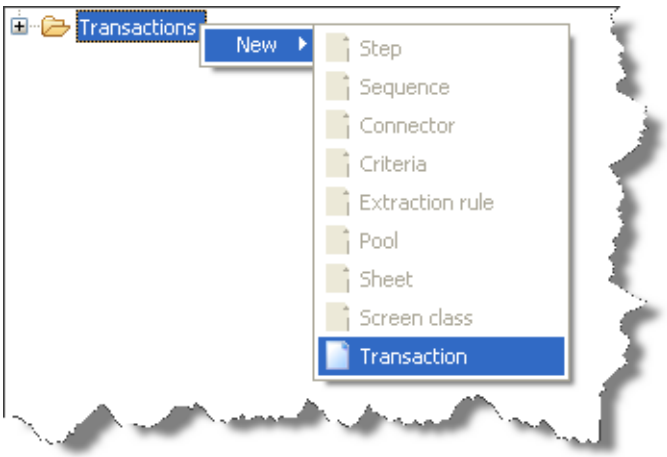


Figure 2 - 73: Creating new transaction

A **New Transaction** wizard opens.

3. Select **Screen transaction**:

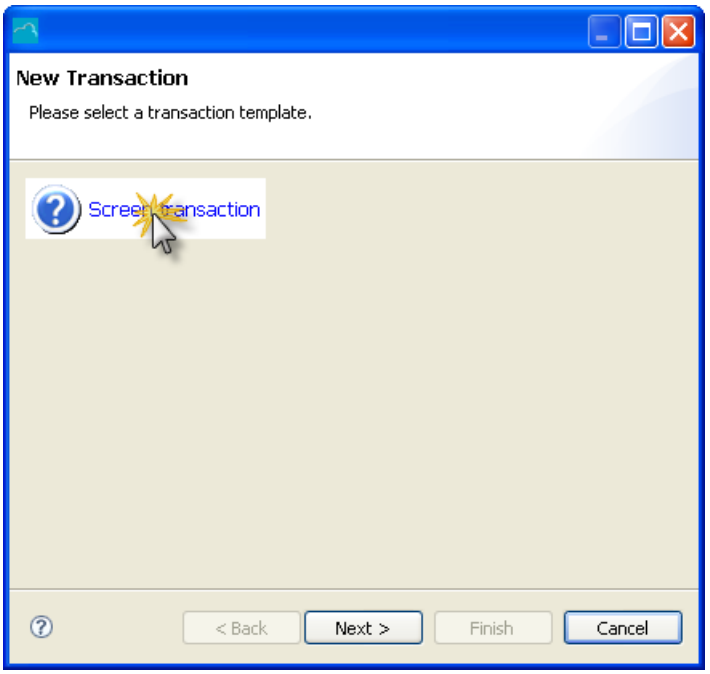


Figure 2 - 74: New Transaction wizard

4 Click on **Next**.

5 In the **Name** field, enter the transaction name (for example `GetArticleData`):

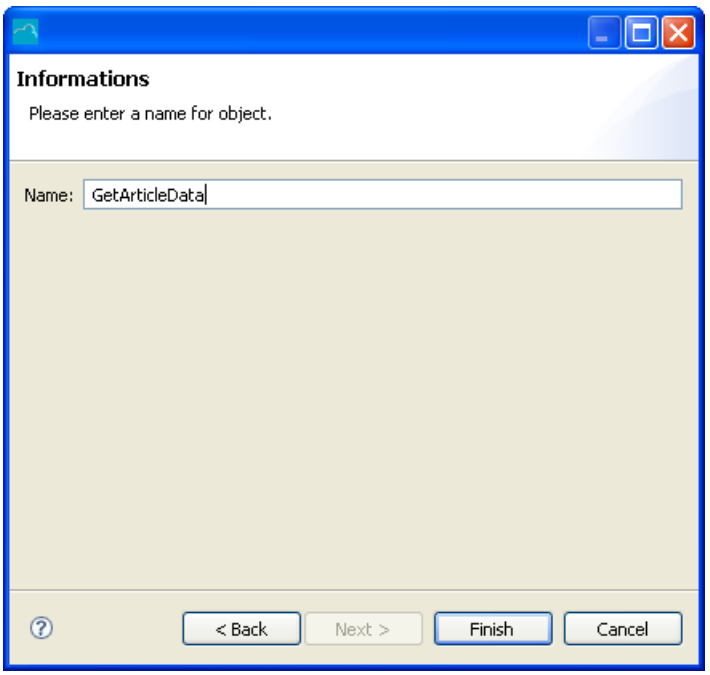


Figure 2 - 75: Entering the transaction name

6 Click on **Finish**.

The transaction is automatically created in the **Transactions** folder of the connector, in addition to the `DefaultTransaction`:

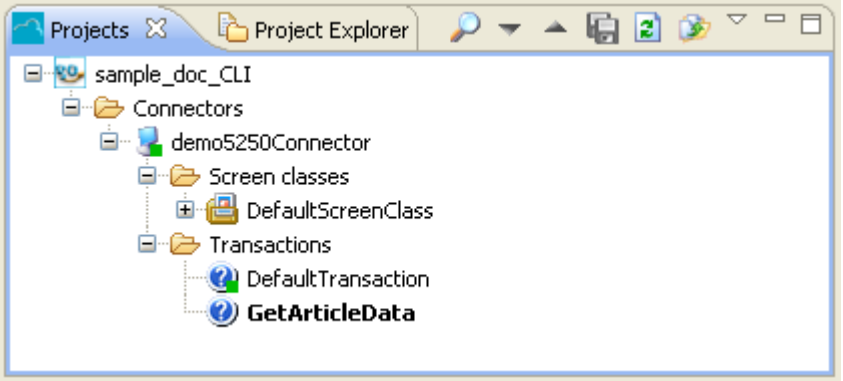


Figure 2 - 76: New transaction (GetArticleData) in Transactions folder

The transaction is now created.

So far the transaction is empty. It only operates the connection with the legacy application according to connector parameters set in *"Creating a Project and Setting Connector Parameters"* on page 2 - 6, but no screen class handler is defined yet.

7 Right-click on the transaction and select **Edit handlers**:

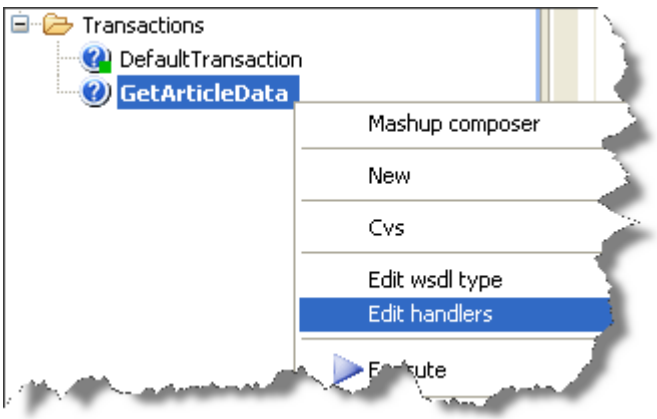


Figure 2 - 77: Edit handlers command

A `GetArticleData` handlers **Editor** opens in a tab above the **Connector View** tab:

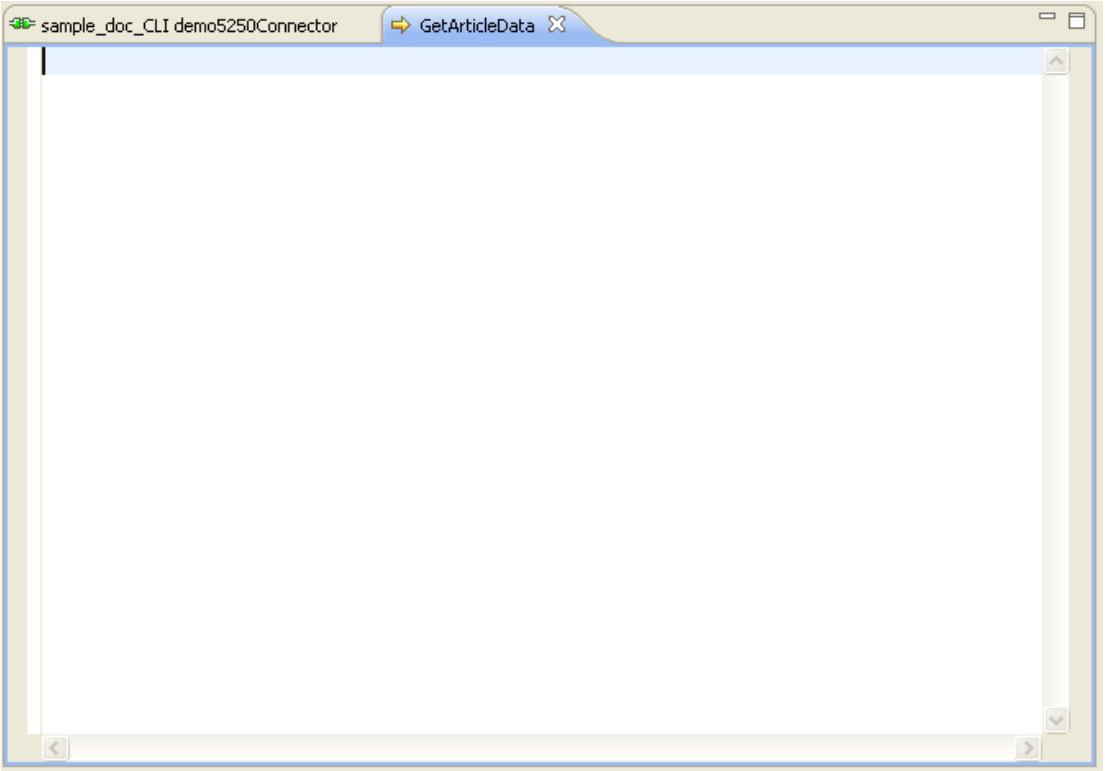


Figure 2 - 78: Transaction handlers editor

The next step consists in filling this transaction document with screen class handler definitions. As mentioned in *"Screen class handler"* on page 1 - 11, screen class handlers are defined as Javascript functions containing Javascripts method calls.

Creating a screen class handler means adding a Javascript function and editing it in the transaction handlers **Editor** by creating as many methods as actions needed on a screen.

The first accessed screen being the login screen (detected by CLI as belonging to the `LoginScreen` screen class), the first screen class handler to be created is: `onLoginScreenEntry`.

To create a screen class handler

1. In the **Projects View**, right-click on the transaction (for example `GetArticleData`) and select ***New > Handler***:

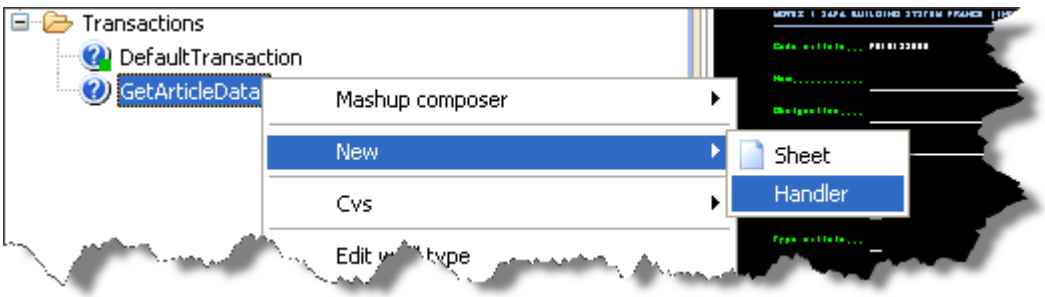


Figure 2 - 79: Creating new handler

A **New transaction handler** window opens:

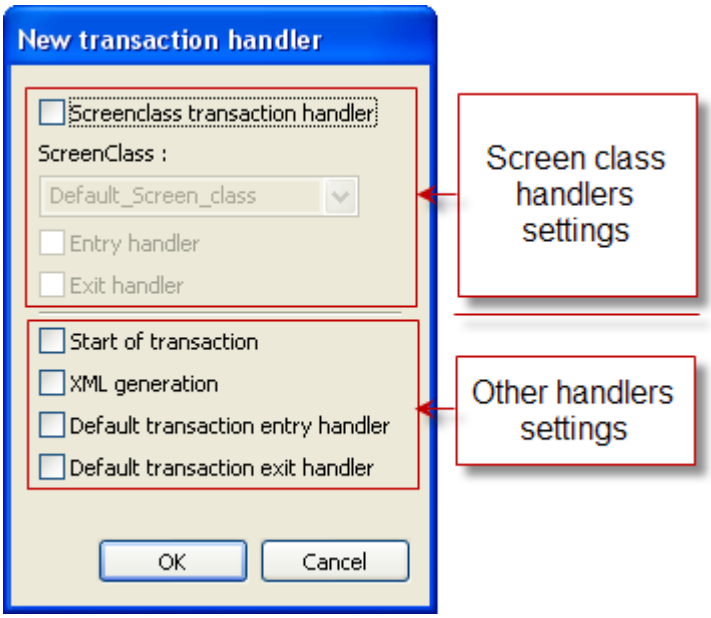


Figure 2 - 80: New transaction handler window

This window is divided into:

- a first part dedicated to *screen class* handlers (with three checkboxes and a drop-down menu);
- a second part dedicated to other types of handlers (with four checkboxes).

The table below describes these elements:

Table 2 - 6: New transaction handler window description

Window Element		Description
Screen class handler settings *Checkboxes*	Screen class transaction handler	Check this box if you want to create a screen class handler (as opposed to other available handlers) that wil be triggered on screen class detection.
	Entry handler	Check this box if screen class handler must be invoked when accessing the screen class. An entry screen class handler handles the following actions: access screen, execute statements on the screen, exit to next screen. Available only if **Screen class transaction handler** checkbox is checked.
	Exit handler	Check this box if the screen class handler must be invoked when exiting the screen class. An exit screen class handler handles the following actions: access screen, extract data according to extraction rules, execute statements on the screen, exit to next screen. Available only if **Screen class transaction handler** checkbox is checked.
Drop-down menu	Screen class	Displays the list of screen classes available in the project. Available only if **Screen class transaction handler** checkbox is checked.
Other handlers settings *Checkboxes*	Start of transaction	Check this box if you want to create a handler that will be triggered when starting the transaction.
	XML generation	Check this box if you want to create a handler that will be triggered at the end of the transaction (once the XML output has been generated).

Table 2 - 6: New transaction handler window description (...)

Window Element		Description
	Default transaction entry handler	Check this box to create a default transaction entry handler.
	Default transaction exit handler	Check this box to create a default transaction exit handler.

The screen class handler we are about to create is triggered on detection of a screen class (`LoginScreen`), so its parameters are the following:

- this is a *screen class transaction handler*, meaning that all the screen class handler settings of the **New transaction handler** window (see Figure 2 - 80) must be set;
- this is an ***entry*** handler, because no data has to be extracted from the login screen;
- it is triggered on detection of the `LoginScreen` screen class.

We will now set the scren class handler parameters:

2 Check the **Screen class transaction handler** checkbox.

3 In the **ScreenClass** drop-down menu, select `LoginScreen`.

4 Check the **Entry handler** checkbox.

5 Click on **OK**.

The `onLoginScreenEntry()` screen class handler appears in the **Functions** folder of the `GetArticleData` transaction of the **Projects View**:

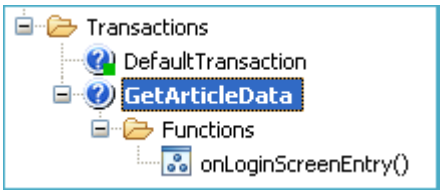


Figure 2 - 81: onLoginScreenEntry() screen class handler

In parallel, the transaction document is filled with a JavaScript function containing only comments. This function, called `onLoginScreenEntry()` corresponds to the screen class handler which will be triggered on detection of the `LoginScreen` screen class:

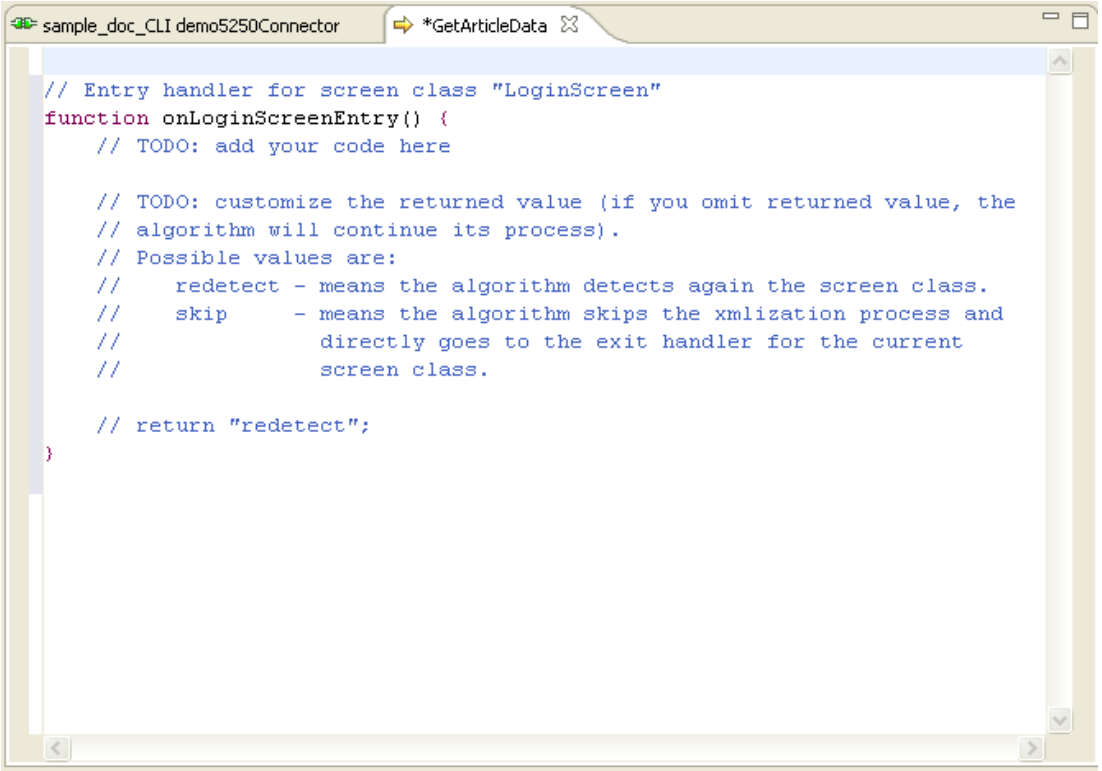


Figure 2 - 82: onLoginScreenEntry() screen class handler definition in Editor

An asterisk to the left of the tab name indicates that the document is being edited and has not been saved yet:

Figure 2 - 83: Document under edition

We will now edit the `onLoginScreenEntry()` screen class handler in the **Editor**, by writing as many JavaScript statements as actions needed in the login screen.

To edit a screen class handler

As mentioned in *"Screen class handler"* on page 1 - 11, screen class handlers are JavaScript functions containing a set of JavaScript statements.

In CLI, the object representing the emulator is the ***javelin*** object. JavaScript statements are methods applied on the *javelin* object. They must comply with the following format:

`javelin.`*method(param1, param2, param3...)*`;`

When accessing the login screen, the cursor is already positioned in the username field ("`Entrez votre nom`"). The actions to be performed in the **Emulator** screen are therefore the following:

1 *Enter* username `"TOTO"`.

2 *Move cursor* to password field.

3 *Enter* password `"TITI"`.

4 *Press* `Enter`.

5 Wait for the screen to be stable.

Each of these actions correspond to a specific JavaScript method. The following table relates screen actions to javelin object methods for the `onLoginScreenEntry()` screen class handler:

Table 2 - 7: LoginScreen screen class - Actions to be performed and methods

Action	Method signature	Javascript statement
Enter username	`send("`*TEXTSTRING*`")`	`javelin.send("TOTO");`
Move cursor	`moveCursor(`*column*`, `*line*`)`	`javelin.moveCursor(column, line);`
Enter password	`send("`*TEXTSTRING*`")`	`javelin.send("TITI");`
Press `Enter`	`doAction(`*action*`)`	`javelin.doAction("KEY_ENTER");`
Wait for screen to be stable	`waitForDataStable(timeout, threshold)`	`javelin.waitForDataStable(timeout, threshold);`

For more information on Javascript methods for the javelin object, see the complete Javelin Object Reference here.

When all actions have been performed on a screen, they made the emaulator go to another screen. CLI must redetect the corresponding screen class. If a screen class handler exists for the detected screen class, the matching screen class handler is triggered, and so on.

To this end, the `redetect` return value is added to the screen class handler:

```
return "redetect";
```

We will now edit the `onLoginScreenEntry()` JavaScript function by including the previously described statements in the JavaScript function.

6 In the `GetArticleData` tab, position your cursor below comments:

```
// Entry handler for screen class "LoginScreen"
function onLoginScreenEntry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, th
    // algorithm will continue its process).
    // Possible values are:
    //    redetect - means the algorithm detects again the screen class.
    //    skip     - means the algorithm skips the xmlization process and
    //               directly goes to the exit handler for the current
    //               screen class.

    // return "redetect";
}
```

Figure 2 - 84: Positioning cursor below comments

7 Type the first JavaScript statement of the `onLoginScreenEntry()` screen class

handler: `javelin.send("TOTO");`

Remember to always end JavaScript statements with a semi-colon.

When typing the statement, pressing the `Crtl+Space` key combination after "`javelin.`" opens a helping list of available methods in a window:

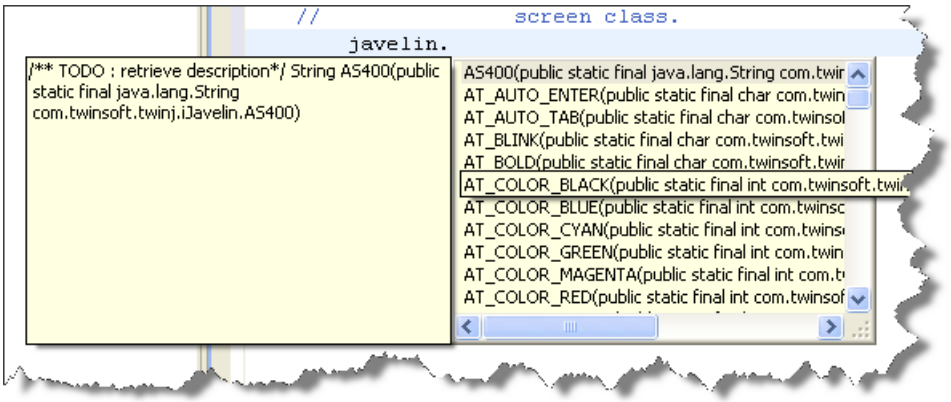


Figure 2 - 85: Javascript method list

You can then select the required method in the list:

Figure 2 - 86: Selecting "send" method in list

The method is automatically inserted with required parameters in the screen class handler:

```
javelin.send(String keystrokes)
```

Figure 2 - 87: Automatic insertion of method

You only need to replace the expected parameter names by the actual values ("TOTO" for this statement) and end the line by a semi-colon.

8 Press `Enter`:

```
javelin.send("TOTO");

// return "redetect";
```

Figure 2 - 88: First JavaScript statement of onLoginScreenEntry() screen class handler

The first JavaScript statement of the `onLoginScreenEntry()` screen class handler is now written.

For the second statement ("move cursor"), we need to know the target position (column, line) of the cursor.

9 Click on the **Emulator** tab.

10 Assuming that the **Emulator** displays the login screen and that the cursor flashes in the "`Entrez votre nom`" field, press the tabulation key.

The cursor now flashes in the password field. Note the cursor coordinates appearing in the lower left corner of the **Connector View**.

Figure 2 - 89: Cursor position "move cursor" statement of onLoginScreenEntry screen class handler

The cursor must be moved to column "60" and line "15", the method is therefore `moveCursor(60, 15)`.

11 Type the second JavaScript statement of the `onLoginScreenEntry()` screen class handler: `javelin.moveCursor(60, 15);`

Please note that the position is given as line (L) and column (C) in the Connector View, while it must be entered as (column, line) in the `moveCursor` method.

12 Press `Enter`:

```
        javelin.send("TOTO");
        javelin.moveCursor(60, 15);


    // return "redetect";
}
```

Figure 2 - 90: Second JavaScript statement of onLoginScreenEntry() screen class handler

The second JavaScript statement of the `onLoginScreenEntry()` screen class handler is now written.

The third statement is similar to the first, only the text string to be sent changes.

13 Copy the first statement.

14 Paste it below the second statement.

15 Replace `TOTO` with `TITI`.

16 Press `Enter`:

```
        javelin.send("TOTO");
        javelin.moveCursor(60, 15);
        javelin.send("TITI");


    // return "redetect";
}
```

Figure 2 - 91: Third JavaScript statement of onLoginScreenEntry() screen class handler

The third JavaScript statement of the `onLoginScreenEntry()` screen class handler is now written.

The next statement ("press `Enter`") is a `doAction` statement. A `doAction` statement executes an action (that is, a special keystroke) on the emulator. It needs as parameter the action to be executed.

The following table describes the list of valid emulator actions for IBM screens:

Table 2 - 8: doAction method codes

Description	Code
PAx [1..3]	KEY_PAx [1..3]
SysReq	KEY_SYSREQ

Table 2 - 8: doAction method codes (...)

Description	Code
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
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

In our case, the action to be performed is: Press `Enter`, so the required code for the `doAction` method is `KEY_ENTER`.

17 Type the fourth JavaScript statement of the `onLoginScreenEntry()` screen class handler: `javelin.doAction("KEY_ENTER");`

18 Press `Enter`:

```
        javelin.send("TOTO");
        javelin.moveCursor(60, 15);
        javelin.send("TITI");
        javelin.doAction("KEY_ENTER");


    // return "redetect";
}
```

Figure 2 - 92: Fourth JavaScript statement of onLoginScreenEntry() screen class handler

The fourth JavaScript statement of the `onLoginScreenEntry()` screen class handler is now written.

The next statement is not truly an automated "manual" action. It only informs the transaction that a waiting time must be observed for the screen to become stable (for the following screen to be loaded). This action is performed through the `waitForDataStable` method.

19 Type the fifth JavaScript statement of the `onLoginScreenEntry()` screen class handler:

```
javelin.waitForDataStable(timeout, threshold);
```

> *"timeout" and "threshold" don't have to be replaced by actual values because they are Javascript variables declared in the Javascript scope of the transaction with default values.*

20 Press `Enter`:

```
        javelin.send("TOTO");
        javelin.moveCursor(60, 15);
        javelin.send("TITI");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    // return "redetect";
}
```

Figure 2 - 93: Fifth JavaScript statement of onLoginScreenEntry() screen class handler

All statements have been written for the `onLoginScreenEntry()` screen class handler.

We must now inform the transaction that once all screen class handler statements have been executed, a new criterion match detection must be carried out on the next screen without extracting data.

This is done through the `redetect` return value.

21 Delete the comment slashes of the `return "redetect"` code line.

The `onLoginScreenEntry()` screen class handler is now properly and fully set:

```
// Entry handler for screen class "LoginScreen"
function onLoginScreenEntry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    redetect - means the algorithm detects again the screen class.
    //    skip     - means the algorithm skips the xmlization process and
    //               directly goes to the exit handler for the current
    //               screen class.
        javelin.send("TOTO");
        javelin.moveCursor(60, 15);
        javelin.send("TITI");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    return "redetect";
}
```

Figure 2 - 94: onLoginScreenEntry() screen class handler definition

22 Save by clicking on the **Save** icon of the CLI application or by pressing the `Ctrl+S` key combination.

We can now check if the transaction performs expected actions on the login screen.

To check if the transaction performs expected actions

Assuming that the **Emulator** currently displays the login screen, right-click on the transaction (`GetArticleData`) then select **Execute** or select the transaction with a left-click and press `F5` key:

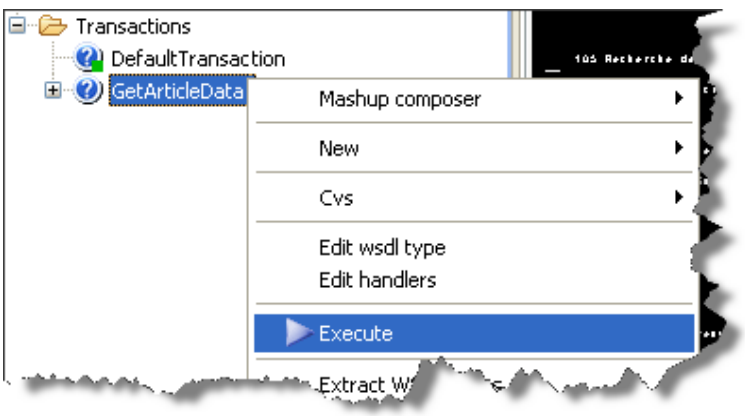


Figure 2 - 95: Executing transaction

Expected result: The transaction connects to the legacy application login screen, enters username and password then accesses the next screen (in our project, the article management general menu `MNS010`). The screen class of the next accessed screen (`MNS010`) is detected and associated extraction rules are executed (in our case, no extraction rule is associated with the `MNS010` screen class).

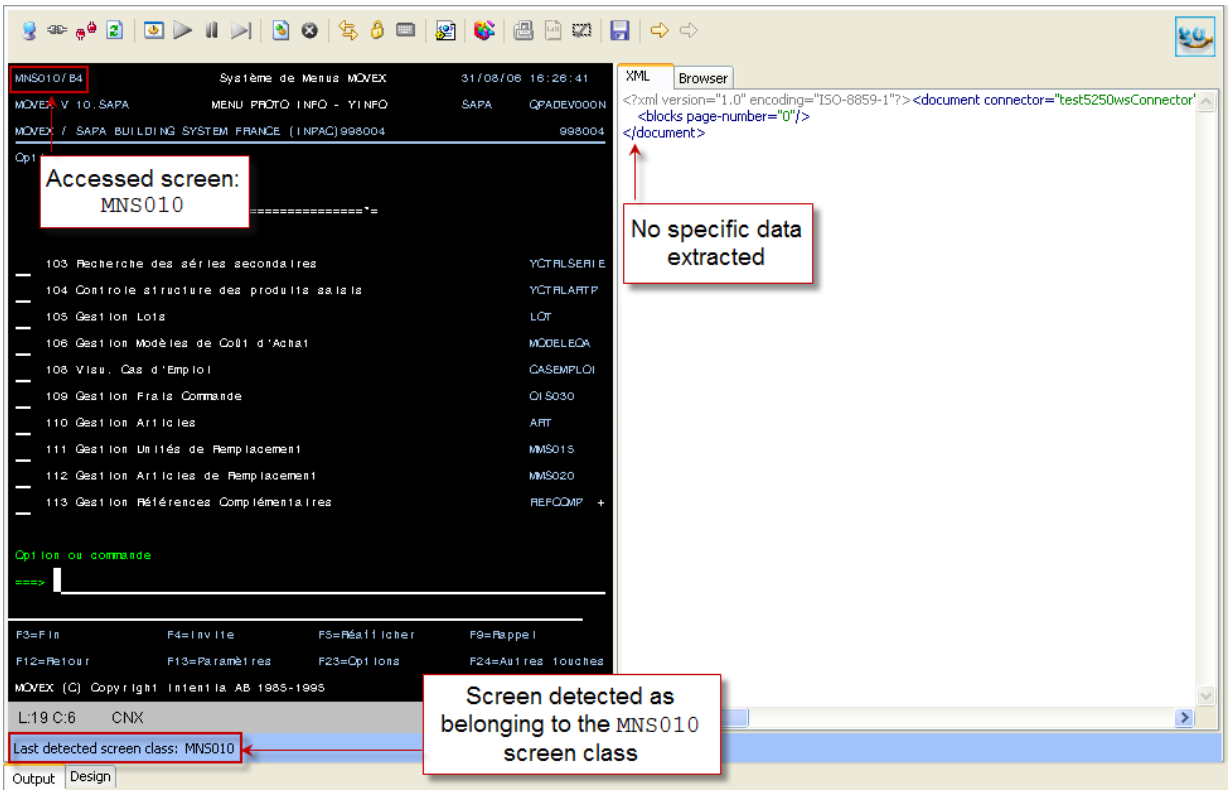


Figure 2 - 96: Accessed screen after execution of transaction

As a conclusion:

- The username and password have been entered.
- The expected screen (MNS010/B4) has been accessed.
- The corresponding screen class (MNS010) has been detected.

The test is conclusive.

The next step in the transaction is to enter, in one of the fields of the `MNS010` screen, the article management code ("`110`") in order to access the next screen (article list screen).

We will now create the screen class handler associated with the `MNS010` screen class, which handles the required data input.

To create the onMNS010Entry() screen class handler

1. In the **Projects View**, right-click on the transaction (for example `GetArticleData`) and select ***New > Handler***:

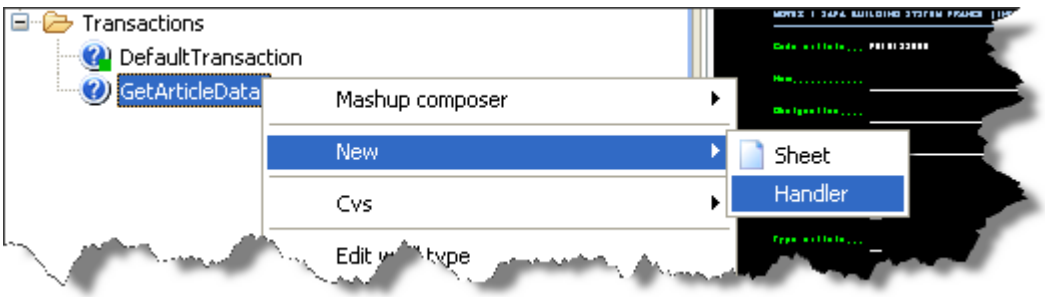


Figure 2 - 97: Creating new handler

A **New transaction handler** window opens.

For more information on the New transaction handler window, see Table "New transaction handler window description" on page 2-44.

The screen class handler we are about to create is triggered on detection of a scren class (`MNS010`), so its parameters are the following:

- this is a *screen class transaction handler*, meaning that all the screen class handler settings of the **New transaction handler** window (see Figure 2 - 80) must be set;
- this is an ***entry*** handler, because no data has to be extracted from the accessed screen;
- it is triggered on detection of the `MNS010` screen class.

We will now set the scren class handler parameters:

2 Check the **Screen class transaction handler** checkbox.

3 In the **ScreenClass** drop-down menu, select `MNS010`.

4 Check the **Entry handler** checkbox.

5 Click on **OK**.

The `onMNS010Entry()` screen class handler appears in the **Functions** folder of the `GetArticleData` transaction of the **Projects View**:

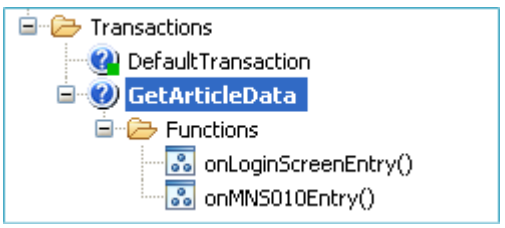


Figure 2 - 98: onMNS010Entry() screen class handler

In parallel, the transaction document is filled with a second JavaScript function called `onMNS010Entry()` corresponding to the `MNS010` screen class:

sample_doc_CLI demo5250Connector | *GetArticleData

```
// Entry handler for screen class "LoginScreen"
function onLoginScreenEntry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    redetect - means the algorithm detects again the screen class.
    //    skip     - means the algorithm skips the xmlization process and
    //               directly goes to the exit handler for the current
    //               screen class.
        javelin.send("TOTO");
        javelin.moveCursor(60, 15);
        javelin.send("TITI");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    return "redetect";
}

// Entry handler for screen class "MNS010"
function onMNS010Entry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    redetect - means the algorithm detects again the screen class.
    //    skip     - means the algorithm skips the xmlization process and
    //               directly goes to the exit handler for the current
    //               screen class.

    // return "redetect";
}
```

Figure 2 - 99: onMNS010Entry() screen class handler definition in Editor

We will now edit the `onMNS010Entry()` screen class handler in the **Editor**, by writing as many JavaScript statements as actions needed in the `MNS010` screen.

To edit the onMNS010Entry() screen class handler

When accessing the article management general menu screen (belonging to the `MNS010` screen class), the actions to be performed are the following:

1 *Move cursor* to "`Option ou commande`" field.

2 *Clean* field of any data.

This action is not mandatory. It is nevertheless recommended before inputing data in a field.

3 *Enter* option "`110`".

4 *Press* `Enter`.

5 Wait for the screen to be stable.

The following table relates screen actions to javelin object methods for the `onMNS010Entry()` screen class handler:

Table 2 - 9: MNS010 screen class - Actions to be performed and methods

Action	Method	Javascript statement
Move cursor	`moveCursor(`*column*`, `*line*`)`	`javelin.moveCursor(column, line);`
Clean field	`doAction(action)`	`javelin.doAction("KEY_ERASEEOF");`
Enter option	`send("`*TEXTSTRING*`")`	`javelin.send("110");`
Press `Enter`	`doAction(`*action*`)`	`javelin.doAction("KEY_ENTER");`
Wait for screen to be stable	`waitForDataStable(timeout, threshold)`	`javelin.waitForDataStable(timeout, threshold);`

For more information on Javascript methods for the javelin object, see the complete Javelin Object Reference here.

When all actions have been performed on a screen, they made the emulator go to another screen class. CLI must redetect the next screen class. If a screen class handler exists for the detected screen class, the screen class handler is triggered, and so on.

To this end, the `redetect` return value is added to the JavaScript function:

```
return "redetect";
```

We will now edit the `onMNS010Entry()` JavaScript function by including the previously described statements in the JavaScript function.

1 In the `onMNS010Entry()` function, position your cursor below comments:

```
// Entry handler for screen class "MNS010"
function onMNS010Entry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    redetect - means the algorithm detects again the screen class.
    //    skip     - means the algorithm skips the xmlization process and
    //               directly goes to the exit handler for the current
    //               screen class.

    // return "redetect";
}
```

Figure 2 - 100: Positioning cursor below comments in MNS010Entry() function

The first statement uses a `moveCursor` method, which takes as parameters the cursor target coordinates (column, line).

2 Click on the **Emulator** tab.

3 Assuming that the **Emulator** displays the `MNS010` screen and that the cursor flashes in the "`Option ou commande`" field, note the cursor coordinates appearing in the lower

right corner of the **Connector View**:

Figure 2 - 101: Cursor position for "move cursor" statement of onMNS010Entry screen class handler

The cursor must be moved to column "6" and line "19", the method is therefore `moveCursor(6, 19)`.

4 Type the first JavaScript statement of the `onMNS010Entry()` screen class handler: `javelin.moveCursor(6, 19);`

Please note that coordinates are given as line (L), column (C) in the Connector View, while it must be entered as column, line in the moveCursor method.

5 Press `Enter`:

```
        javelin.moveCursor(6, 19);

    // return "redetect";
}
```

Figure 2 - 102: First JavaScript statement of onMNS010Entry() screen class handler

The first JavaScript statement of the `onMNS010Entry()` screen class handler is now written.

The second statement of the `onMNS010Entry()` screen class handler can be summed

up as: "empty field". This emulator action is performed using the `doAction` method with the "`KEY_ERASEEOF`" parameter.

> *For more information on doAction parameters,* see Table *"doAction method codes" on page 2-50*.

6 Type the second JavaScript statement of the `onMNS010Entry()` screen class handler: `javelin.doAction("KEY_ERASEEOF");`

7 Press `Enter`:

```
        javelin.moveCursor(6, 19);
        javelin.doAction("KEY_ERASEEOF");

    // return "redetect";
}
```

Figure 2 - 103: Second JavaScript statement of onMNS010Entry() screen class handler

The second JavaScript statement of the `onMNS010Entry()` screen class handler is now written.

The third statement corresponds to the sending of the article management option ("`110`") in the "`Option ou commande`" field. The required method is `send` with the parameter "`110`": `send("110")`.

8 Type the third JavaScript statement of the `onMNS010Entry()` screen class handler: `javelin.send("110");`

9 Press `Enter`:

```
        javelin.moveCursor(6, 19);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("110");

    // return "redetect";
}
```

Figure 2 - 104: Third JavaScript statement of onMNS010Entry() screen class handler

The third JavaScript statement of the `onMNS010Entry()` screen class handler is now written.

The fourth statement of the `onMNS010Entry()` screen class handler is another emulator action, corresponding to the validation of the article management code: "press `Enter`". This action is performed using the `doAction` method with the "`KEY_ENTER`" parameter.

10 Type the fourth JavaScript statement `onMNS010Entry()` screen class handler: `javelin.doAction("KEY_ENTER");`

11 Press `Enter`:

```
        javelin.moveCursor(6, 19);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("110");
        javelin.doAction("KEY_ENTER");

    // return "redetect";
```

Figure 2 - 105: Fourth JavaScript statement of `onMNS010Entry()` *screen class handler*

The fourth JavaScript statement of the `onMNS010Entry()` screen class handler is now written.

The next statement does not correspond to a manual action:"wait for data to be stable". This statement informs the transaction that it must wait for the screen to be stable. This action is performed through the `waitForDataStable` method.

12 Type the fifth JavaScript statement of the `onMNS010Entry()` screen class handler:

```
javelin.waitForDataStable(timeout, threshold);
```

> *"timeout" and "threshold" don't have to be replaced by actual values because they are Javascript variables declared in the Javascript scope of the transaction with default values.*

13 Press `Enter`:

```
        javelin.moveCursor(6, 19);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("110");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    // return "redetect";
```

Figure 2 - 106: Fifth JavaScript statement of onMNS010Entry() screen class handler

All statements have been written for the `onMNS010Entry()` screen class handler.

We must now inform the transaction that once all screen class handler statements have been executed, a new criterion match detection must be carried out on the next screen without extracting data.

This is done through the `redetect` return value.

14 Delete the comment slashes of the `return "redetect"` code line.

The `onMNS010Entry()` screen class handler is now properly and fully set:

```
// Entry handler for screen class "MNS010"
function onMNS010Entry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //     redetect - means the algorithm detects again the screen class.
    //     skip     - means the algorithm skips the xmlization process and
    //                directly goes to the exit handler for the current
    //                screen class.
        javelin.moveCursor(6, 19);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("110");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    return "redetect";
}
```

Figure 2 - 107: onMNS010Entry() screen class handler definition

15 Save by clicking on the **Save** icon of the CLI application or by pressing the `Ctrl+S` key combination.

We can now check if the transaction performs expected actions on the `MNS010` screen.

To check if the transaction performs expected actions

Assuming that the **Emulator** currently displays the login screen, right-click on the transaction (`GetArticleData`) then select **Execute** or select the transaction with a left-click and press `F5` key:

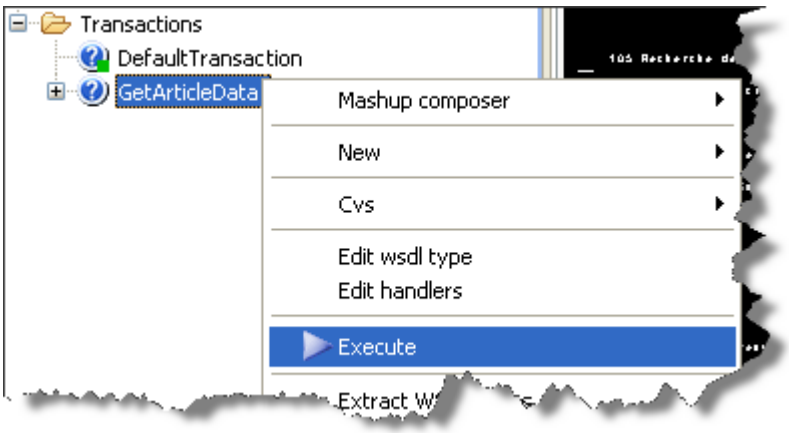


Figure 2 - 108: Executing transaction

Expected result:

The transaction:

1 Enters login information on the login screen then validate.

2 Accesses the `MNS010` screen.

3 Enters the article management code ("`110`") in the "`Option ou commande`" field of

the `MNS010` screen then validates.

4 Accesses the `MMS001/B1 screen` and detects the screen class (`MMS001B1`).

5 Stops. The associated extraction rule (in our case, the `ArticlesTable` extraction rule) is then executed.

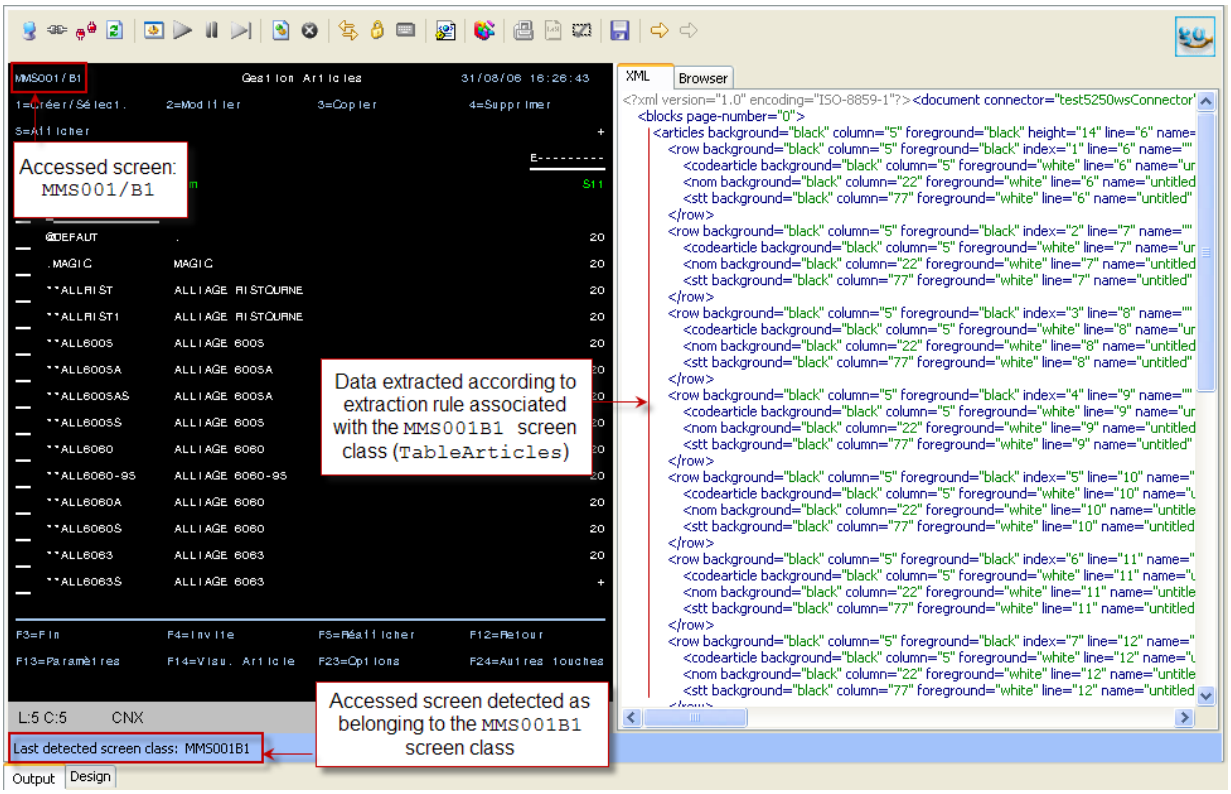


Figure 2 - 109: Accessed screen (MMS001B1) after execution of transaction

As a conclusion:

- Login information has been entered in login screen and validated.
- The following screen (`MNS010`) has been accessed and the corresponding screen class detected.
- The article management code ("`110`") has been entered in `MNS010 screen` and validated.
- The expected screen (`MMS001/B1`) has been accessed.
- The corresponding screen class (`MMS001B1`) has been detected.
- The associated extraction rule (`ArticlesTable`) has been executed.

The test is conclusive.

The next step in the transaction is to input, in one of the fields of the the `MMS001/B1` screen, the article code ("`123456789`");

We will now create the screen class handler which handles the data input, by keeping in mind that data must be extracted from the screen (using the `ArticlesTable` extraction rule) prior to inputing data in the emulator.

This is a major difference in relation to previously created screen class handlers. Indeed, previous screen class handlers were defined as ***entry*** screen class handlers. For data to be

extracted on detection of a screen class, a screen class handler must be defined as an ***exit*** screen class handler.

To create the onMMS001B1Exit() screen class handler

1 In the **Projects View**, right-click on the transaction (for example `GetArticleData`) and select ***New > Handler***:

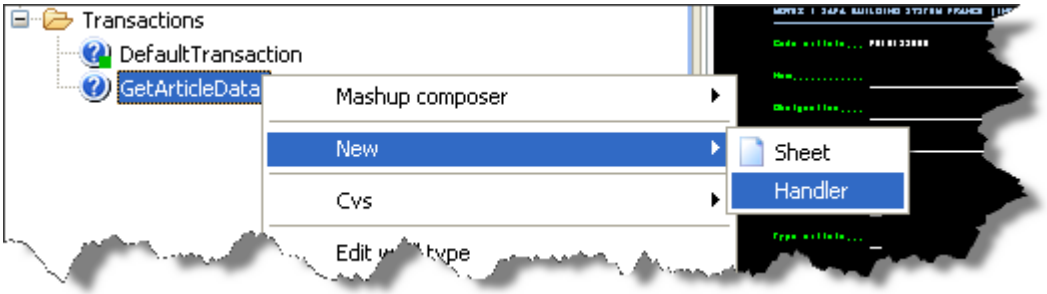


Figure 2 - 110: Creating new handler

A **New transaction handler** window opens.

> *For more information on the New transaction handler window, see Table "New transaction handler window description" on page 2-44.*

The screen class handler we are about to create is triggered on detection of a screen class (`MMS001B1`), so its parameters are the following:

- this is a *screen class transaction handler*, meaning that all the screen class handler settings of the **New transaction handler** window (see Figure 2 - 80) must be set;
- this is an ***exit*** handler, because data has to be extracted from the accessed screen;
- it is triggered on detection of the `MMS001B1` screen class.

We will now set the scren class handler parameters:

2 Check the **Screen class transaction handler** checkbox;

3 In the **ScreenClass** drop-down menu, select `MMS001B1`;

4 Uncheck the **Entry handler** checkbox;

5 Check the **Exit handler** checkbox;

6 Click on **OK**.

The `onMMS001B1Exit()` screen class handler appears in the **Functions** folder of the `GetArticleData` transaction of the **Projects View**:

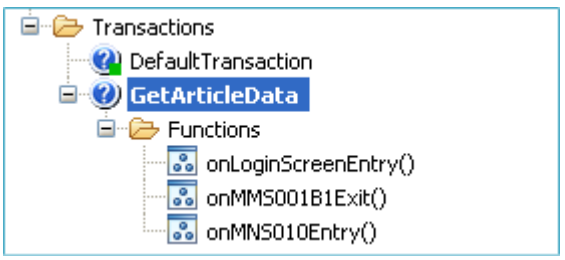


Figure 2 - 111: onMMS001B1Exit() screen class handler

In parallel, the transaction document is filled with a third JavaScript function called `onMMS001B1Exit()` corresponding to the `MMS001B1` screen class:

```
sample_doc_CLI demo5250Connector    *GetArticleData

    return "redetect";
}

// Entry handler for screen class "MNS010"
function onMNS010Entry() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    redetect - means the algorithm detects again the screen class.
    //    skip     - means the algorithm skips the xmlization process and
    //               directly goes to the exit handler for the current
    //               screen class.
        javelin.moveCursor(6, 19);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("110");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    return "redetect";
}

// Exit handler for screen class "MMS001B1"
function onMMS001B1Exit() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    accumulate - means the algorithm accumulates XML data and go to
    //                 the next detected screen class.

    // return "accumulate";
}
```

Figure 2 - 112: onMMS001B1Exit() screen class handler definition in Editor

We will now edit the `onMMS001B1Exit()` screen class handler in the **Editor**, by writing as many JavaScript statements as actions needed in the `MMS001B1` screen.

To edit the onMMS001B1Exit() screen class handler

When accessing the article list screen (belonging to the `MMS001B1` screen class), the actions to be performed are similar to actions performed on the `MNS010` screen following:

1 *Move cursor* to "`Code article`" field.

2 *Clean* field of any data.

This action is not mandatory. It is nevertheless recommended before inputing data in a field.

3 *Enter* article code "`123456789`".

4 *Press* `Enter`.

5 Wait for the screen to be stable.

The following table relates actions to methods for the `onMMS001B1Exit()` screen class handler:

Table 2 - 10: MMS001B1 screen class - Actions to be performed and methodsB

Action	Method	Javascript statement
Move cursor	`moveCursor(`*column, line*`)`	`javelin.moveCursor(column, line);`
Clean field	`doAction(action)`	`javelin.doAction("KEY_ERASEEOF");`
Enter product number	`send("`*TEXTSTRING*`")`	`javelin.send("123456789");`
Press `Enter`	`doAction(`*action*`)`	`javelin.doAction("KEY_ENTER");`
Wait for screen to be stable	`waitForDataStable(timeout, threshold)`	`javelin.waitForDataStable(timeout, threshold)`

For more information on Javascript methods for the javelin object, see the complete Javelin Object Reference here.

The `onMMS001B1Exit()` screen class handler being defined as an ***exit*** screen class handler (see Table *"Screen class handler types"* on page 1-12), the extraction rule (`ArticlesTable`) associated with the `MMS001B1` screen class is executed prior to statements.

In our project, we want final extracted data (data extracted from article description screen, see Figure 2 - 6) to be added to data extracted using the `ArticlesTable` extraction rule. To this end, the `accumulate` return value is added to the JavaScript function:

```
return "accumulate";
```

We will now edit the `onMMS001B1Exit()` JavaScript function by including the previously described statements in the JavaScript function.

1 In the `onMMS001B1Exit()` function, position your cursor below comments:

```
// Exit handler for screen class "MMS001B1"
function onMMS001B1Exit() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    accumulate - means the algorithm accumulates XML data and go to
    //                 the next detected screen class.

    // return "accumulate";
}
```

Figure 2 - 113: Positioning cursor below comments in MMS001B1Exit() function

The first statement uses a `moveCursor` method, which takes as parameters the cursor target coordinates (column, line).

2 Click on the **Emulator** tab.

3 Assuming that the **Emulator** displays the `MMS001B1` screen and that the cursor flashes in the "`Code article`" field, note the cursor coordinates appearing in the lower left corner of the **Connector View**:

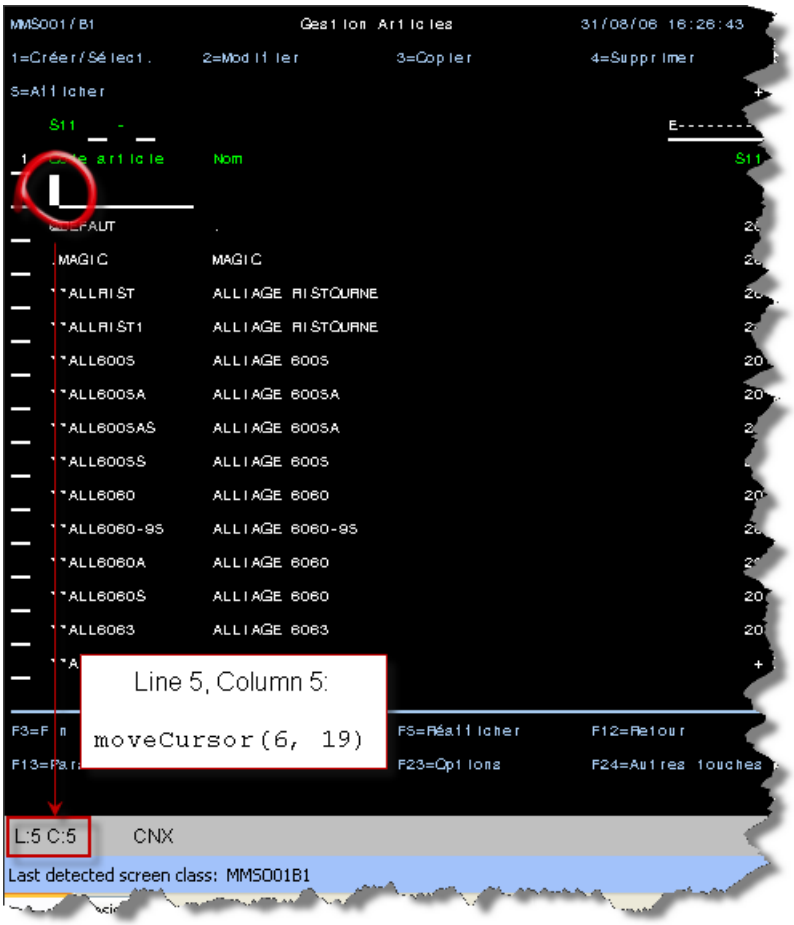


Figure 2 - 114: Cursor position for "move cursor" statement of onMMS001B1Exit screen class handler

The cursor must be moved to column "5" and line "5", the method is therefore `moveCursor(5, 5)`.

4 Type the first JavaScript statement of the `onMMS001B1Exit()` screen class handler: `javelin.moveCursor(5, 5);`

Please note that coordinates are given as line (L), column (C) in the Connector View, while it must be entered as column, line in the `moveCursor` method.

5 Press `Enter`:

```
        javelin.moveCursor(5, 5);

    // return "accumulate";
```

Figure 2 - 115: First JavaScript statement of onMMS001B1Exit() screen class handler

The first JavaScript statement of the `onMMS001B1Exit()` screen class handler is now written.

The second statement of the `onMMS001B1Exit()` screen class handler can be summed up as: "empty field". This emulator action is performed using the `doAction` method with the "`KEY_ERASEEOF`" parameter.

For more information on doAction parameters, see Table *"doAction method codes" on page 2-50*.

6 Type the second JavaScript statement of the `onMMS001B1Exit()` screen class handler: `javelin.doAction("KEY_ERASEEOF");`

7 Press `Enter`:

```
        javelin.moveCursor(5, 5);
        javelin.doAction("KEY_ERASEEOF");

    // return "accumulate";
}
```

Figure 2 - 116: Second JavaScript statement of onMMS001B1Exit() screen class handler

The second JavaScript statement of the `onMMS001B1Exit()` screen class handler is now written.

The third statement corresponds to the sending of the article code ("`123456789`") in the "`Code article`" field. The required method is `send` with the parameter "`123456789`": `send("123456789")`.

8 Type the third JavaScript statement of the `onMMS001B1Exit()` screen class handler: `javelin.send("123456789");`

9 Press `Enter`:

```
        javelin.moveCursor(5, 5);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("123456789");

    // return "accumulate";
}
```

Figure 2 - 117: Third JavaScript statement of onMMS001B1Exit() screen class handler

The third JavaScript statement of the `onMMS001B1Exit()` screen class handler is now written.

The fourth statement of the `onMMS001B1Exit()` screen class handler is another emulator action, corresponding to the validation of the article code: "press `Enter`". This action is performed using the `doAction` method with the "`KEY_ENTER`" parameter.

10 Type the fourth JavaScript statement `onMMS001B1Exit()` screen class handler: `javelin.doAction("KEY_ENTER");`

11 Press `Enter`:

```
        javelin.moveCursor(5, 5);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("123456789");
        javelin.doAction("KEY_ENTER");

    // return "accumulate";
}
```

Figure 2 - 118: Fourth JavaScript statement of `onMMS001B1Exit()` screen class handler

The fourth JavaScript statement of the `onMMS001B1Exit()` screen class handler is now written.

The next statement is not truly an automated "manual" action. It only informs the transaction that a waiting time must be observed for the screen to become stable. This action is performed through the `waitForDataStable` method.

12 Type the fifth JavaScript statement of the `onMMS001B1Exit()` screen class handler:

```
javelin.waitForDataStable(timeout, threshold);
```

"timeout" and "threshold" don't have to be replaced by actual values because they are Javascript variables declared in the Javascript scope of the transaction with default values.

13 Press `Enter`:

```
        javelin.moveCursor(5, 5);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("123456789");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);


    // return "accumulate";
}
```

Figure 2 - 119: Fifth JavaScript statement of onMMS001B1Exit() screen class handler

All statements have been written for the `onMMS001B1Exit()` screen class handler.

We must now inform the transaction that all extracted data must be added one to another (in our project, data extracted using *XML tag names* extraction rules must be added to data extracted using the *ArticlesTable* extraction rule).

This is done through the `accumulate` return value.

14 Delete the comment slashes of the `return "accumulate"` code line.

The `onMMS001B1Exit()` screen class handler is now properly and fully set:

```
// Exit handler for screen class "MMS001B1"
function onMMS001B1Exit() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //     accumulate - means the algorithm accumulates XML data and go to
    //                  the next detected screen class.
        javelin.moveCursor(5, 5);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send("123456789");
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    return "accumulate";
}
```

Figure 2 - 120: onMMS001B1Exit() screen class handler definition

15 Save by clicking on the **Save** icon of the CLI application or by pressing the `Ctrl+S` key combination.

We can now check if the transaction performs expected actions on the `MMS001B1` screen.

To check if the transaction performs expected actions

Assuming that the **Emulator** currently displays the login screen, right-click on the transaction (`GetArticleData`) then select **Execute** or select the transaction with a left-click and press

F5 key:

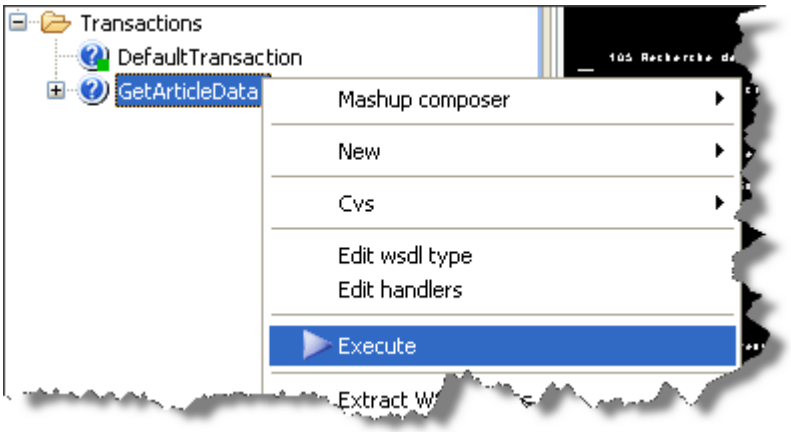


Figure 2 - 121: Executing transaction

Expected result:

The transaction:

1 Enters login information on the login screen then validate.

2 Accesses the `MNS010` screen.

3 Enters the article management code ("`110`") in the "`Option ou commande`" field of the `MNS010` screen then validates.

4 Accesses the `MMS001/B1` screen. The associated extraction rule (`ArticlesTable`) is executed.

5 Enters the article code ("`123456789`") in the "`Code article`" field of the `MMS001/B1` screen then validates.

6 Accesses the `MMS001/E` screen.

7 Stops. Associated extraction rules (`Tag article_status`, `Tag article_rsp` and `Tag product_group`) are executed.

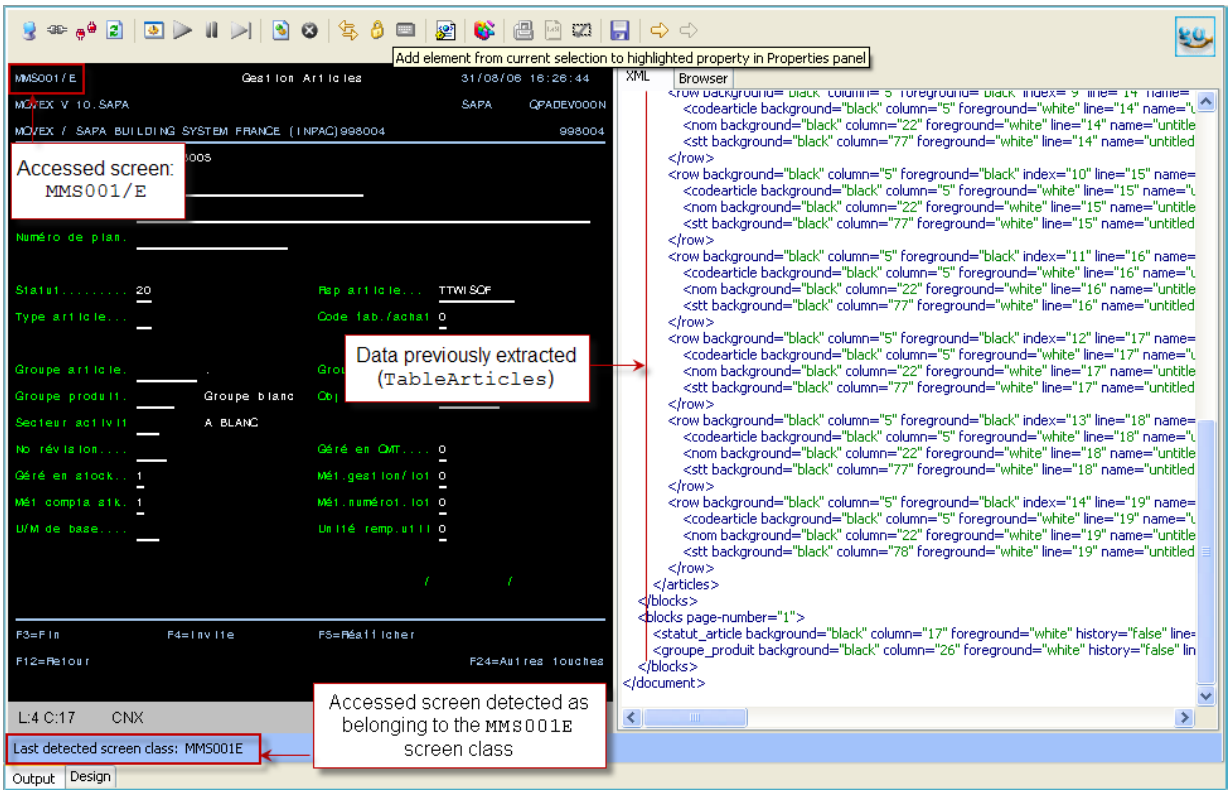


Figure 2 - 122: Accessed screen (MMS001E) after execution of transaction

As a conclusion:

- Login information has been entered in login screen and validated.
- The following screen (`MNS010`) has been accessed and the corresponding screen class (`MNS010`) detected.
- The article management code ("`110`") has been correctly entered in `MNS010` screen and validated.
- The following screen (`MMS001/B1`) has been accessed, the corresponding screen class (`MMS001B1`) has been detected and the associated extraction rule (`ArticlesTable`) has been executed.
- The article code ("`123456789`") has been entered in `MMS00B1` screen and validated.
- The following screen (`MMS001/E`) has been accessed, the corresponding screen class (`MMS001E`) has been detected and the associated extraction rules (`Tag article_status`, `Tag article_rsp` and `Tag product_group`) have been executed.

The test is conclusive.

The article code entered in the article list screen being a variable, we will now create a variable for the transaction. Using a variable gives more flexibility, since the article code is no longer "hard-typed" in the transaction JavaScript code. When defined as a variable, the article code can be entered when calling the transaction from a Web browser (see ...).

To set a transaction variable

We will now set the transaction *variable,* that is the transaction input data (i.e. article code).

Any transaction variable is added to the executed transaction JavaScript scope.

1 In the **Projects View**, right-click on the transaction (`for example GetArticleData`).

A contextual menu appears.

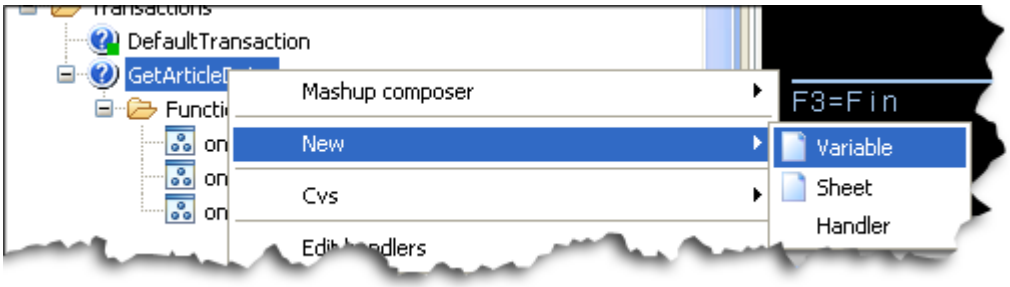


Figure 2 - 123: Creating a new transaction variable

2 Select ***New > Variable***.

A **New Variable** wizard is automatically launched.

3 Select **Variable**.

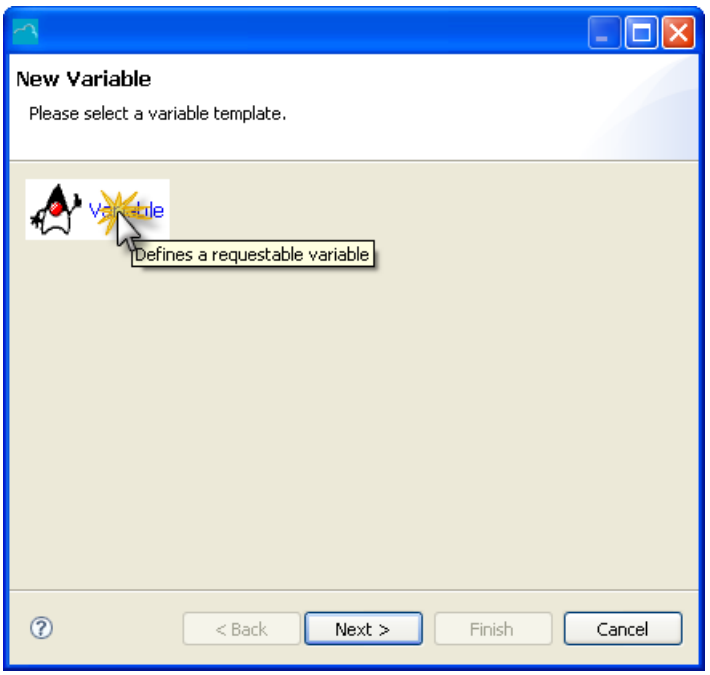


Figure 2 - 124: New Variable wizard

4 Click on **Next**;

5 In the **Name** field that appears, type in the name of the variable (for example `article_no`).

Figure 2 - 125: Giving a name to the new variable

6 Click on **Finish**.

The new variable appears in the transaction **Variables** folder in the **Projects View**:

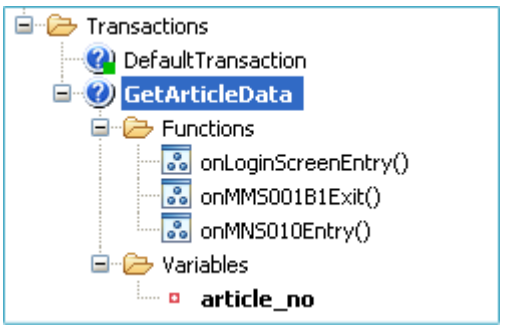


Figure 2 - 126: New variable in transaction Variables folder

7 Save your project by clicking on or by pressing `Ctrl + S`.

The transaction variable is now created and saved. We can now set its properties.

8 Select the `article_no` variable in the **Projects View**.

The **Properties View** is automatically updated:

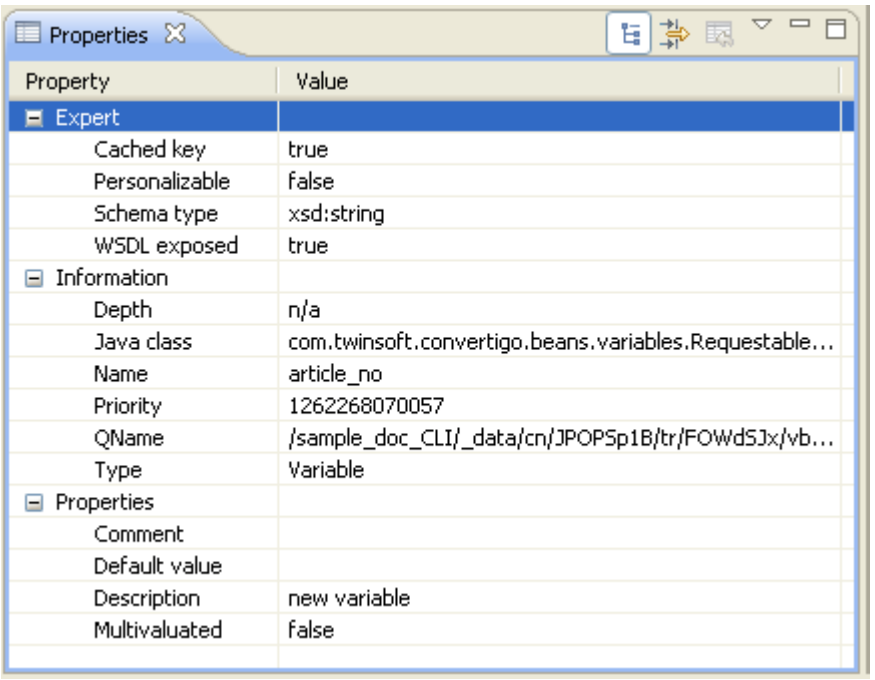


Figure 2 - 127: Variable properties

9 Click in the Value column of the **Description** property.

The property default value (`new variable`) is highlighted.

Figure 2 - 128: Description variable property

10 Type in a description for the variable (for example `article code`).

11 Press `Enter`.

The property's value is updated. We will now set its **Default value** property, used for transaction testing purpose, by repeating the three previous steps.

12 Click in the Value column of the **Default value** property.

The property is highlighted.

Figure 2 - 129: Default value variable property

13 Type in a default value for the variable (for example `123456789`).

14 Press `Enter`.

In the context of this Quick Guide project, all other values are left as default values.

The transaction variable is now created and properly set.

15 Save your project by clicking on or by pressing `Ctrl + S`.

We will now change the transaction definition by replacing the hard-typed article code (`"123456789"`) with the variable.

16 Click on the `GetArticleData` tab to display the **Editor**.

17 Find the following JavaScript statement in the `onMMS001B1Exit()` function: `javelin.send("123456789");`

18 Replace `"123456789"` with the variable: `article_no`:

```
// Exit handler for screen class "MMS001B1"
function onMMS001B1Exit() {
    // TODO: add your code here

    // TODO: customize the returned value (if you omit returned value, the
    // algorithm will continue its process).
    // Possible values are:
    //    accumulate - means the algorithm accumulates XML data and go to
    //                 the next detected screen class.
        javelin.moveCursor(5, 5);
        javelin.doAction("KEY_ERASEEOF");
        javelin.send(no_article);
        javelin.doAction("KEY_ENTER");
        javelin.waitForDataStable(timeout, threshold);

    return "accumulate";
}
```

Variable in JavaScript statement

Figure 2 - 130: Variable in onMMS001B1Exit() Javascript function

To check if the transaction still returns expected results, follow the instructions of *"To check if the transaction performs expected actions"* on page 2 - 69.

The setting up of the transaction is now complete. Another method for testing a transaction is through the use of a test platform which is included in the project. The next section describes how to use such test platform.

2.2.5 Executing a Transaction using the Test Platform

The Legacy integration project is now set up.

You can test the transaction either:

- by executing it from the Studio using the **Execute** command (see *"To check if the transaction performs expected actions"* on page 2-69);
- by calling a properly formatted URL from a Web browser to access a test platform included in the project.

To test the transaction using the test platform

1. Launch a standard Web browser;
2. In the URL address field, type in the URL in the following format:

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/projects/
<ProjectName>
```

For example, in our project:

- *HostName* = `localhost`;
- *ConvertigoPortNumber* = `18080`;
- *ProjectName* = `sample_doc_CLI`;

For example, in a browser URL address bar:

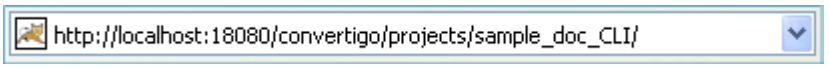


Figure 2 - 131: Convertigo test platform URL

The test platform opens:

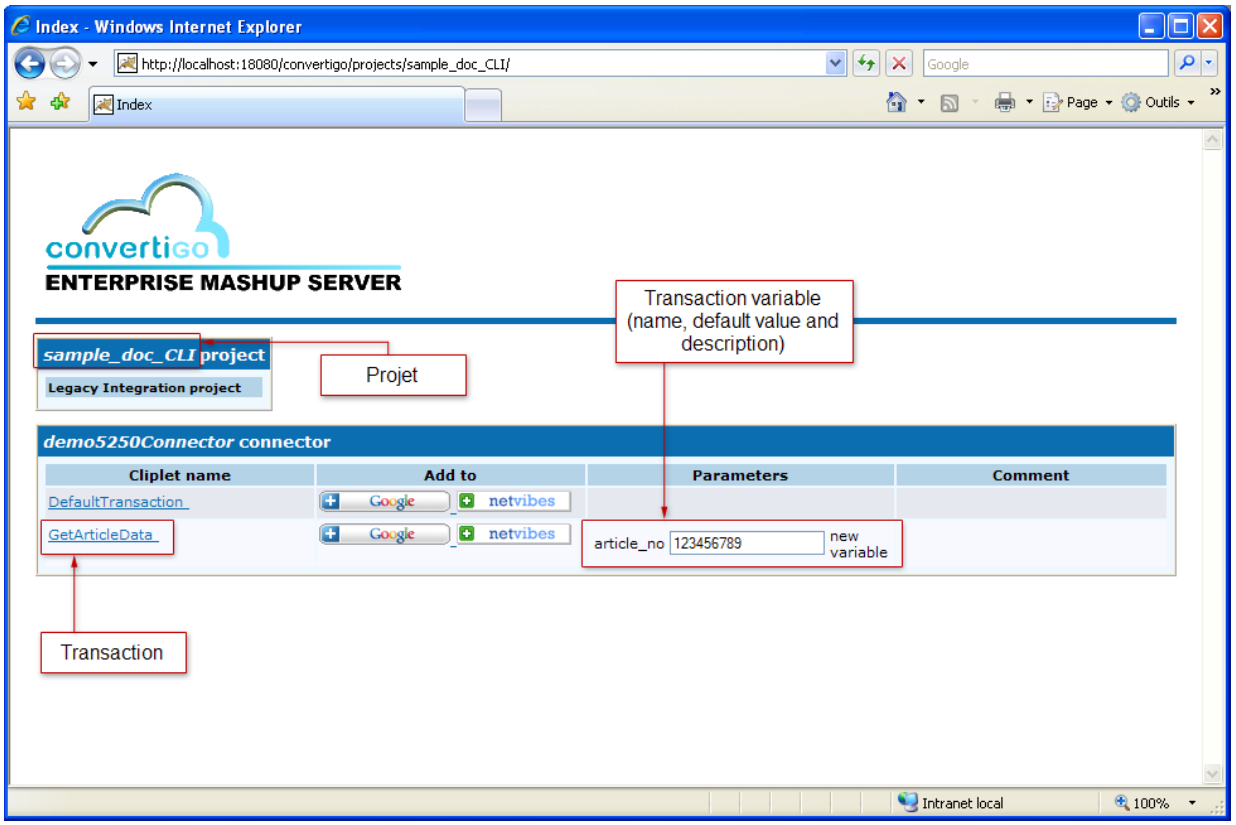


Figure 2 - 132: Test platform

It displays:

- in a first table:
  - the **project name** (in our example, `sample_doc_CLI`);
  - the **project type** (a Legacy Integration project);
- in a second table:
  - the ***connector*** defined for the project (in our example, `demo5250Connector`);
  - all ***transactions*** defined for the connector (in our example, `DefaultTransaction`, `GetArticleData`) and their *variables* (`article_no` for the `GetArticleData` transaction).

> *All information defined for the variable (description, default value, etc) appears in the **Parameters** column.*

3 Replace the default article code with another value (for example `PO987654`);

4 Click on the `GetArticleData` link.

The transaction starts. The **Connector Editor** of the CLI Studio shows the successive steps of the transaction being executed in real time.

5 Once the transaction has been completed, data extracted according to the `ArticlesTable`, `Tag article_status`, `Tag article_rsp` and `Tag product_group` extraction rules appears in an **Execution result** window in the test platform:

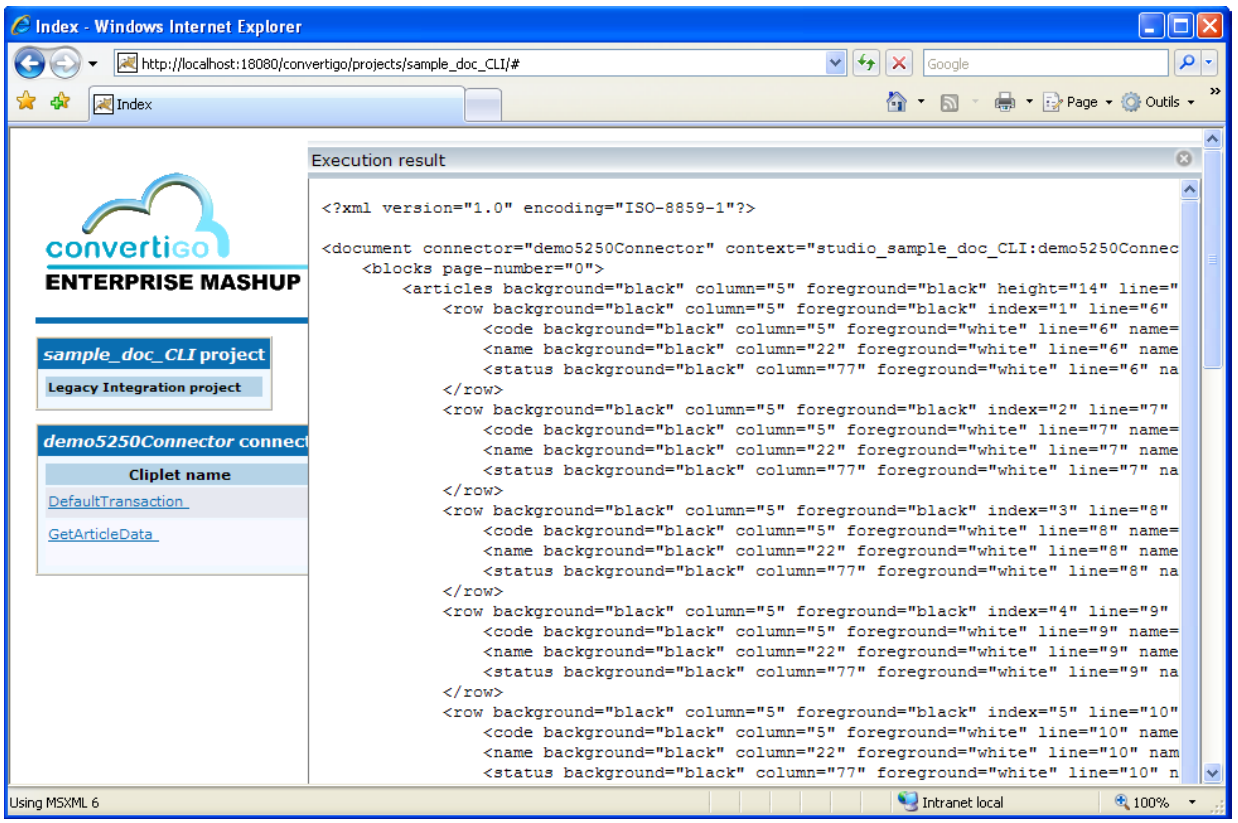


Figure 2 - 133: XML output in the Execution result window of the test platform

3 Exposing Projects as Web Services and Presenting Data

The previous chapter describes how to extract data into a simple XML output document.

The first section of this chapter introduces Web services and gives instructions on how to expose projects either as REST or SOAP Web services.

The second section is dedicated to XSL transformation. It describes how to transform XML data into an HTML document for display purpose using XSL style sheets.

- Exposing Projects as Web Services
- XSL AJAX Presentation

Examples of this chapter are based on the Web integration project described in chapter "My First Convertigo Legacy Integrator Project" on page 2 - 1.

3.1 Exposing Projects as Web Services

Convertigo Legacy Integrator allows to expose projects either as REST or as SOAP Web services.

Exposing projects as REST Web services is a straightforward process: it requires calling the transaction from any Web Browser via a properly formatted URL.

Exposing projects as SOAP Web services requires setting transactions as public methods and selecting data types before generating SOAP request and response envelopes.

This section describes how to expose projects as Web services in both architectures:

- Exposing a Project as a REST Web Service
- Exposing a Project as a SOAP Web Service

3.1.1 Exposing a Project as a REST Web Service

CLI transactions are by design exposed as methods of a REST Web service: each transaction can be called via an URL; each transaction is a REST method providing data.

Transactions result in the generation of an XML document including data represented in XML

format.

The URL syntax used for the external call is *standard* URL syntax.

To invoke a transaction as a REST Web service method

1. Launch your default Web Browser;
2. In the URL address bar, type in the URL in the following format:

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/projects/
<ProjectName>/.xml?__transaction=
<TransactionName>&<VariableName>=<VariableValue>
```

For example, in our project:

- *HostName* = `localhost`;
- *ConvertigoPortNumber* = `18080`;
- *ProjectName* = `sample_doc_CLI`;
- *TransactionName* = `GetArticleData`;
- *VariableName* = `no_article`;
- *VariableValue* = `PO987654`;

For example, in a Mozilla Firefox URL address bar:

Figure 3 - 1: URL syntax for exposing transactions as REST Web services

3. Press **Enter**.

   The transaction is launched in the CLI Engine.

4. Switch to the **CLI Studio**.

   In the **Emulator**, you can see Convertigo navigating through legacy application screens (login screen, `MNS010`, `MMS001B1`, then `MMS001E`), inputing data (either data hard-typed in the transaction JavaScript code or variable default value data).

5. Click on the **Output** tab of the **Connector View**.

   The view displays extracted data in XML format.

6. Switch to the Web Browser launched in step 1.

   CLI has sent the extracted data in XML format to its client:

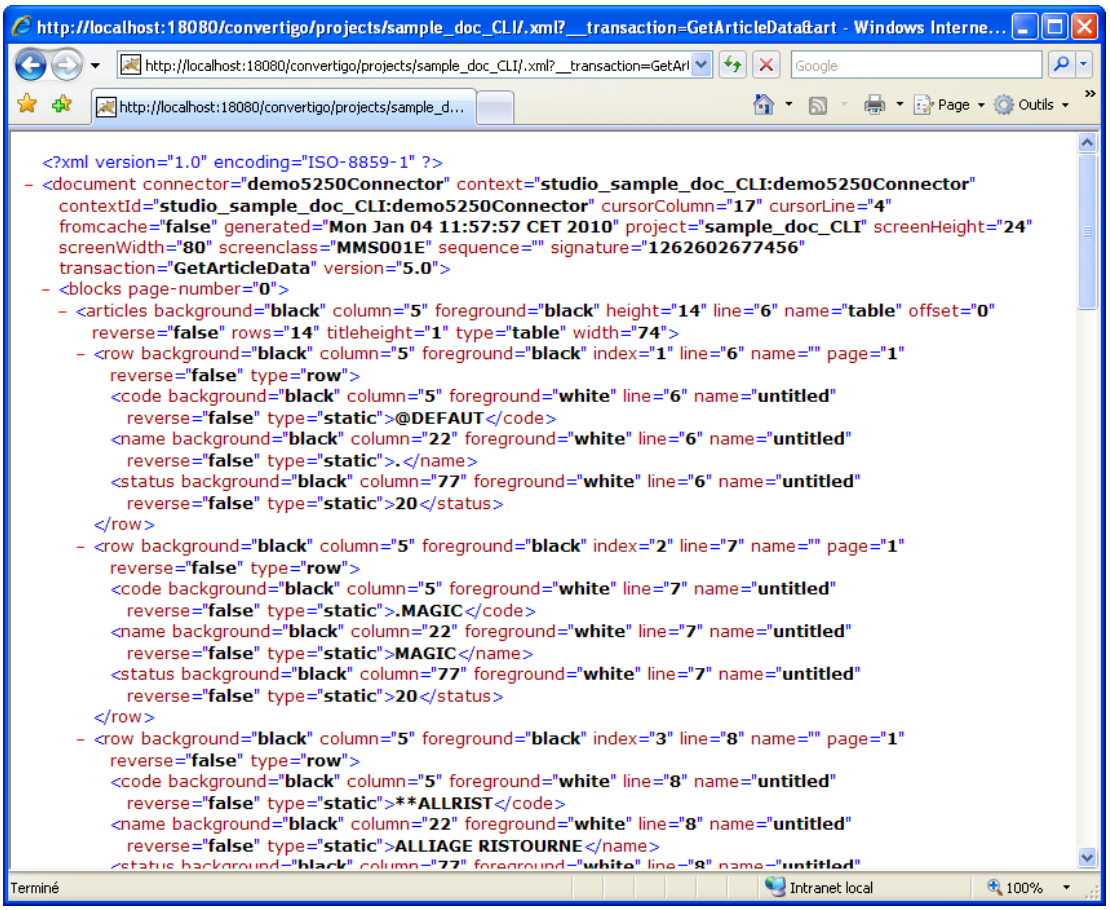


Figure 3 - 2: Transaction exposed as a REST Web service

3.1.2 Exposing a Project as a SOAP Web Service

A CLI project can be seen as a SOAP *service* and CLI transactions as SOAP *methods.*

Exposing projects (resp. transactions) as SOAP Web services (resp. Web service methods) requires:

1 *Informing* Convertigo that **one or several** transactions will be exposed as SOAP Web service methods; this is done by setting one or several transactions as *public methods* through the **Public Method** transaction property.

2 *Describing* all methods and *request/response* SOAP envelopes for these methods in WSDL (*Web Services Description Language*) format.

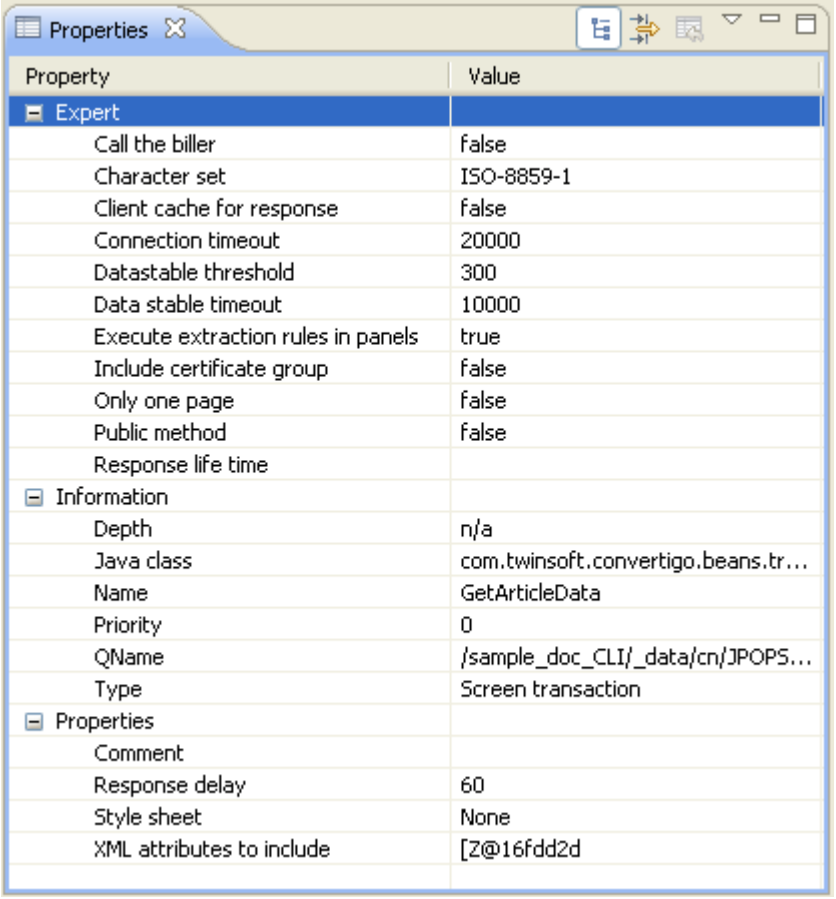
For more information on WSDL, see the WSDL tutorial from W3Schools available at http://www.w3schools.com/wsdl/.

The following procedure gives instructions on how to to expose transactions as SOAP Web service methods.

To expose transactions as SOAP Web service methods

1 In the **Projects View**, click on the transaction (`GetArticleData` in our project).

The **Properties View** is automatically updated:



Properties

Property	Value
Expert	
Call the biller	false
Character set	ISO-8859-1
Client cache for response	false
Connection timeout	20000
Datastable threshold	300
Data stable timeout	10000
Execute extraction rules in panels	true
Include certificate group	false
Only one page	false
Public method	false
Response life time	
Information	
Depth	n/a
Java class	com.twinsoft.convertigo.beans.tr...
Name	GetArticleData
Priority	0
QName	/sample_doc_CLI/_data/cn/JPOPS...
Type	Screen transaction
Properties	
Comment	
Response delay	60
Style sheet	None
XML attributes to include	[Z@16fdd2d

Figure 3 - 3: GetArticleData transaction properties

2 Set the **Public Method** property to `true` and press **Enter**:

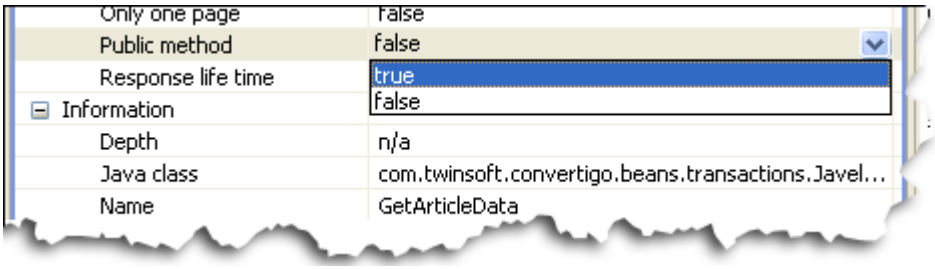


Figure 3 - 4: Setting the Public Method property to true

3 Save the project by clicking on or by pressing `Ctrl + S`.

When the **Public Method** property is set to `true`, the transaction is exposed in the SOAP WSDL as a Web service method. The `GetArticleData` transaction created in our sample project will therefore be exposed as a Web service method, and the `sample_doc_CLI` project as a Web service.

We now need to generate the schema (in XML format) of:

- the project,
- each transaction (one in our case).

The overall schema structure for the whole project (for the `sample_doc_CLI` Web service) is automatically generated by Convertigo. Schema parts corresponding to each

exposed transaction are automatically generated as well, but expected data must be set prior to generating them.

To sum up, for a transaction:

- The *request schema* is automatically generated by CLI.
- The *response schema* is generated by CLI based on a sample output XML.

The WSDL is automatically generated in both "RPC" and "document/literal" modes.

4 Assuming that the transaction has been executed and that its response XML is displayed in the **Output** tab of the **Connector View**, right-click on the transaction in the **Projects View**;

5 Select **Update schema from current generated XML**:

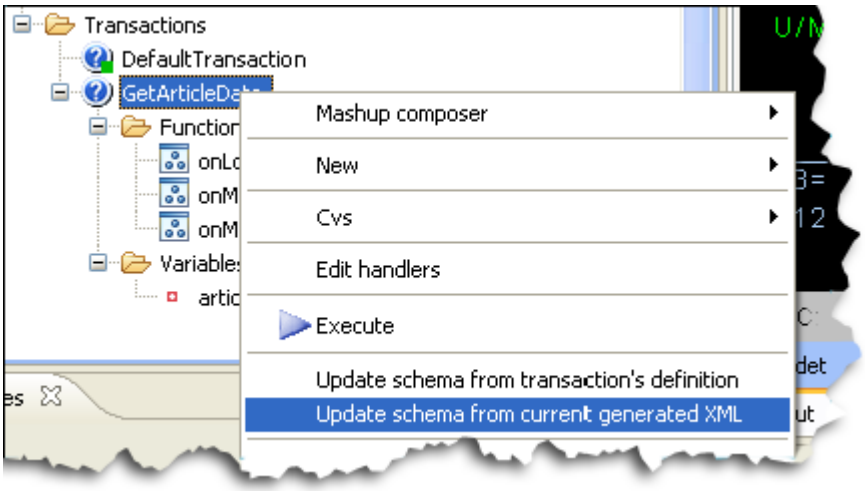


Figure 3 - 5: Update transaction schema from current generated XML

A confirmation window opens:

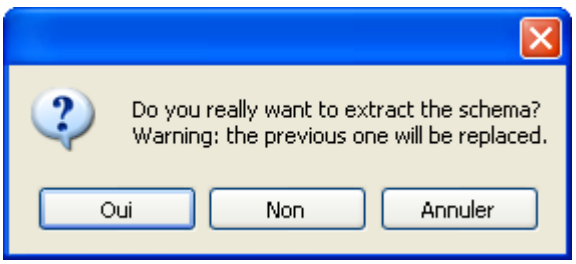


Figure 3 - 6: Update transaction schema - Confirmation window

6 Click on **Yes**.

Clicking on Yes overwrites previously stored schema.

If no XML document has been generated priori to launching the **Update schema from**

current generated XML, a window opens:

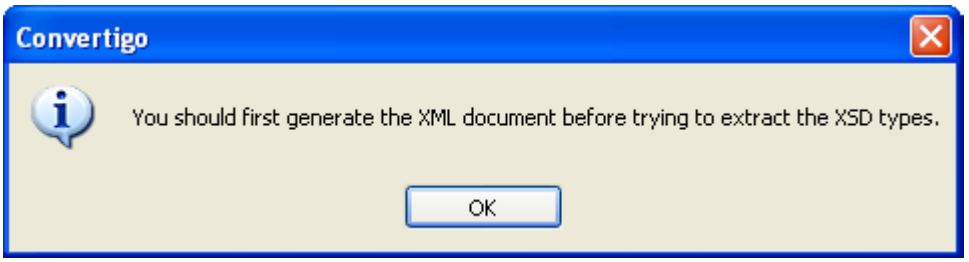


Figure 3 - 7: Generate XML first

Should it be your case, generate the XML document first by following the instructions of:

- *"To check if the transaction performs expected actions"* on page 2 - 69, or
- *"To test the transaction using the test platform"* on page 2 - 76.

Otherwise, the **transaction response schema** is generated: the `GetArticleData` transaction appears bold in the **Projects View**.

7 Save the project by clicking on or by pressing `Ctrl + S`.

Data types used to describe the transaction and all other types generated by CLI for this project are fully described in the `sample_doc_CLI.xsd` file. It can be opened in an **Editor**.

To edit the `sample_doc_CLI.xsd` file

1 In the **Projects View,** click on the **Project Explorer** tab.

2 Expand the project folder.

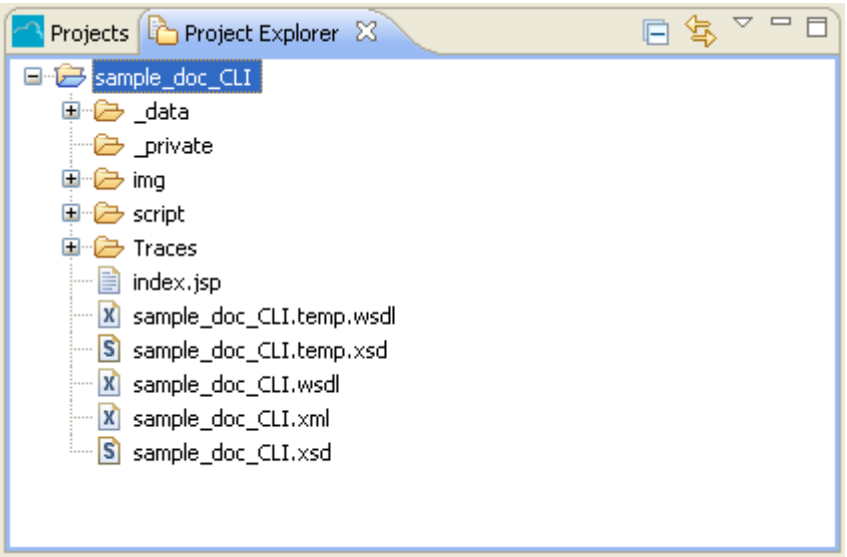


Figure 3 - 8: Project folder in Project Explorer

3 Doucle-click on the `sample_doc_CLI.xsd` file to open it.

The `sample_doc_CLI.xsd` file appears in the **Editor**:

- in the **Design** tab of the **Editor**, the schema appears as follows:

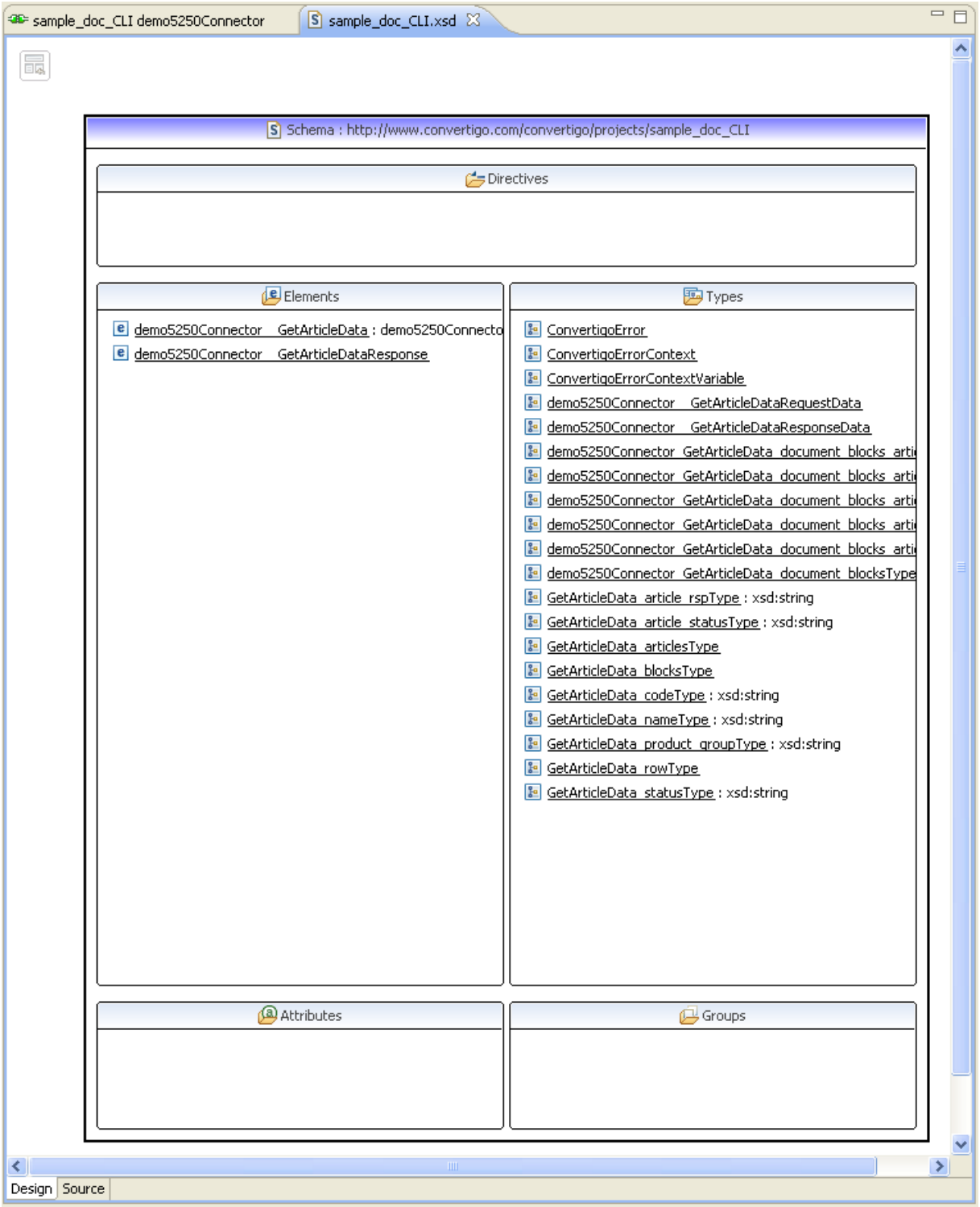


Figure 3 - 9: Project schema in Design tab of Editor

- in the **Source** tab of the **Editor**:

sample_doc_CLI demo5250Connector | sample_doc_CLI.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:sample_doc_CLI_ns="http:/
<xsd:complexType name="ConvertigoError">
<xsd:sequence>
<xsd:element name="context" type="sample_doc_CLI_ns:ConvertigoErrorContext"/>
<xsd:element name="exception" type="xsd:string"/>
<xsd:element name="message" type="xsd:string"/>
<xsd:element name="stacktrace" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ConvertigoErrorContextVariable">
<xsd:attribute name="name" type="xsd:string"/>
<xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="ConvertigoErrorContext">
<xsd:sequence>
<xsd:element maxOccurs="unbounded" minOccurs="0" name="variable" type="sample_doc_CLI_ns
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GetArticleData_blocksType">
<xsd:sequence>
<xsd:element minOccurs="0" name="articles" type="sample_doc_CLI_ns:GetArticleData_articl
<xsd:element minOccurs="0" name="article_status" type="sample_doc_CLI_ns:GetArticleData_
<xsd:element minOccurs="0" name="article_rsp" type="sample_doc_CLI_ns:GetArticleData_art
<xsd:element minOccurs="0" name="product_group" type="sample_doc_CLI_ns:GetArticleData_p
</xsd:sequence>
<xsd:attribute name="page-number" use="optional"/>
</xsd:complexType>
<xsd:complexType name="GetArticleData_article_rspType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="background"/>
<xsd:attribute name="column"/>
<xsd:attribute name="foreground"/>
<xsd:attribute name="history"/>
<xsd:attribute name="line"/>
<xsd:attribute name="name"/>
<xsd:attribute name="reverse"/>
<xsd:attribute name="size"/>
<xsd:attribute name="type"/>
<xsd:attribute name="underline"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="GetArticleData_articlesType">
<xsd:sequence>
<xsd:element maxOccurs="unbounded" minOccurs="0" name="row" type="sample_doc_CLI_ns:GetA
</xsd:sequence>
<xsd:attribute name="background"/>
<xsd:attribute name="column"/>
<xsd:attribute name="foreground"/>
```

Design | Source

Figure 3 - 10: Project schema in Source tab of Editor

Data types generated in step 6 of previous process appear in the schema:

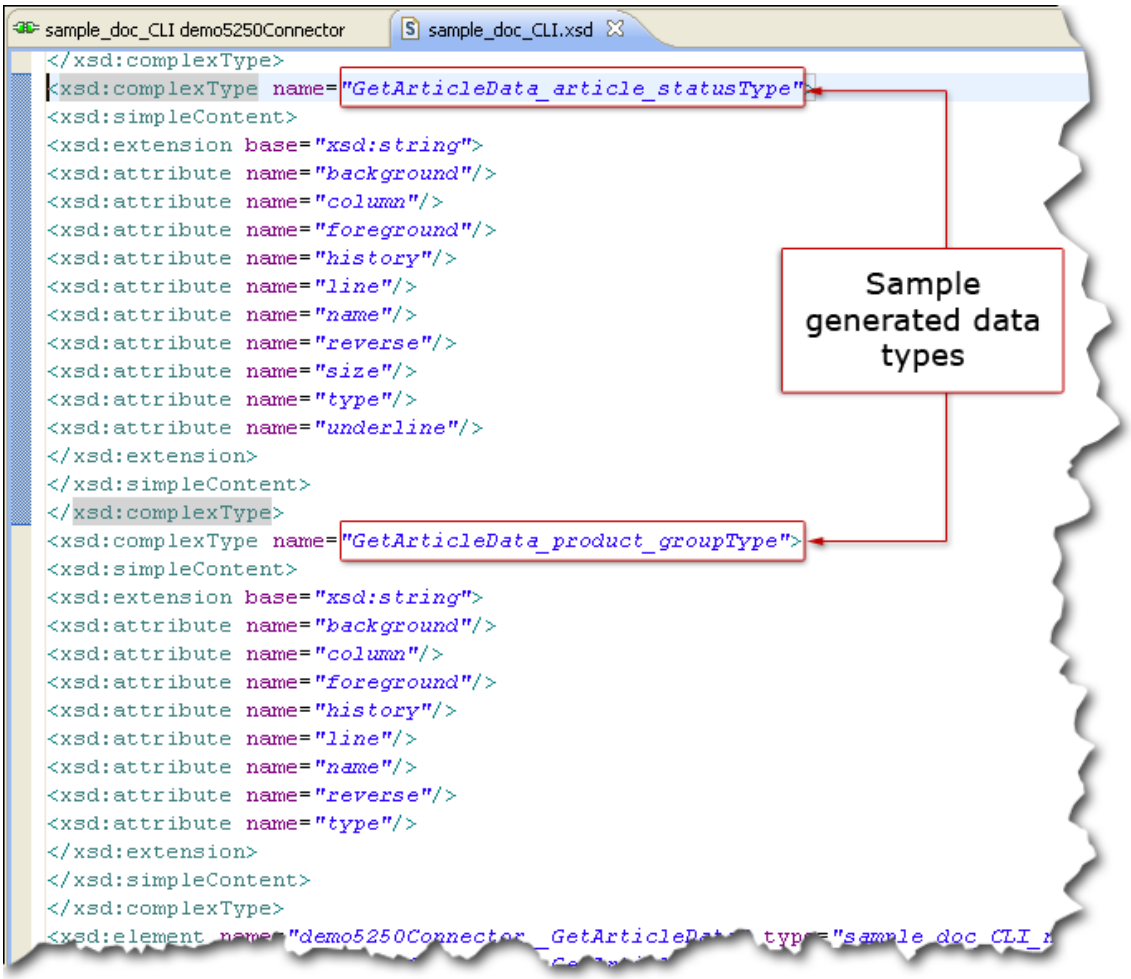


Figure 3 - 11: Transaction data types in schema

They correspond to the transaction response XML, obtained through the execution of extraction rules defined in the project: `ArticlesTable`, `Tag artile_rsp`, `Tag article_status`, `Tag product_group`. Data types corresponding to these extraction rules appear in the **Editor** in the following format:

*<transactionName>*_*<RuleXMLTagName>*`Type`

Editing the project schema can be usefull when, for example, specific restrictions are to be set on some elements.

The `GetArticleData` transaction is now exposed by CLI as a SOAP Web service method of a SOAP Web service (the project).

To display the SOAP Web service WSDL

1. Launch your default Web Browser;
2. In the URL address bar, type in the URL in the following format:

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/projects/
<ProjectName>/<SOAPmode>?wsdl
```

For example, in our project:

- *`HostName`* = `localhost`;
- *`ConvertigoPortNumber`* = `18080`;
- *`ProjectName`* = `sample_doc_CLI`;
- *`SOAPmode`* = `.ws` for RPC mode, *`SOAPmode`* = `.wsl` for document/litteral mode.

For example, in a browser URL address bar:

Figure 3 - 12: URL syntax for displaying the SOAP Web service WSDL

3 Press **Enter**.

The Web browser displays the SOAP Web service WSDL with:

- the list of methods (in our project, the `GetArticleData` method);
- the method input variable;
- the output format description.

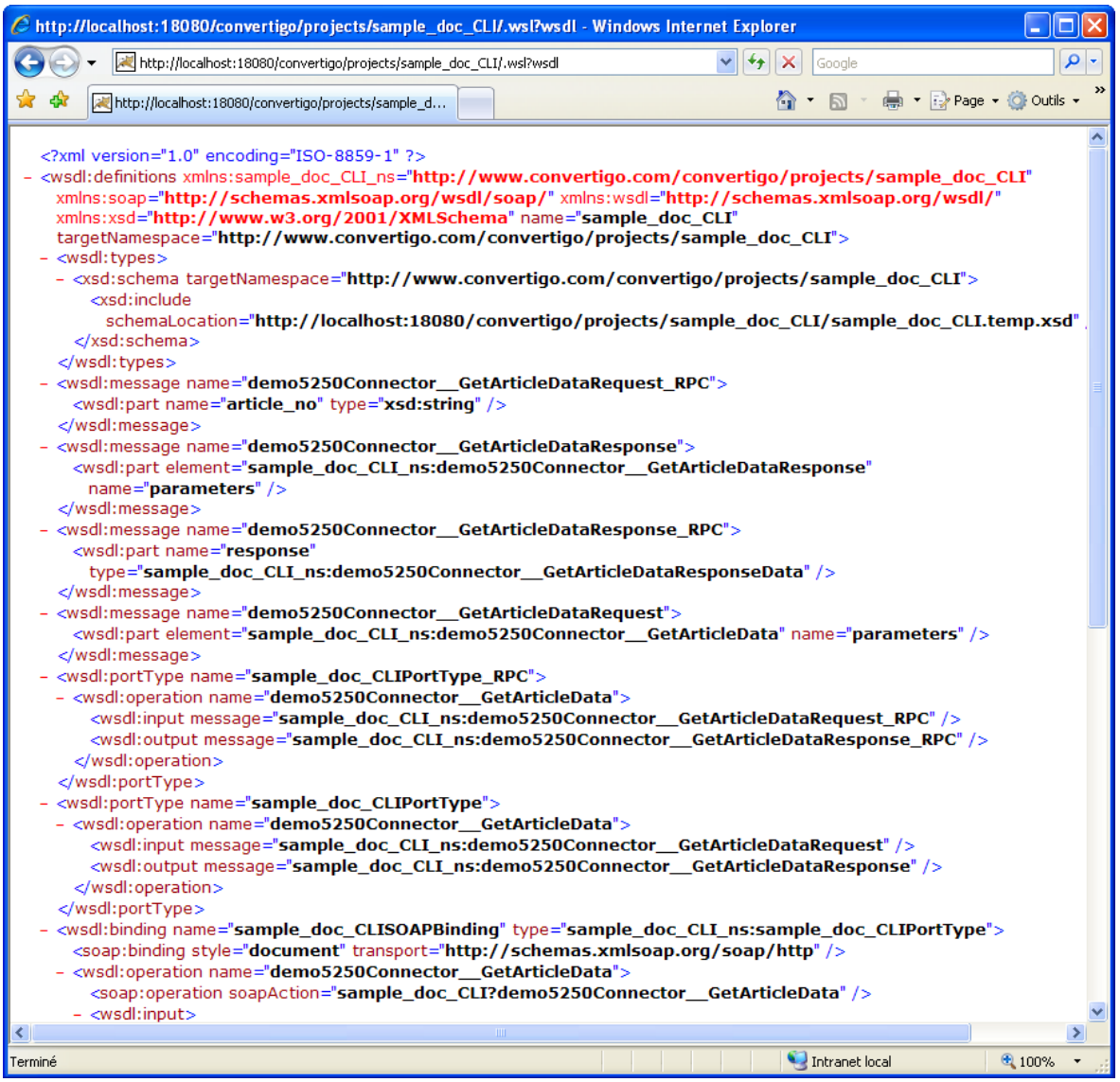


Figure 3 - 13: SOAP Web service WSDL

To invoke a transaction as a SOAP Web service method

Eclipse provides a tool named **Web Services Explorer** which is used, among other features, to invoke SOAP Web services without writing SOAP envelopes. We will launch it to invoke our newly created SOAP Web service.

1 Click on ***Run > Launch the Web Services Explorer*** or click on the icon in the studio toolbar:

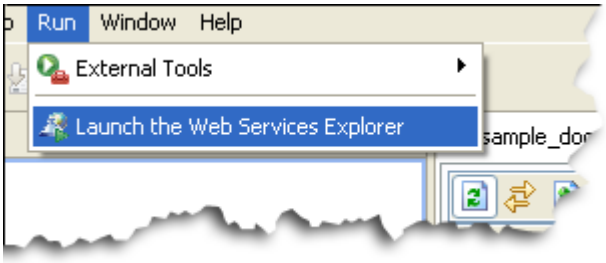


Figure 3 - 14: Launching the Web Services Explorer

The **Web Services Explorer** view opens in front of the **Connector View**:

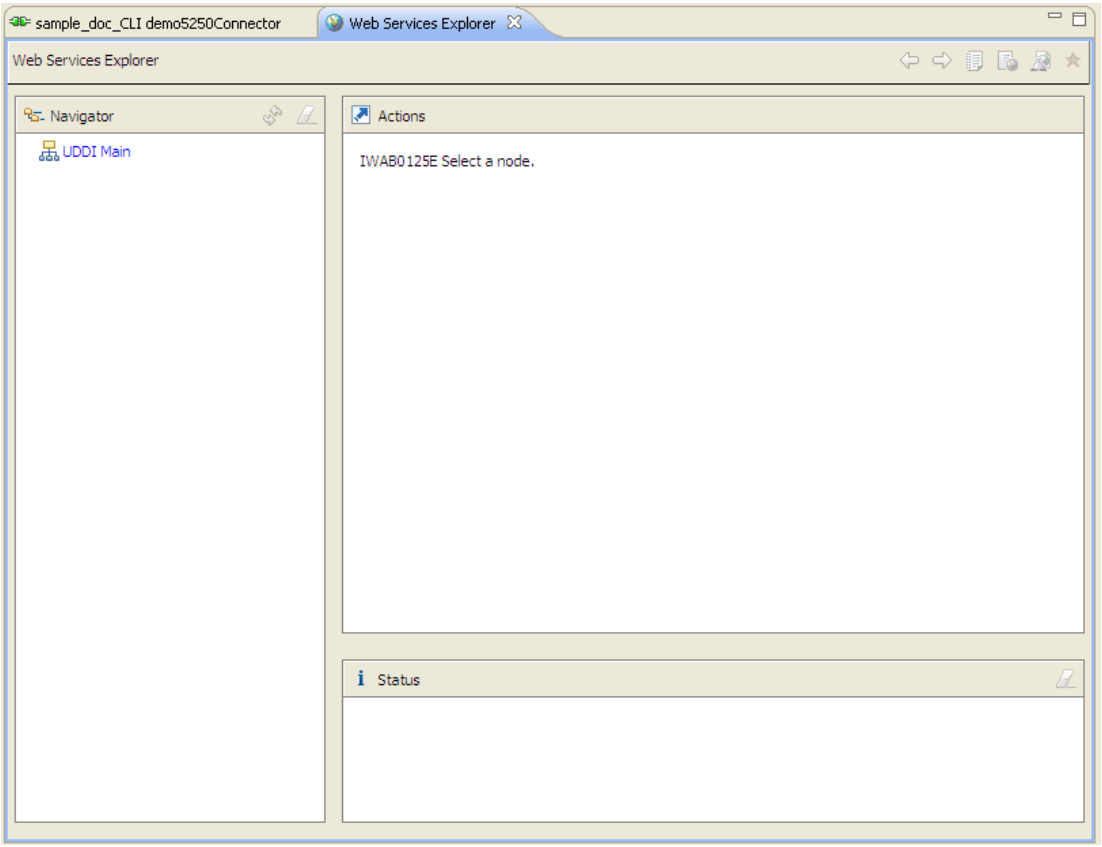


Figure 3 - 15: Web Services Explorer

The **Web Services Explorer** connects to the service WSDL and displays *WSDL binding details* such as*:*

- *methods* (called *Operations* in the interface);
- *endpoints*.

The interface contains three panes and six icons:

Table 3 - 1: Web Services Explorer

Explorer elements		Description
Pane	Navigator	Contains the WSDL tree structure: service URL, methods, WSDL Binding details, etc.
	Actions	Contains, in the form of links and buttons, actions to be performed on methods, endpoints, services, etc..
	Status	Displays error messages and other information.
Icons		*Back* icon. Navigates back one page in the Explorer.
		Forward icon. Navigates forward one page in the Explorer.
Icons		*UDDI Page* icon. Displays the **UDDI Main** tree in the **Navigator** and related actions ion UDDI registries n the **Actions** pane.
		WSIL Page icon. Displays the **WSIL Main** tree in the **Navigator** and related actions on WSILs in the **Actions** pane.
		WSDL Page icon. Displays the **UDDI Main** tree in the **Navigator** and related actions on WSDL services in the **Actions** pane.
		Favorites icon. Displays favorites UDDI registries, WSILs and WSDL services.

2 Click the **WSDL Page** icon in the **Web Services Explorer** toolbar.

3 In the **Navigator** pane of the **Web Services Explorer**, click on **WSDL Main** link.

The **Actions** pane of the **Web Services Explorer** is automatically updated:

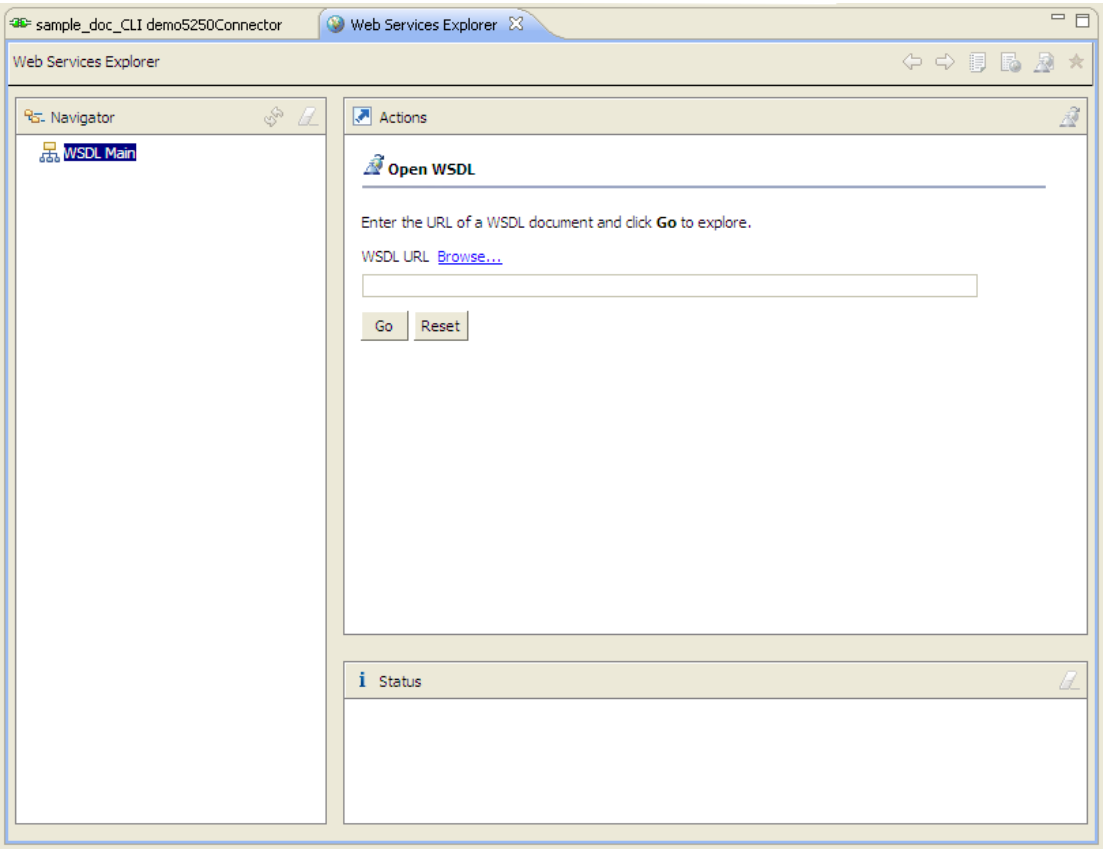


Figure 3 - 16: Web Services Explorer

4 In the **WSDL URL** of this pane, enter the URL of the SOAP service in the following format:

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/projects/
<ProjectName>/<SOAPmode>?wsdl
```

For example, in our project:

- *HostName* = `localhost`;
- *ConvertigoPortNumber* = `18080`;
- *ProjectName* = `sample_doc_CLI`
- *SOAPmode* = `.ws` for RPC mode, *SOAPmode* = `.wsl` for document/litteral mode.

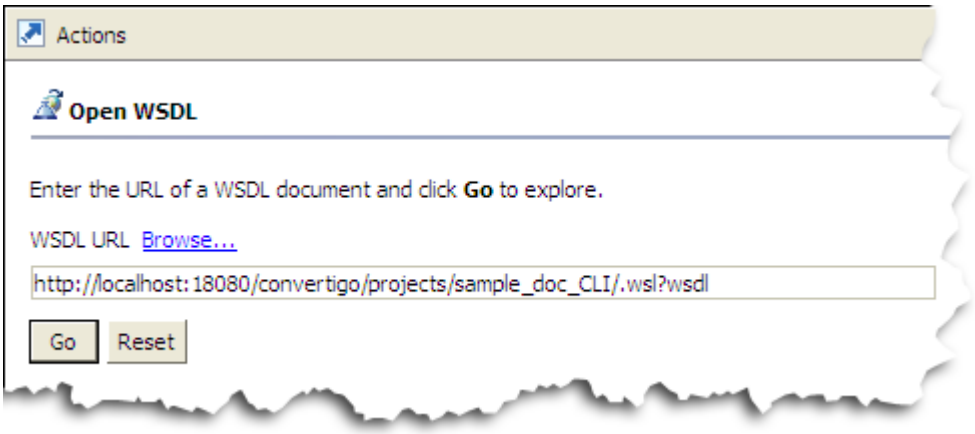


Figure 3 - 17: URL of the SOAP service in the Web Services Explorer

5 Click on **Go**. The three panes are automatically updated:

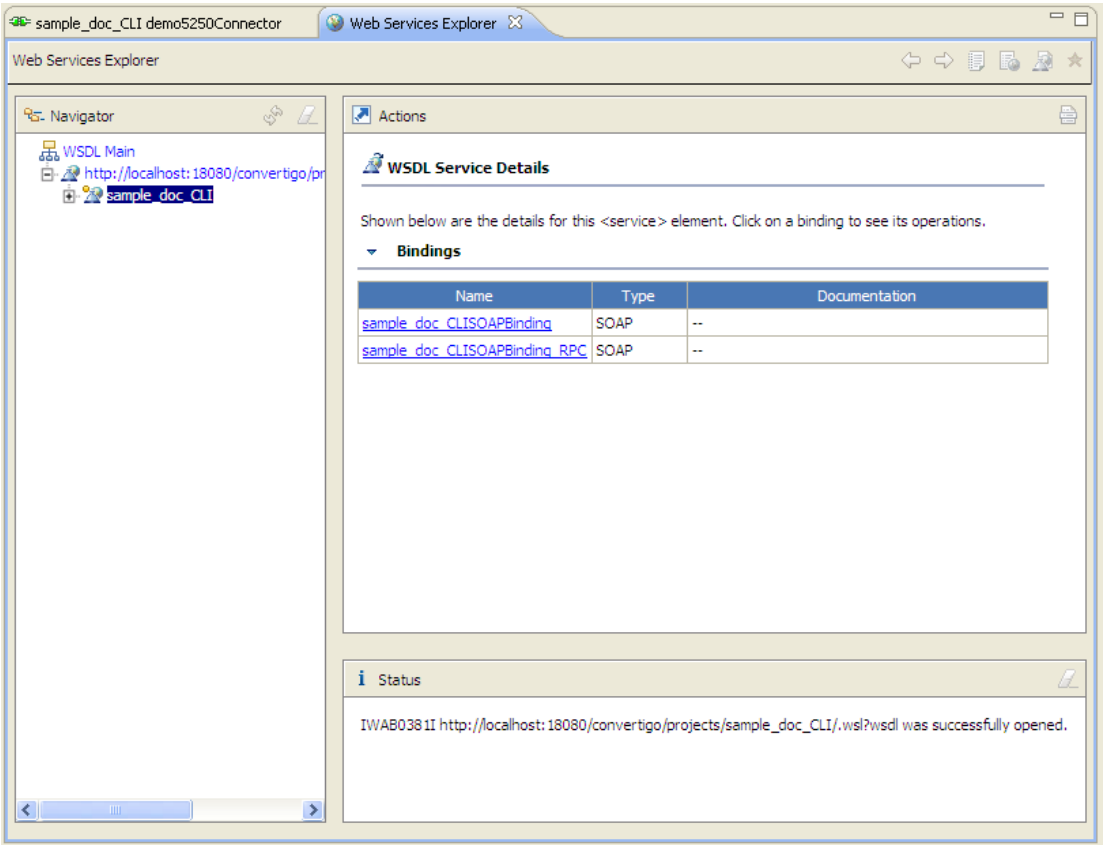


Figure 3 - 18: Web Services Explorer panes after a WSDL URL is reached

6 Expand `sample_doc_CLI` in the **Navigator** pane, then click on `sample_doc_CLISOAPBinding` link. The **Actions** panes is updated:

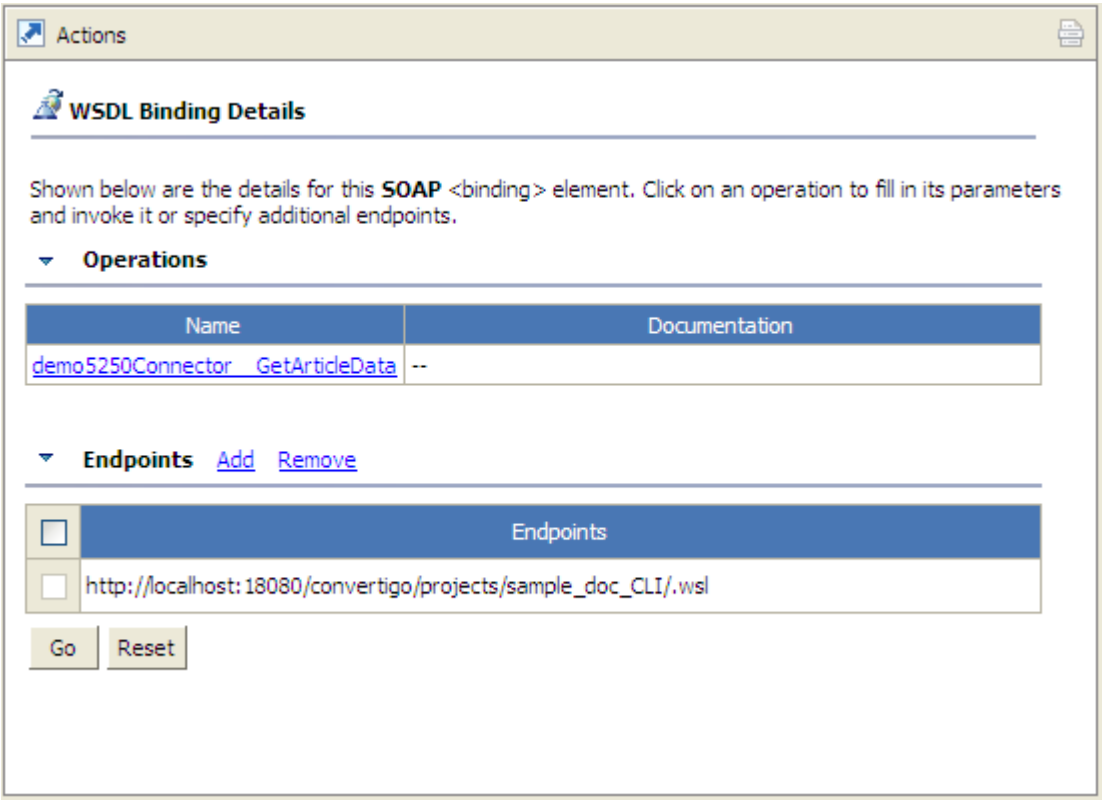


Figure 3 - 19: WSDL Binding Details in Actions pane of the Web Services Explorer

The **Operations** table of the **Actions** pane displays all available *methods* (corresponding to transactions) for the invoked service (`GetArticleData`):

Figure 3 - 20: List of methods displayed in the Web Services Explorer

The **Navigator** pane displays the elements of the WSDL service in a tree structure:

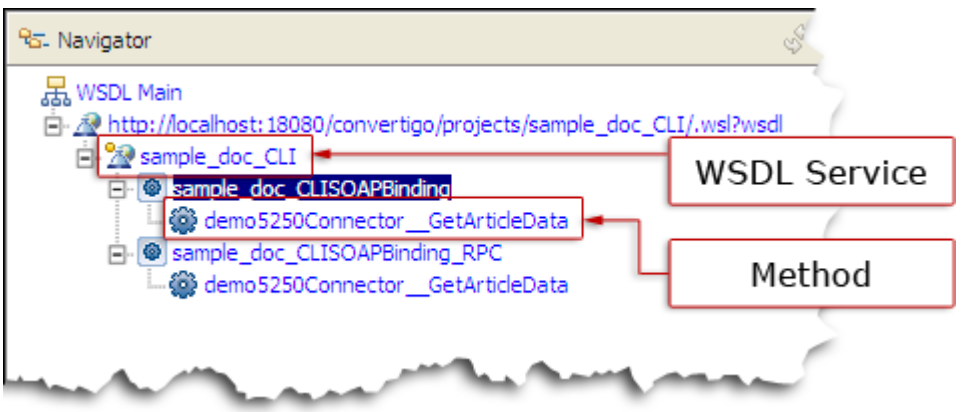


Figure 3 - 21: Web service elements displayed in the Navigator pane

7 Click on `demo5250Connector__GetArticleData` in the **Operations** table. The

Actions pane is automatically updated:

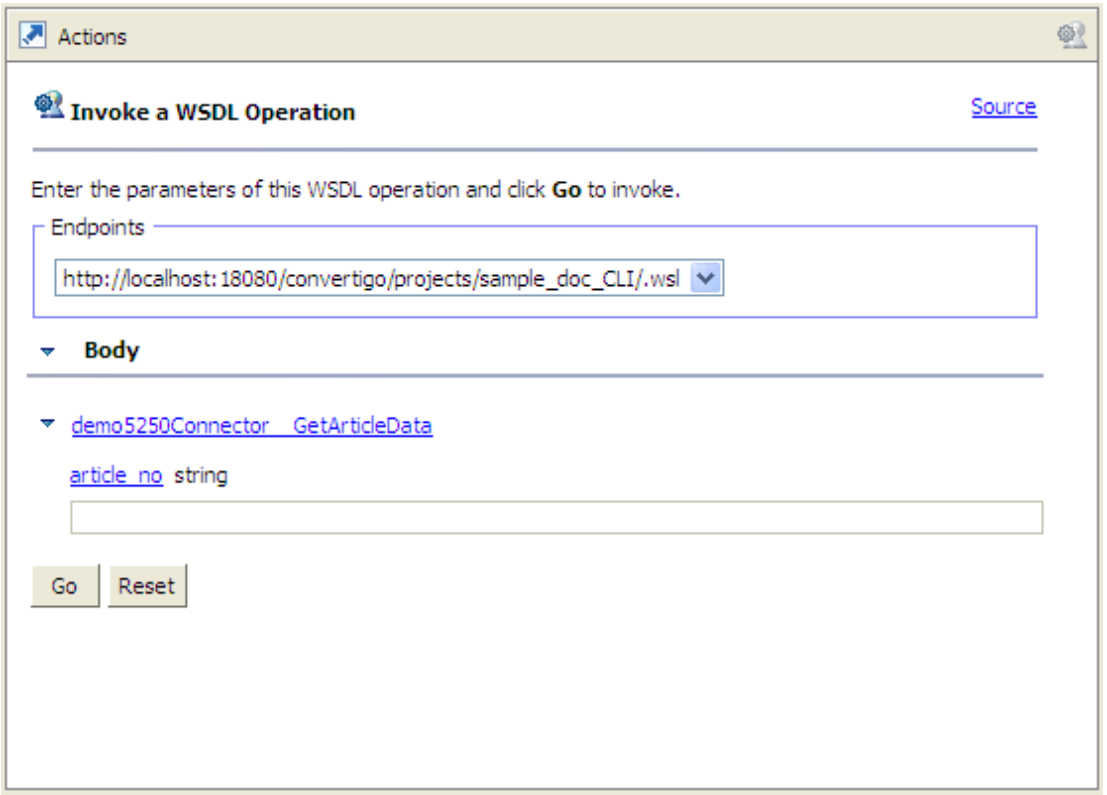


Figure 3 - 22: Invoke a web service method view in the Actions pane

The method input variable as well as its type are displayed (here, `article_no`, variable of `string` type).

8 Type in an article number in the **article_no** field: for example `123456789`.

9 Click on **Go**:

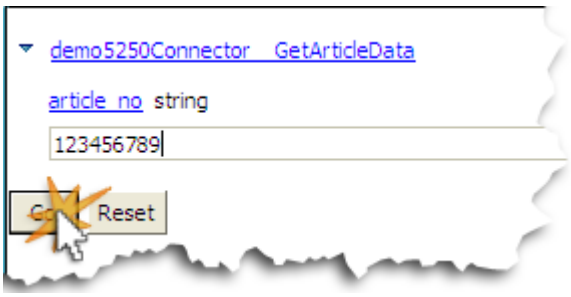


Figure 3 - 23: Invoking a web service method

The **Status** pane is automatically updated. A **source** link appears on the upper right corner of the pane.

10 Click on the **source** link:

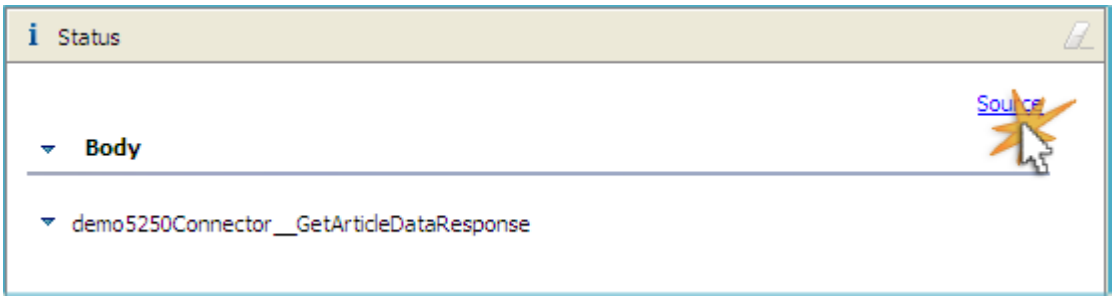


Figure 3 - 24: Status pane after web service method invocation

The **Status** pane diplays :

- the SOAP *request* envelope `GetArticleDataRequest`, which was generated using the input variable entered as article number:

Figure 3 - 25: SOAP Request envelope detail

- the SOAP *response* envelope `GetArticleDataResponse`, which corresponds to the WSDL description previously generated. The data included in the response envelope is similar to the data obtained when invoking a REST service. The difference is that it is embedded in a SOAP envelope:

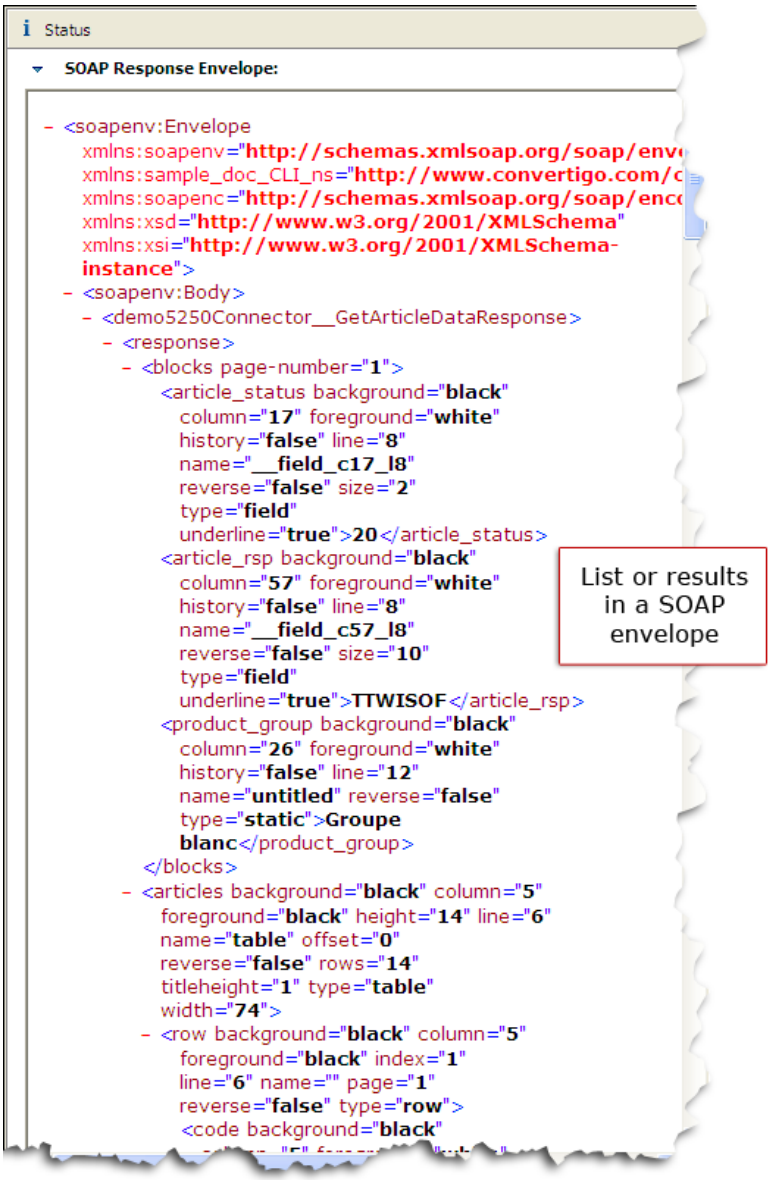


Figure 3 - 26: SOAP response envelope detail

3.2 XSL AJAX Presentation

Convertigo features a *test platform* for calling projects, visualizing project-related transactions and testing them.

This platform is accessed using any standard Web browser.

To see how to launch the test platform, see "Executing a Transaction using the Test Platform" on page 2-75.

The test platform includes an AJAX library (JavaScript) for calling transactions and presenting them using XSL (for *eXtensible Stylesheet Language*) style sheets.

XSL style sheets process XML nodes of extracted XML data to transform them into an HTML document. The XSL *transformation* is performed by the AJAX library included in the project.

The basic process is the following:

1. *Creating* an XSL style sheet that processes our transaction XML output. This is done in two steps: first creation of a CLI **Sheet** object, then creation and edition of the related `.xsl` file;
2. *Associating* the previously created style sheet with the transaction for CLI to know that, on call of a transaction, this XSL style sheet must be returned together with associated XML data.

We will now create the XSL style sheet and associate it to our transaction.

To set up an XSL transformation

1. In the **Projects View**, right-click on the transaction (`GetArticleData`).
2. Select ***New > Sheet***.

   A **New Sheet** wizard opens:

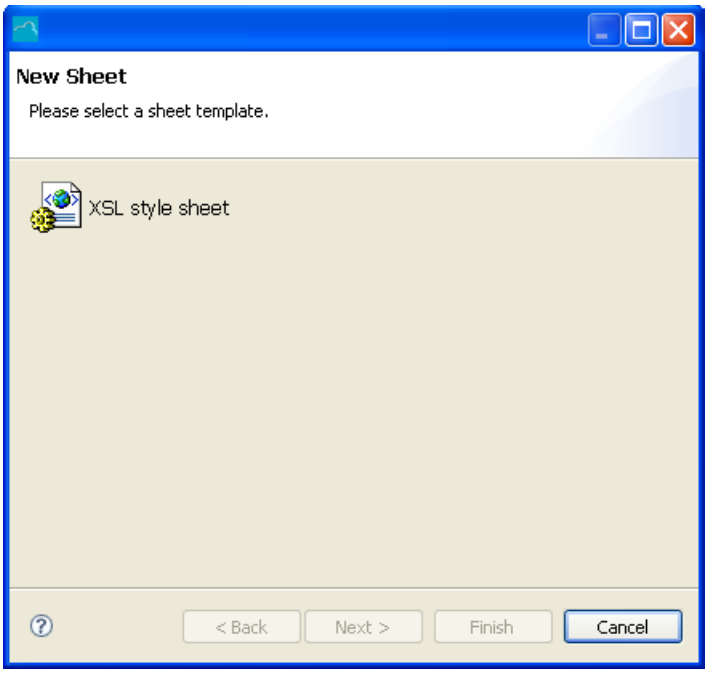


Figure 3 - 27: New Sheet wizard

3. Click on **XSL style sheet**.
4. Click on **Next**:

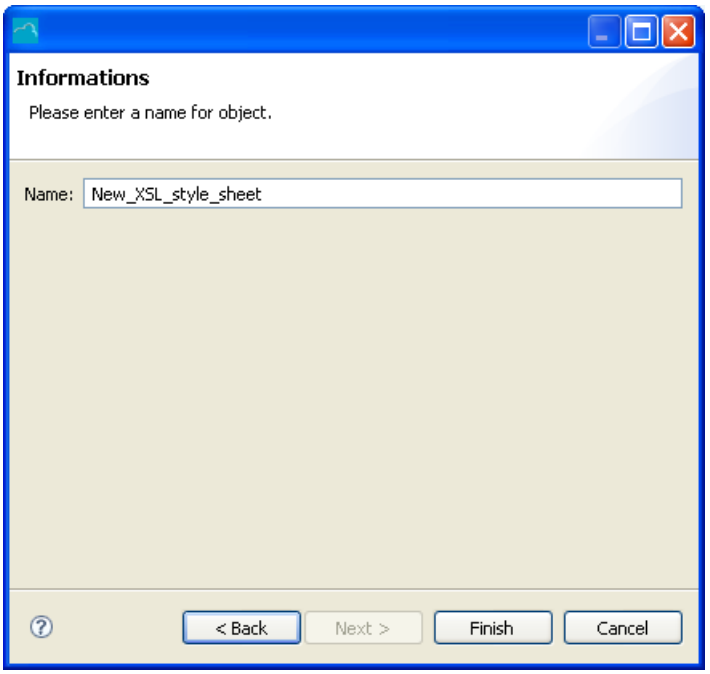


Figure 3 - 28: Entering a name for the new style sheet

5 Enter a name in the **Name** field (for example `ArticlesList`).

6 Click on **Finish**.

The new style sheet object is created in the **Sheets** folder of the transaction:

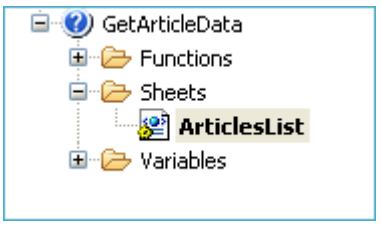


Figure 3 - 29: New style sheet object in the Projects View

Now that the Convertigo sheet object has been created, we will associate it with a physical file, `articlesList.xsl`, that will be edited for presenting data as required.

7 In the **Projects View**, click on the **Project Explorer** tab.

8 Right-click on the project (for example `sample_doc_CLI`).

9 Select ***New > File***.

A **New File** wizard opens:

Figure 3 - 30: New File wizard

10 In the **File name** field, enter a filename (`articlesList.xsl` for example).

11 Click on **Finish**.

The `articlesList.xsl` file is:

- created in the **Project Explorer** tree structure:

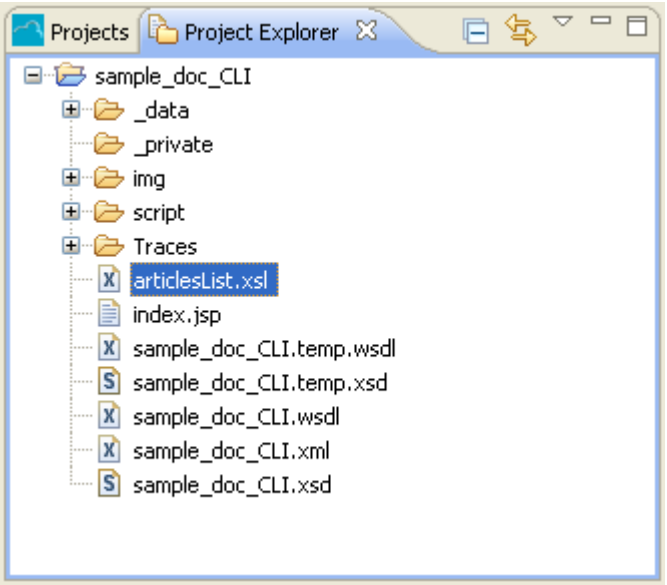


Figure 3 - 31: New file in Project Explorer

- displayed in the **Editor** (here, after the **Source** tab has been selected):

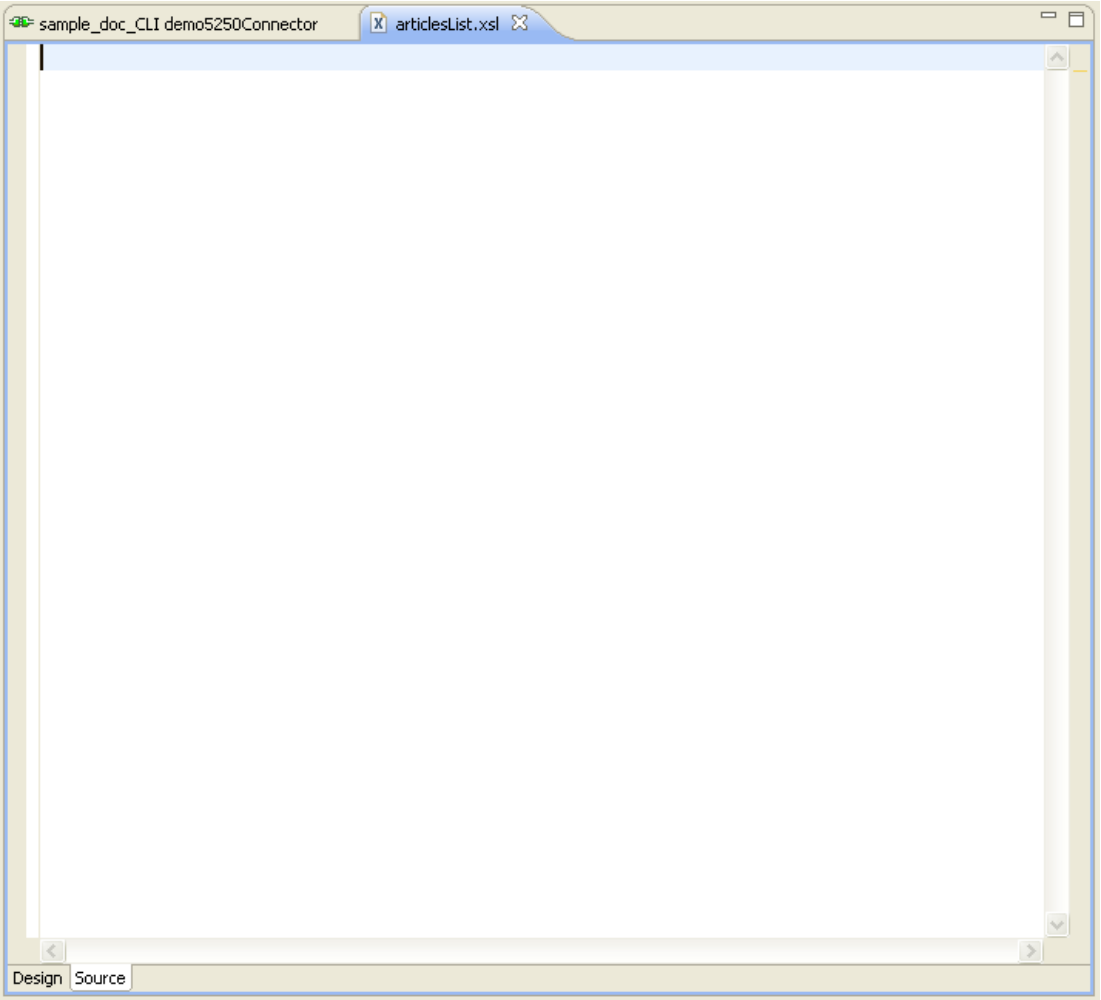


Figure 3 - 32: XSL style sheet

After edition, the XSL style sheet is divided into *template rules* that match parts of the XML document to be processed: the document (`document` element) and the list of results (`articles` element).

In our case, it is edited to obtain a document defined as follows:

- the first template rule matches on "/document" node: it defines the HTML code for the Web page title ("Articles list" in <h2> tag name);

```
<xsl:template match="/document">
    <h2>Articles list</h2>
    <xsl:apply-templates/>
</xsl:template>
```

Figure 3 - 33: Template rule on document

- the second template rule matches on "articles" element (the `articles` table of the XML output): it defines the articles list as an HTML table divided into three columns called `Code`, `Name` and `Status`. For each row of the `articles` table, the `code`,

name and `status` values are displayed in previously defined columns.

```
<xsl:template match="articles">
    <table cellspacing="2" cellpadding="3" border="0">
        <tr>
            <td><b>Code</b></td>
            <td><b>Name</b></td>
            <td><b>Status</b></td>
        </tr>

        <xsl:for-each select="row">
            <tr>
                <td><xsl:value-of select="code"/></td>
                <td><xsl:value-of select="name"/></td>
                <td><xsl:value-of select="status"/></td>
            </tr>
        </xsl:for-each>
    </table>
</xsl:template>
```

Figure 3 - 34: Template rule on articles

- the third template rule matches on "article_status", "article_rsp" and "product_group" elements (the data of the XML output corresponding to the search article code in the transaction): it defines that these data won't appear in the HTML page thanks to no content in the template rule.

```
<xsl:template match="article_status | article_rsp | product_group"/>
```

Figure 3 - 35: Template rule on article_status, article_rsp and product_group

For more information on XSL, see the XSLT tutorial from W3Schools at http://www.w3schools.com/xsl/.

Now that the XSL style sheet has been created, we must associate it to the sheet previously created in the transaction, by setting the **URL** sheet property.

12 In the **Projects View**, click on the sheet (`ArticlesList`).

The **Properties View** is automatically updated:

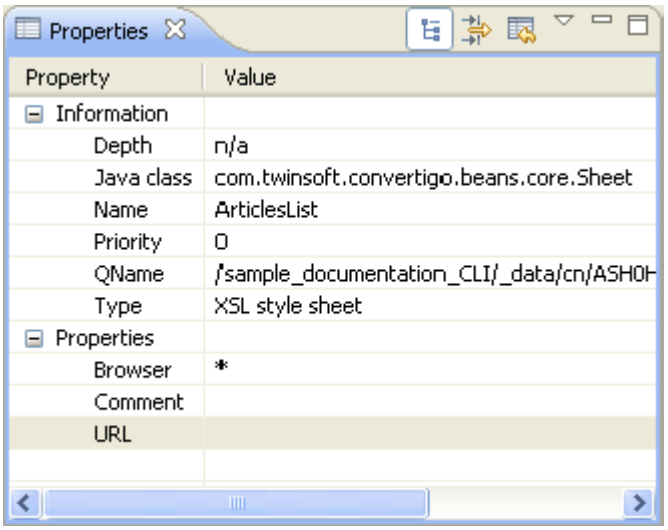


Figure 3 - 36: Sheet properties

13 Click on the value of the **URL** property. The value is highlighted.

14 Type in the name of the previously created XSL style sheet file: `articlesList.xsl` for example.

15 Press `Enter`.

16 Save the project by clicking on or by pressing `Ctrl + S`.

The sheet object is now created. The next step consists in associating the sheet with the transaction execution, by setting the transaction **Style Sheet** property.

This property specifies whether or not and which style sheet must be used to perform the XSL transformation after the transaction execution. The property can take on three values:

Table 3 - 2: Style Sheet transaction property

Style Sheet value	Description
None	Does not process with XSLT. Usually set for Web services (SOAP or REST), as plain XML data is then to be returned.
From transaction	Uses the XSL style sheet attached to this transaction.
From last detected screen class	Uses XSL style sheet attached to the last detected screen class.

17 In the **Projects View**, click on the transaction (`GetArticleData`).

The **Properties View** is automatically updated:

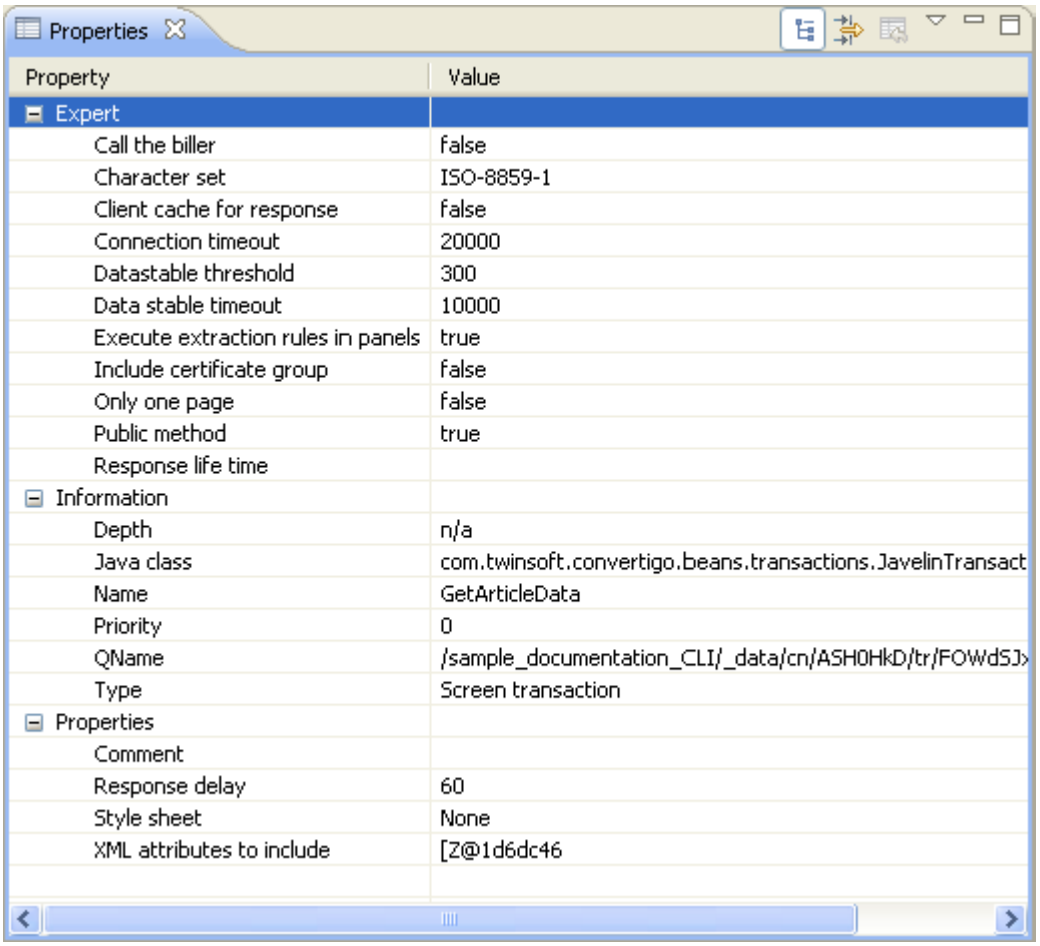


Figure 3 - 37: Transaction properties

18 Click on the value of the **Style Sheet** property. A drop-down menu appears:

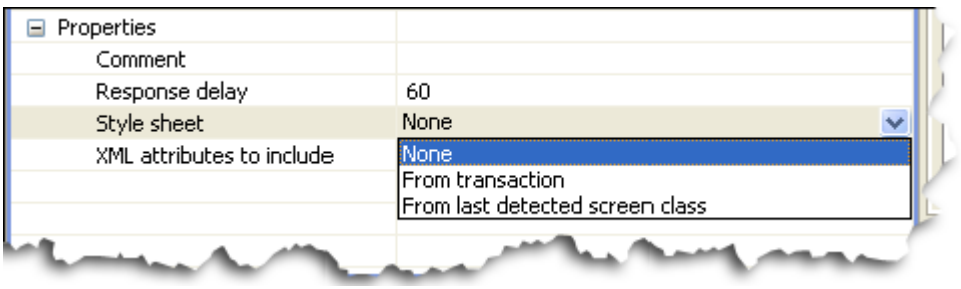


Figure 3 - 38: Transaction Style Sheet property

19 Select **From transaction**.

20 Press `Enter`.

21 Save the project by clicking on or by pressing `Ctrl + S`.

We will switch back to the test platform and invoke the transaction to see how XML data was transformed.

22 Follow instructions of procedure *"Executing a Transaction using the Test Platform"* on page 2 - 75.

The XML output generated by the transaction (see Figure 2 - 133) is processed by the

defined XSL style sheet. It appears in the **Execution result** window as an HTML document:

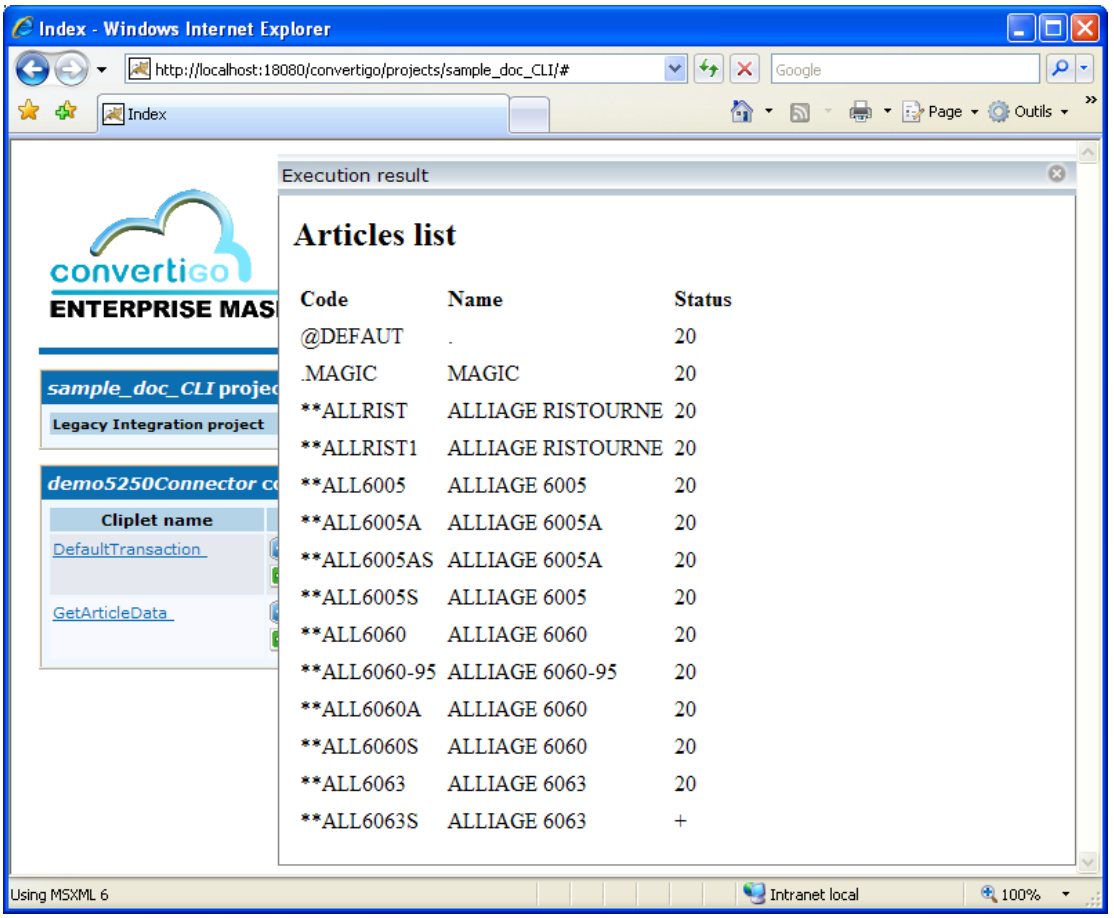


Figure 3 - 39: XML output after XSL transformation