convertigo
MASHUP YOUR BUSINESS

Starting with Convertigo Mashup Composer

Quick Guide

CEMS 5.3.1

TABLE OF CONTENTS

Getting Familiar with Convertigo Mashup Composer 1

CMC Purpose 1-1

CMC Concepts and Objects 1-1

General Concept 1-1

Basic Principle 1-2

Mashup Composer elements linked to Convertigo Objects 1-3

Implementing a CMC project 1-5

CMC Elements 1-6

Definitions 1-6

Widget 1-6

Action 1-7

Event 1-8

Interaction 1-9

Mapping function 1-9

Page 1-10

Convertigo Studio 1-10

General Presentation 1-10

View Description 1-11

Projects view 1-12

Properties view 1-14

Connector view 1-18

Mashup Composer 1-19

My First Convertigo Mashup Composer Project ... 2

Prerequisites - Key Quick Guide Projects 2-1

Mashup Composer Project Description 2-2

Setting Up the Convertigo Part of the Project - Creating Mashup Elements 2-4

- Creating projects in Convertigo Studio 2-5
- Creating all needed actions in CWC project 2-5
- Creating needed event in CLI project 2-6
- Creating interaction hub project 2-13
- Deploying all projects to Convertigo Server 2-23

Setting Up the Mashup Composer Part of the Project - Developing the Composite Application 2-26

- Switching to the Mashup Composer 2-27
- Declaring Widgets in Mashup Composer 2-29
  - GoogleClip widget 2-30
  - ListOfArticles widget 2-42
  - InteractionHub widget 2-43
- Creating a Page Including Three Widgets 2-45

Finalizing and Testing the Composite Application 2-61

- Finalizing widgets 2-62
  - Finalize ListOfArticles widget 2-62
  - Finalize InteractionHub widget 2-75
- Testing the Composite Application 2-83

LIST OF FIGURES

Figure 1 - 1: Widget principle 1-2

Figure 1 - 2: Widget with additional actions and events 1-2

Figure 1 - 3: Widgets interaction 1-3

Figure 1 - 4: Widget with actions based on transactions (or sequences), and events based on extraction rules or manually added 1-3

Figure 1 - 5: Widgets interacting directly 1-4

Figure 1 - 6: Widgets interacting through a business logic 1-5

Figure 1 - 7: GoogleClip widget 1-7

Figure 1 - 8: Action performed in widget 1-8

Figure 1 - 9: Convertigo Studio (displaying a web application in HTML connector) 1-11

Figure 1 - 10: Formatting of unsaved, inherited and disabled objects in the Projects view 1-13

Figure 1 - 11: Properties view 1-14

Figure 1 - 12: Text property field edition 1-15

Figure 1 - 13: Java expression field edition 1-15

Figure 1 - 14: Text property edition via edition window 1-15

Figure 1 - 15: Javascript expression window 1-16

Figure 1 - 16: Edition window 1-16

Figure 1 - 17: Configuration window: Attributes editor 1-17

Figure 1 - 18: Configuration window: Mashup event editor 1-17

Figure 1 - 19: Property edition via drop-down menu 1-18

Figure 1 - 20: Selecting a value in the drop-down menu 1-18

Figure 1 - 21: Style sheet in XSL format opened in Document Editor in the Connector view ... 1-19

Figure 2 - 1: Widget interaction ... 2-3

Figure 2 - 2: Transactions in sample_documentation_CWC project ... 2-6

Figure 2 - 3: Displaying GetArticlesData transaction's result from sample_documentation_CLI project ... 2-7

Figure 2 - 4: Selecting the transaction-related style sheet ... 2-8

Figure 2 - 5: Style sheet properties ... 2-8

Figure 2 - 6: CLI project file tree ... 2-9

Figure 2 - 7: articlesList.xsl file in editor ... 2-10

Figure 2 - 8: Edited articlesList.xsl ... 2-12

Figure 2 - 9: Launching the New Project wizard ... 2-13

Figure 2 - 10: New Project Wizard ... 2-14

Figure 2 - 11: Selecting the Mashup Composition sample project ... 2-14

Figure 2 - 12: Sample project in Projects view ... 2-15

Figure 2 - 13: Launching the New Project wizard ... 2-15

Figure 2 - 14: New Project Wizard ... 2-16

Figure 2 - 15: Selecting a project type ... 2-16

Figure 2 - 16: Entering a project name ... 2-17

Figure 2 - 17: sample_doc_CMC project appearing in the Projects view ... 2-17

Figure 2 - 18: Connector view displaying CMC Void connector ... 2-18

Figure 2 - 19: CMC project file tree ... 2-18

Figure 2 - 20: default_hub.xml file in editor ... 2-19

Figure 2 - 21: Editing source widget name in the mapping configuration file ... 2-20

Figure 2 - 22: Editing triggered event name in the mapping configuration file ... 2-20

Figure 2 - 23: Editing target widget name in the mapping configuration file ... 2-21

Figure 2 - 24: Removing unused JavaScript code ... 2-21

Figure 2 - 25: Mapping action name to __transaction variable ... 2-22

Figure 2 - 26: Mapping keyword variable ... 2-22

Figure 2 - 27: Finished mapping configuration file ... 2-23

Figure 2 - 28: Deploying a Convertigo project to a Convertigo Server ... 2-24

Figure 2 - 29: Entering a Convertigo Server address 2-25
Figure 2 - 30: Entering the Convertigo Server login 2-25
Figure 2 - 31: Progression status bar 2-26
Figure 2 - 32: Confirmation message of deployed project 2-26
Figure 2 - 33: Mashup Composer URL in a Web browser 2-27
Figure 2 - 34: Mashup Composer Home page 2-28
Figure 2 - 35: Logging in to the Mashup Composer as administrator 2-28
Figure 2 - 36: Toolbar available while logged in 2-29
Figure 2 - 37: Convertigo Server test platform URL 2-31
Figure 2 - 38: Test platform page for sample_documentation_CWC project 2-31
Figure 2 - 39: Test platform left menu 2-32
Figure 2 - 40: Test platform part showing project's details 2-32
Figure 2 - 41: Test platform part showing project's transactions 2-33
Figure 2 - 42: Test platform part displaying transaction's result 2-34
Figure 2 - 43: Declaring a widget from the GoGoogleCom transaction 2-35
Figure 2 - 44: Copying GoGoogleCom gagdet XML 2-36
Figure 2 - 45: Opening the Application Registry administration page 2-37
Figure 2 - 46: Application Registry administration page 2-37
Figure 2 - 47: Accessing Gadget registry 2-38
Figure 2 - 48: Creating a new gadget 2-39
Figure 2 - 49: Pasting gadget content 2-39
Figure 2 - 50: Setting gadget (widget) name 2-40
Figure 2 - 51: Gadget Details summary 2-40
Figure 2 - 52: Gadget Details summary with available Categories 2-41
Figure 2 - 53: Adding Gadget to Categories 2-41
Figure 2 - 54: GetArticleData transaction in test platform 2-42
Figure 2 - 55: Declaring a widget from the GetArticleData transaction 2-43
Figure 2 - 56: Declaring a widget from the DefaultHub transaction 2-44
Figure 2 - 57: Navigating to site's Home page 2-45

Figure 2 - 58: Adding a new page 2-46
Figure 2 - 59: Page creation wizard 2-46
Figure 2 - 60: Filling page name and display name 2-47
Figure 2 - 61: Selecting empty layout for the page 2-47
Figure 2 - 62: Clicking on finish button 2-48
Figure 2 - 63: Newly created empty page 2-48
Figure 2 - 64: Opening Page Editor 2-49
Figure 2 - 65: Page Editor 2-49
Figure 2 - 66: Finding widgets in Gadgets category 2-50
Figure 2 - 67: Adding GoogleClip widget to the page 2-51
Figure 2 - 68: GoogleClip widget in the page 2-51
Figure 2 - 69: Adding interactionHub widget to the page 2-52
Figure 2 - 70: All widgets added to the page 2-53
Figure 2 - 71: Selecting Containers tab in Page Editor toolbar 2-54
Figure 2 - 72: Available containers for Mixed Layout category 2-55
Figure 2 - 73: Adding One Row, Two Columns cfontainer to the page 2-56
Figure 2 - 74: Container added in the page 2-57
Figure 2 - 75: Moving InteractionHub widget in the layout 2-58
Figure 2 - 76: Moving GoogleClip widget in the layout 2-59
Figure 2 - 77: All widgets laid out in the page 2-60
Figure 2 - 78: Clicking on finish button 2-60
Figure 2 - 79: Article data mashup page 2-61
Figure 2 - 80: Convertigo Server Administration Console URL 2-63
Figure 2 - 81: Login page of Server Administration Console 2-63
Figure 2 - 82: Typing in user ID and password 2-64
Figure 2 - 83: Administration Console Home page 2-64
Figure 2 - 84: Administration Console left menu 2-65
Figure 2 - 85: Status widget in Administration Console 2-65
Figure 2 - 86: System Information widget in Administration Console 2-66

Figure 2 - 87: Monitor widget in Administration Console 2-67
Figure 2 - 88: Clicking on Trace player button 2-68
Figure 2 - 89: Trace player widget in Administration Console 2-68
Figure 2 - 90: Available traces in Trace player widget 2-69
Figure 2 - 91: Setting trace running in Trace player widget 2-69
Figure 2 - 92: Trace added confirmation message 2-70
Figure 2 - 93: Trace appearing in Trace player widget 2-70
Figure 2 - 94: Trace running in Trace player widget 2-70
Figure 2 - 95: Clicking on Projects button 2-71
Figure 2 - 96: Project list widget in Administration Console 2-71
Figure 2 - 97: Accessing sample_documentation_CLI project's edition 2-72
Figure 2 - 98: Project sample_documentation_CLI edition widget 2-72
Figure 2 - 99: Displaying sample_documentation_CLI project's connector properties 2-73
Figure 2 - 100: Updating connector's Connection address property 2-74
Figure 2 - 101: Project updated confirmation message 2-74
Figure 2 - 102: ListOfArticles widget displaying the list of articles from the legacy application 2-75
Figure 2 - 103: Page Editor opened on Article data mashup page 2-76
Figure 2 - 104: InteractionHub widget's toolbar 2-77
Figure 2 - 105: InteractionHub widget's settings 2-77
Figure 2 - 106: Edited InteractionHub widget's settings 2-78
Figure 2 - 107: Clicking on finish button 2-78
Figure 2 - 108: Hidden InteractionHub widget 2-79
Figure 2 - 109: CMC project file tree 2-80
Figure 2 - 110: default_hub.html file in editor 2-80
Figure 2 - 111: Edited default-hub.html 2-81
Figure 2 - 112: Images folder in sample_doc_CMC project 2-82
Figure 2 - 113: InteractionHub widget displaying image 2-83
Figure 2 - 114: Articles data mashup page displaying two widgets in "start" state 2-84
Figure 2 - 115: Articles data mashup page displaying three widgets in "start" state 2-85

Figure 2 - 116: Test result 2-86

LIST OF TABLES

Table 1 - 1: CMC elements 1-6

Table 1 - 2: Interaction elements 1-9

Table 1 - 3: Convertigo Studio views 1-11

Table 1 - 4: Standard Eclipse buttons 1-12

Table 1 - 5: **Projects** view tabs and icons 1-12

Table 1 - 6: Properties View icons 1-14

Table 1 - 7: Tab icons in Connector View 1-18

Table 2 - 1: Sample project widgets - Name and description 2-2

Table 2 - 2: `IMG` element attributes 2-11

Table 2 - 3: Sample project widgets - Transaction / project 2-29

Table 2 - 4: `IMG` element attributes 2-80

1 Getting Familiar with Convertigo Mashup Composer

This chapter presents the purpose, concepts and objects of **Convertigo Mashup Composer** (**CMC**), one of Convertigo Enterprise Mashup Server's components.

It also describes the four views of the Eclipse-based Convertigo Studio and their elements (tabs, icons, menu), as well as the Mashup Composer.

- CMC Purpose
- CMC Concepts and Objects
- Convertigo Studio
- Mashup Composer

1.1 CMC Purpose

The purpose of **Convertigo Mashup Composer** (**CMC**) is to quickly and easily build composite applications from a catalog of *Widgets* (*Gadgets*) including widgets' interactions.

Widgets and interactions between them are created by developers in the Convertigo Studio. Then, they are deployed in the **Convertigo Server** and the **Mashup Composer** to be a part of a global composite application.

1.2 CMC Concepts and Objects

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

- General Concept
- CMC Elements
- Definitions

1.2.1 General Concept

A composite application combines one or several *widgets* linked together through *interactions*.

Basic Principle

Basically, a widget is a graphical element:

- executing *actions* in input,
- triggering *events* in output,
- able to *interact* with one or several other widgets as part of a composite application.

To be used in a composite application, a *widget* must first be created based on a *default action*:

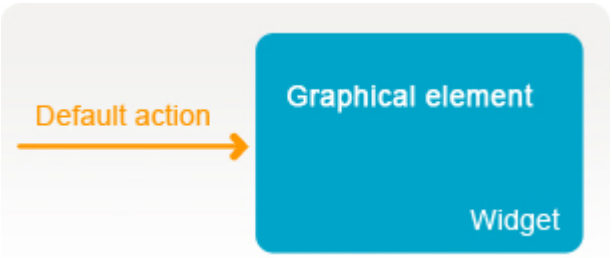

Figure 1 - 1: Widget principle

Other actions and events can be added to the widget:

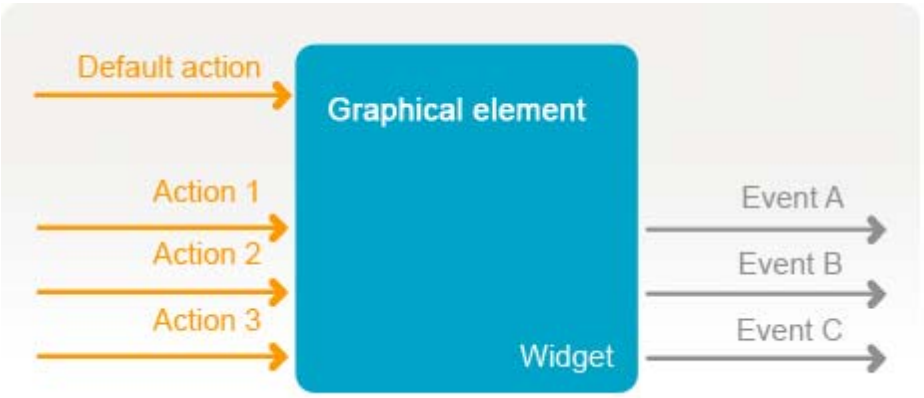

Figure 1 - 2: Widget with additional actions and events

As said in introduction to this section, a composite application combines one or several *widgets* linked together through *interactions*.

In fact, when all widgets needed in a composite application have been created (with their default action) and associated with required actions and events, they are functionally related one to another *in a same page* through *interactions.*

An *interaction* specifies which *actions* are to be performed in a widget `w2` when a given *event* is triggered from a widget `w1`:

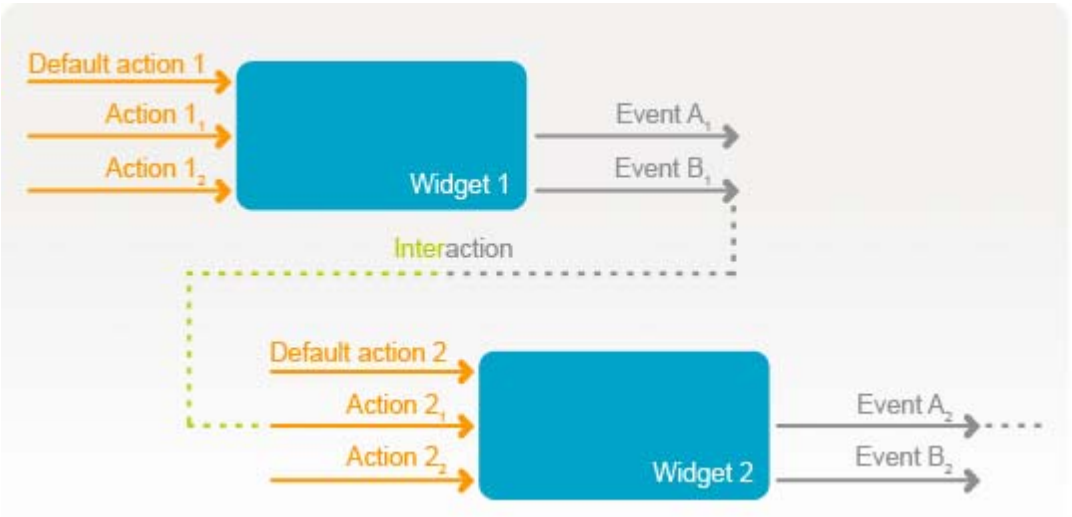

Figure 1 - 3: Widgets interaction

A single *event* can trigger in turn one or several actions, in other widgets or itself, meaning that a widget can be related to one or several other widgets.

Mashup Composer elements linked to Convertigo Objects

Every Mashup Composer element described in previous section is linked to Convertigo concepts or objects.

Actions (widget inputs) are based on existing transactions or sequences, with their variables as parameters (see Figure 1 - 4). They can be any transaction or sequence created using one of Convertigo features (CLI, CLP, CWI, CWC, CMS).

The widget's *default action* is then its *start transaction* (or *sequence*). It will be executed when the widget loads. Initially, the *widget* is deployed in **Mashup Composer** from this transaction.

Events (widget outputs) can be based on defined extraction rules or can be manually added in the widget's HTML display through the XSL style sheet.

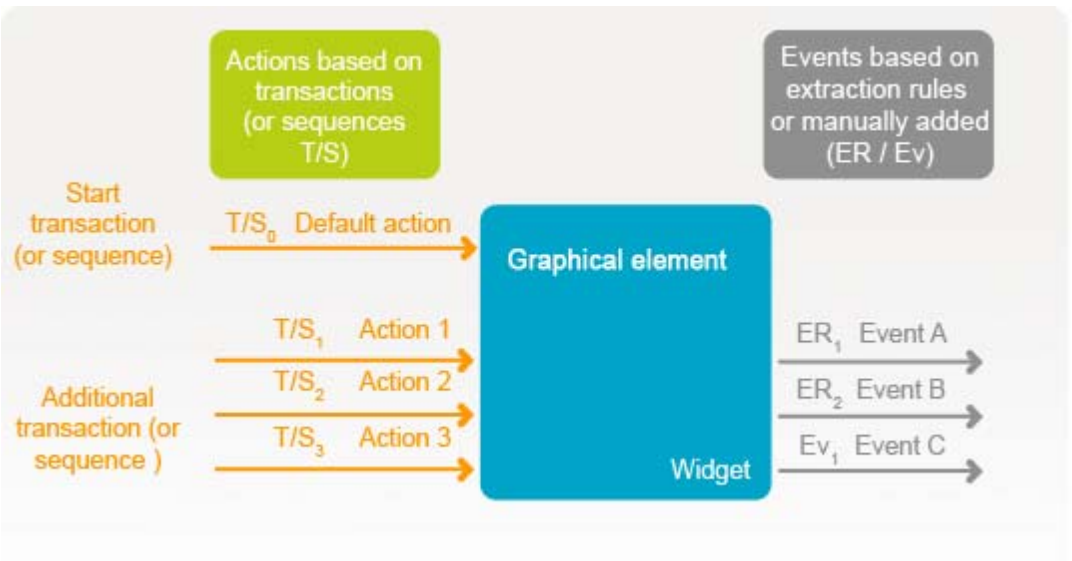

Figure 1 - 4: Widget with actions based on transactions (or sequences), and events based on extraction rules or manually added

Interactions are defined in a dedicated *Interaction hub* project, in an XML configuration file. A *hub widget* is deployed in the **Mashup Composer** from this project.

An event (triggered from widget `w1`) can be related to an action (to be performed in widget `w2` or widget `w3`) by an interaction either:

- *directly*: event and action are ***compatible*** by definition and can be directly linked:

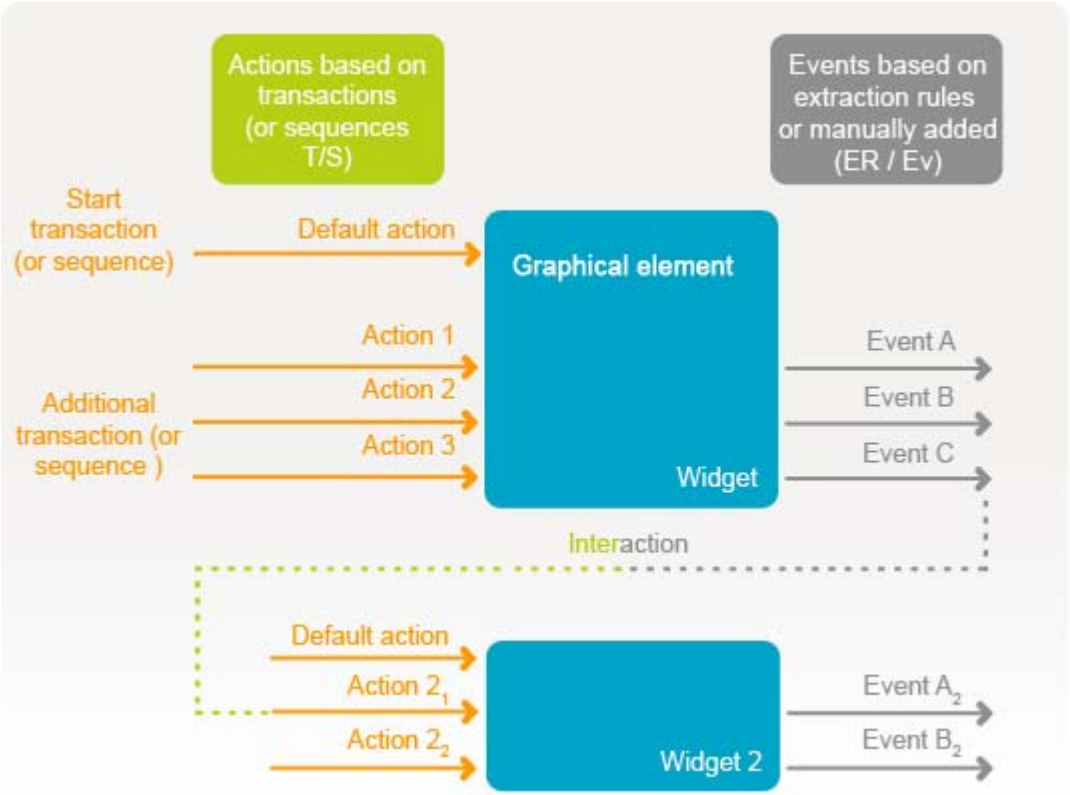

Figure 1 - 5: Widgets interacting directly

- through a *business logic*: event and action need a business logic to be made compatible:

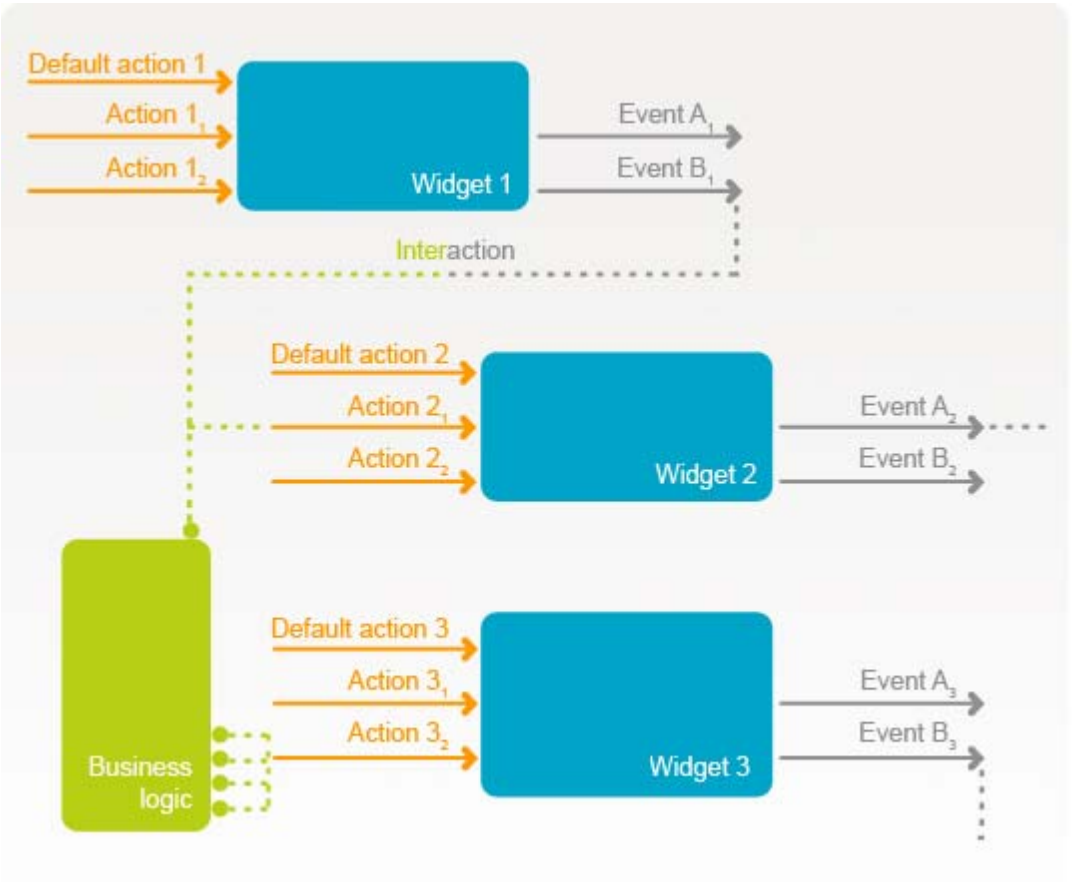

Figure 1 - 6: Widgets interacting through a business logic

Widgets are not initially created to interact specifically one with another. Only when developpers start designing a composite application do they know which interactions will be needed between widgets. The purpose of business logics is to make actions and events ***compatible****, if they are not so by definition, and to help Convertigo developers set up interactions.*

IMPLEMENTING A CMC PROJECT

A CMC project is implemented in four steps:

1 All Convertigo projects needed to create the composite application are created in the **Convertigo Studio** (see *"Convertigo Studio"* on page 1-10), including:

A. projects providing widgets,

- with actions
- and events,

B. and project defining interactions.

2 All projects created to implement widgets and interactions are deployed from the **Convertigo Studio** to the **Convertigo Server**.

3 All widgets needed to create the composite application are declared in the **Mashup Composer** thanks to the projects test-platform.

4 The composite application is developed in the **Mashup Composer**, where all previously declared elements are accessible (see *"Mashup Composer"* on page 1-19),

including:

A. pages creation,

B. widgets add on the pages,

C. and pages' layout.

1.2.2 CMC Elements

The following table describes the previously mentioned CMC elements:

Table 1 - 1: CMC elements

Element	Description
Widget	Defines a portable piece of web-application or web application itself that can be installed and executed within any separate web page by an end user. In CMC, a widget is based upon a default action, which is an existing transaction (or sequence).
Action	Defines an action performed in a widget. In CMC, actions are based upon existing transactions or sequences.
Event	Defines an event triggered from a widget, which usually triggers in turn one or several actions in other widgets as part of an interaction.
Interaction	Defines an interaction between an event and one or several actions involving one or several widgets. An event can trigger an action in the same widget (internal interaction), one other widget or several other widgets.
Page	Defines a page including widgets (with interactions connecting them or not).

Once projects are deployed to the **Convertigo Server** including the **Mashup Composer**, and once widgets are declared from the test-platform into the **Mashup Composer**, they appear in the Mashup Composer and the composite application can be built (see *"Mashup Composer"* on page 1-19).

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

1.2.3 Definitions

WIDGET

A widget defines a portable piece of web-application or web application itself that can be installed and executed within any separate web page by an end user.

As mentioned in *"Basic Principle"* on page 1-2, a widget is basically a graphical element executing *actions* (based on existing transactions or sequences) in input and triggering *events* (based on existing extraction rules or defined through XSL) in output.

A widget:

- is displayed - possibly together with other widgets - in a *page* created using the Mashup Composer,
- displays:
  - when first launched, what results from executing its start transaction (or sequence) ;

that is, its *default action*, or

- what results from executing any other transaction (or sequence), that is any *action* the widget is able to execute, triggered in the context of a given *interaction*.

In the context of this Quick Guide project, a widget called `GoogleClip` is created. Its start transaction is the `GoGoogleCom` transaction from the `sample_documentation_CWC` project set in the context of the *Starting with Convertigo Web Clipper* Quick Guide. The `GoGoogleCom` transaction connects to the Google home page and clips the search form.

When launched, the `GoogleClip` widget therefore displays the clipped Google search form:

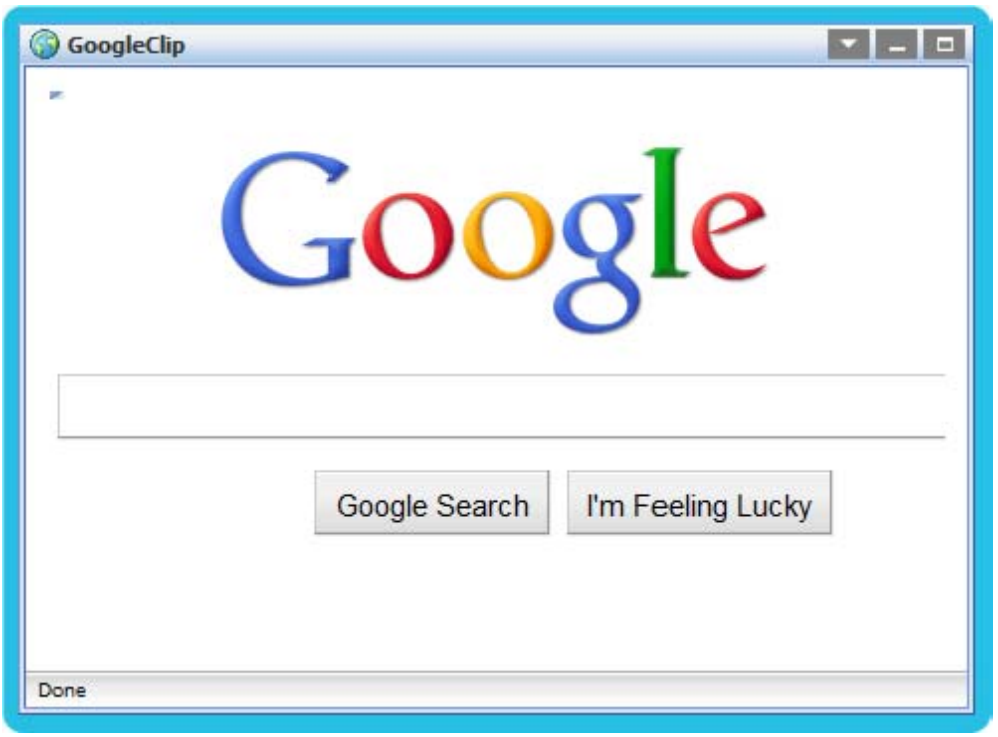

Figure 1 - 7: GoogleClip widget

When created, a widget is always associated with a default action corresponding to its start transaction. When launched, a widget automatically executes its default action and displays its result.

In addition to the default action, a widget is also associated with additional *actions* and *events* and is linked to other widgets through *interactions*.

Widgets are declared in the Mashup Composer using the test-plateform by clicking on the ***Create gadget*** command on the required transaction or sequence (see *"Declaring Widgets in Mashup Composer"* on page 2-29).

ACTION

An action is a widget input. It confers to the widget the ability to perform the action, usually following the triggering of an *event* in the context of a widget *interaction*, and to display a web-based graphical result.

An action is based upon an existing transaction or sequence. It represents the execution of this transaction or sequence in the widget. An action uses as input data the transaction's or

sequence's variable(s).

For example, in the context of this Quick Guide project, an action is associated with the `GoogleClip` widget. It is based on the `SearchGoogleClip` transaction, which launches a keyword-based Google search then clips results.

When performed, this action therefore launches a Google search on a keyword parameter and clips results, which are displayed in the associated widget, `GoogleClip`:

Figure 1 - 8: Action performed in widget

Actions are not declared in the Mashup Composer nor in the widget for now. The actions a widget can execute are all the transactions and sequences existing in the same project as the widget's default action (start transaction or sequence).

Event

An event is a widget output. It confers to the widget the ability to trigger an event, which triggers in turn one or several *actions* in one or several other wigdets in the context of an *interaction*.

An event is bound to an *action* either directly (parameters from the event and those from the action are compatible) or through a *mapping function* (see *"Interaction"* on page 1-9).

An event can be based upon an existing extraction rule proposing this functionality or can be manually added in the HTML display of the widget through the XSL style sheet. In this case, the event is programmatically triggered by calling a specific JavaScript function from Convertigo library.

An event represents the execution of a user action in the widget that leads to launch *actions* in one or several other widgets (in the context of an *interaction*).

It generates as output data a list of parameters that are defined through the extraction rule or through the programmed JavaScript call.

Events are not declared in the Mashup Composer nor in the widget for now. The developper knows the events its widget may trigger.

INTERACTION

An interaction binds a specific widget output (an *event*) to one or several widget inputs (*actions*).

An interaction is modeled in an XML structure that can be translated in the form of a sentence:
On <*source_widget_name*> widget,
When <*event_name*> occurs,
On <*target_widget_name*> widget,
Do <*action_name*> action,
Mapping parameters thanks to <*function_content*> mapping function.

The following table details interaction elements:

Table 1 - 2: Interaction elements

Interaction Element	Description
`source_widget_name`	Name of the widget from which the `event` is triggered.
`event_name`	Name of the triggered event.
`target_widget_name`	Name of the widget into which the `action` has to be performed.
`action_name`	Name of the action to be performed in the target widget when the event is triggered from the source widget.
`function_content`	Mapping function content. This is the mapping code performing the binding between event parameters and action variables (see *"Mapping function"* on page 1-9),

MAPPING FUNCTION

In an interaction, event and action(s) are bound together through a *mapping function*.

As mentioned before, actions are based upon transactions (or sequences) and events are based upon extraction rules or are manually defined. A mapping must be performed between interacting event and action(s) for their parameters to match.

For example, if an event:

- is triggered when clicking on a table row containg several columns, and
- triggers an action based upon a transaction taking a `keyword` variable in input,

then one column values of the clicked row must be bound to the `keyword` parameter, for the

action to be executed using the proper parameter value (that is, the data sent from the event).

Interactions and mapping functions are set in an **Interaction hub** project in the Convertigo Studio (see *"Convertigo Studio"* on page 1-10) after projects implementing interacting widgets have been created. This Interaction hub project is deployed as a *hub widget* in the **Mashup Composer**.

PAGE

A page allows to create the composite application by adding and laying out widgets (displayed widgets and interaction hub widget). Several pages may compose a composite application. They are created and laid out directly in the **Mashup Composer**.

1.3 Convertigo Studio

This section describes the Convertigo 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 Convertigo 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:

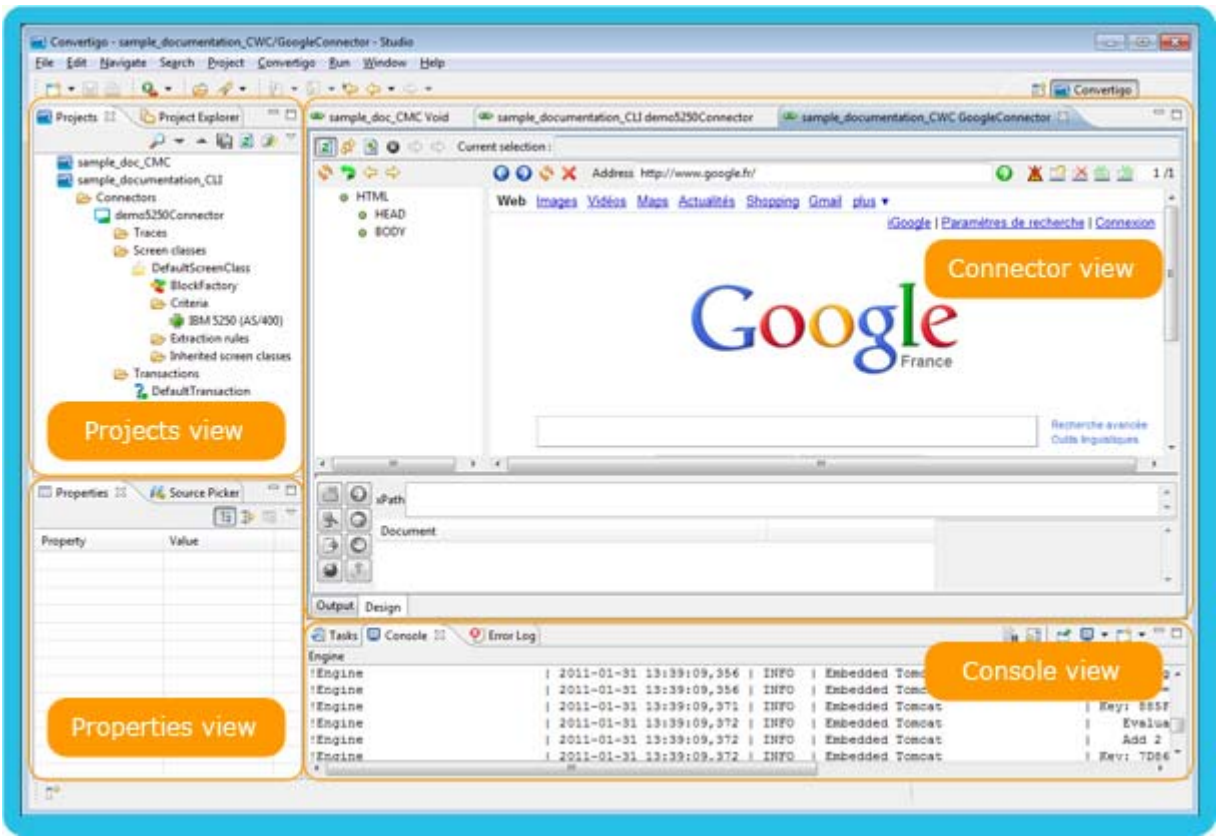

Figure 1 - 9: Convertigo Studio (displaying a web application in HTML connector)

The following table describes the views of the Convertigo Studio:

Table 1 - 3: Convertigo Studio views

View Name	Description
Projects view	Displays in a tree structure all projects (and related objects) used for declaring the Mashup Composer elements. Any kind of Convertigo project (CWC, CLI, CWI, etc.) can be opened in the **Projects** view.
Properties view	Displays the properties of objects 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. Depending on the type of project opened (CLI, CWI, CWC, etc.), the **Connector** view can be divided into two or more parts. For more information on project-specific **Connector** views, refer to the dedicated Quick Guides. The **Connector** view can also contain **Document Editors**, when files (in editable format such as .xml, .css or .xsl) are edited.
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 - 4: 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 Convertigo 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 - 4 on page 1-12), the **Projects** view includes tabs and icons:

Table 1 - 5: **Projects** view tabs and icons

View Element		Description
Tabs	Projects	Convertigo view. Contains all projects and related objects. Convertigo objects are presented in a tree structure with folders.
	Project Explorer	Standard Eclipse view. Contains all projects and related resources. 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. Opens a wizard allowing to import a 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 - modified and unsaved object;
- red - disabled object;
- orange - not executable object (child of a disabled object);
- grey - inherited object.

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

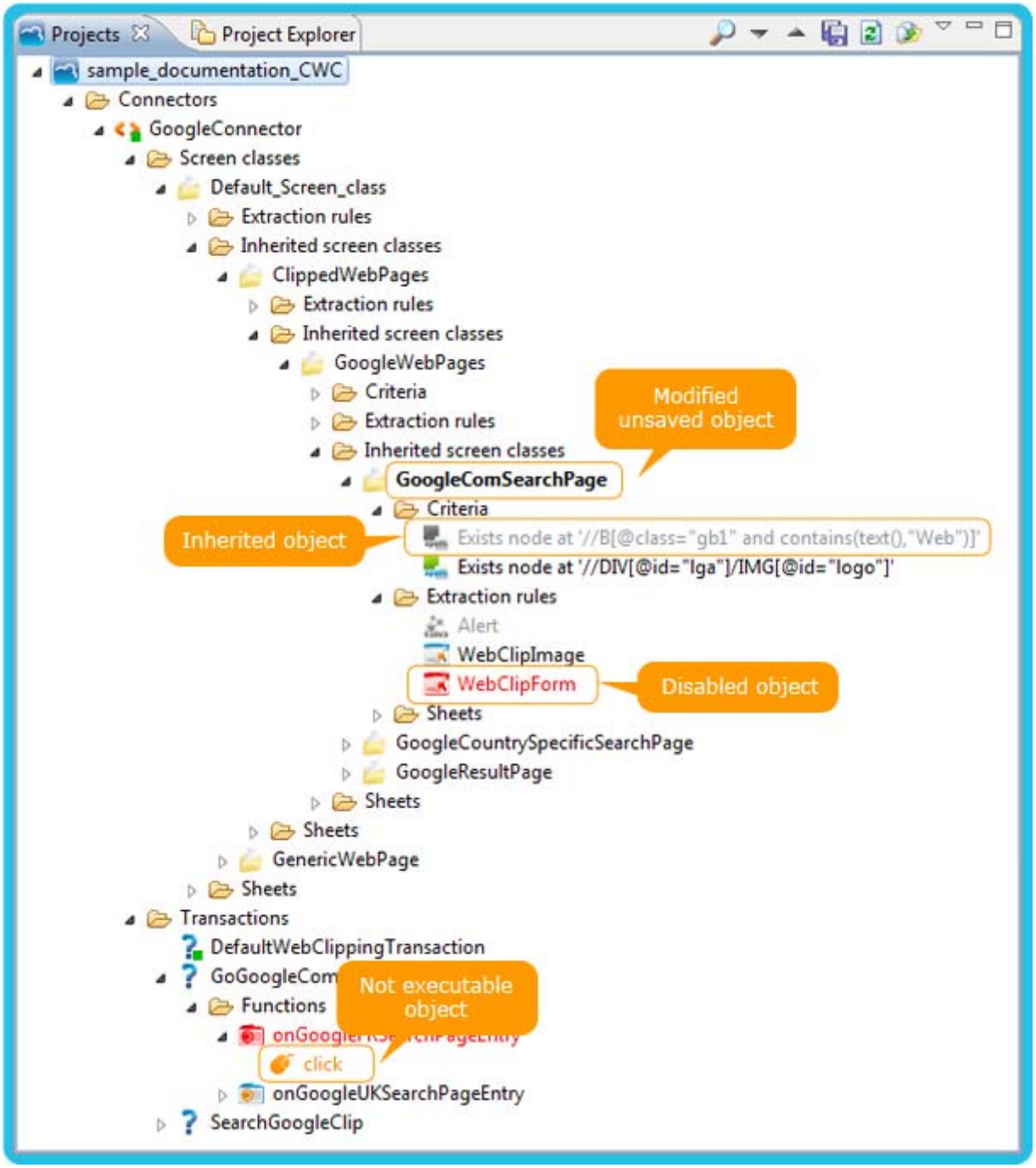

Figure 1 - 10: Formatting of unsaved, inherited and disabled objects in the Projects view

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 (*Base properties*, *Expert*, *Information*, *Selection*, *Configuration*, etc. - categories can be hidden by clicking on) that can be either expanded or collapsed by clicking respectively on or .

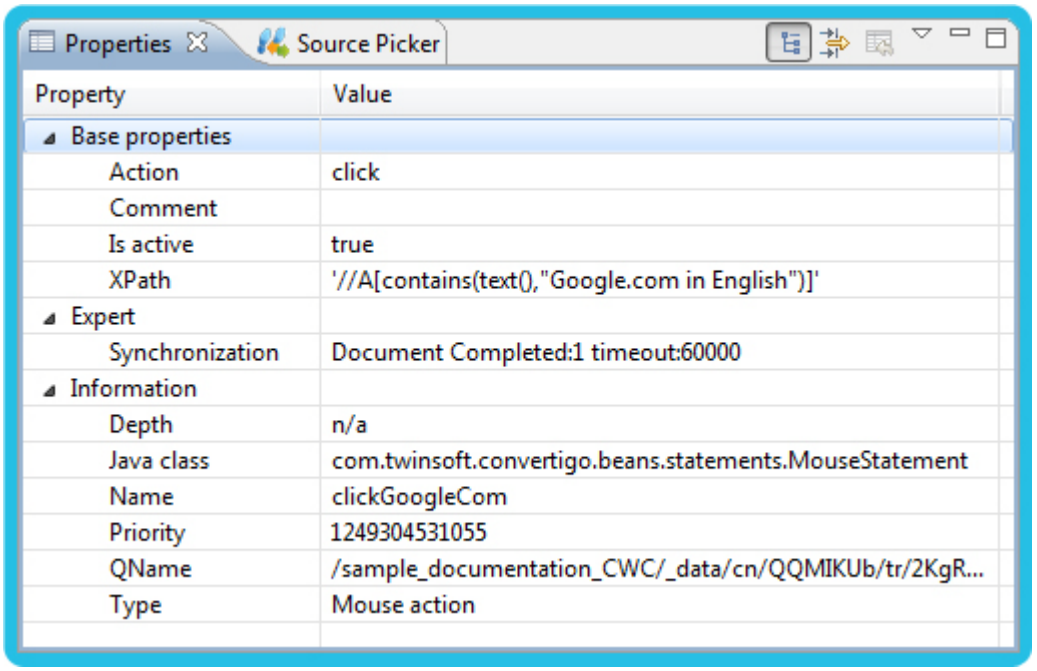

Figure 1 - 11: Properties view

In addition to object properties and standard Eclipse buttons (see Table 1 - 4 on page 1-12), the **Properties** view includes one tab (*Properties*) and icons:

Table 1 - 6: 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`:

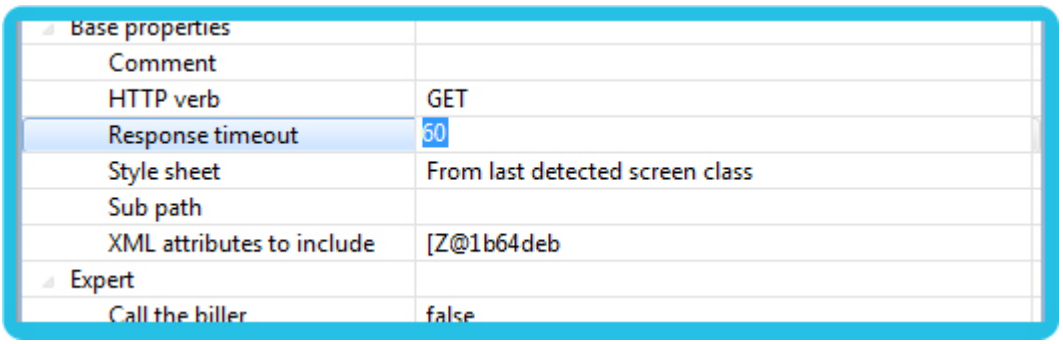

Figure 1 - 12: Text property field edition

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

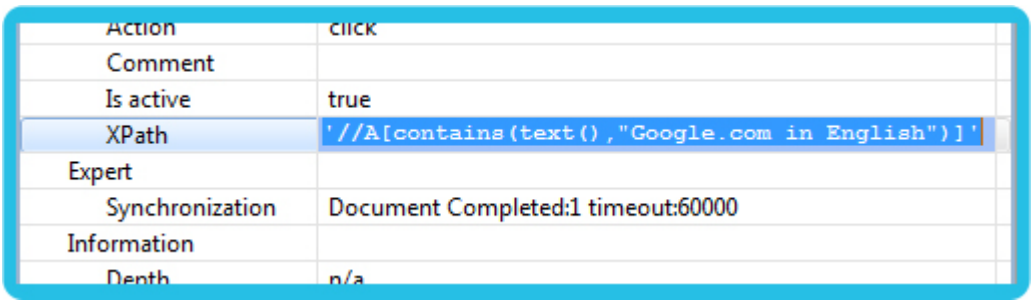

Figure 1 - 13: Java expression field edition

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

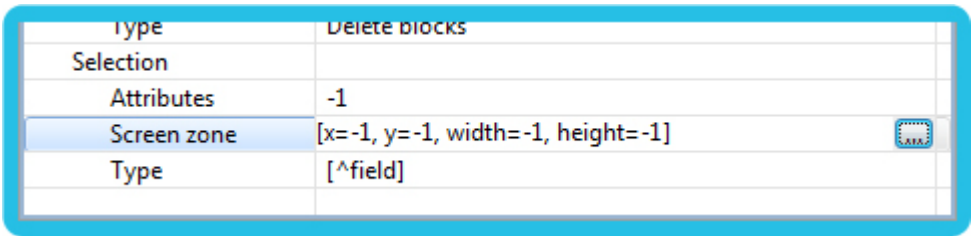

Figure 1 - 14: Text property edition via edition window

Click on . Depending on the property type:

- a Javascript expression window or an edition window opens:

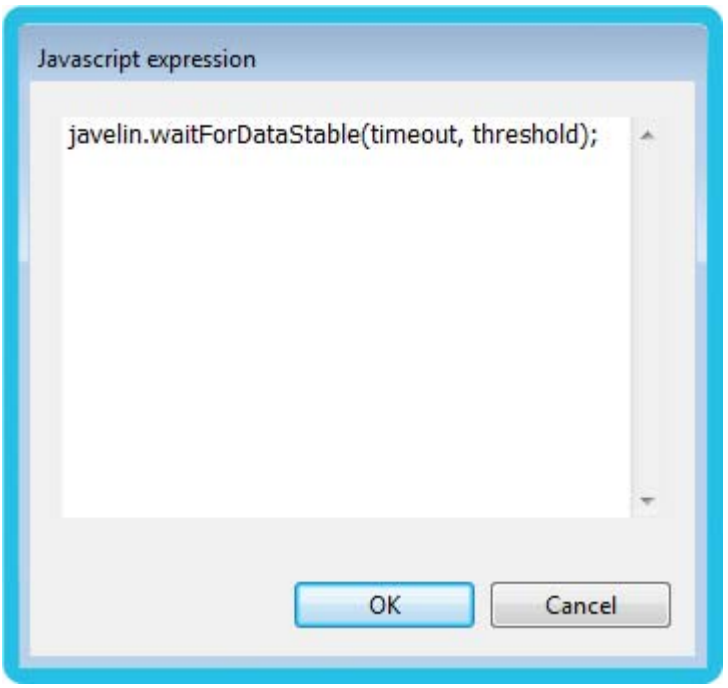

Figure 1 - 15: Javascript expression window

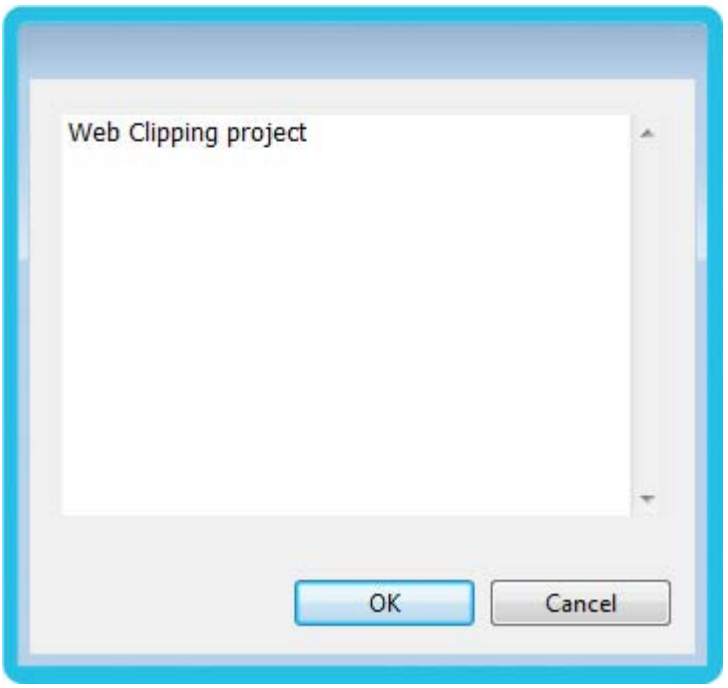

Figure 1 - 16: Edition window

- Enter the new JavaScript expression or the text comment in the window.
- Click on **OK**.

- a configuration window opens (two following examples: the **Attributes** editor and the Mashup event editor for legacy extraction rules):

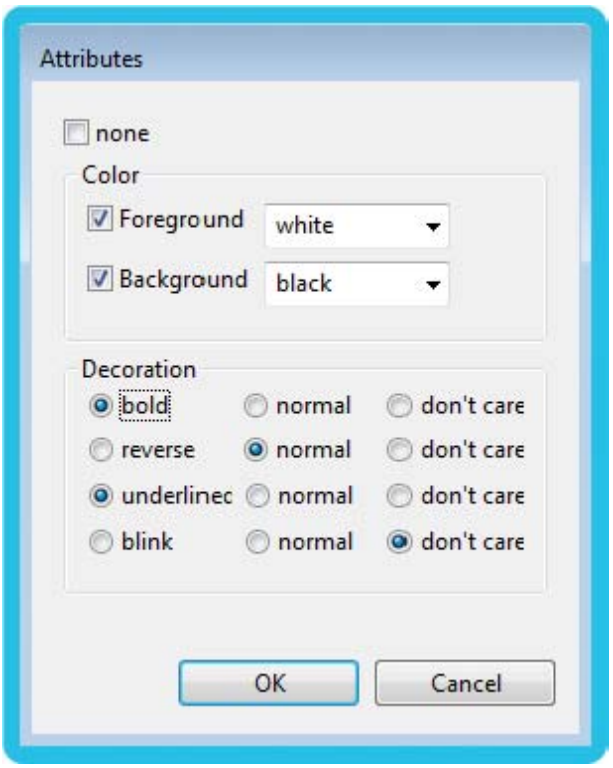

Figure 1 - 17: Configuration window: Attributes editor

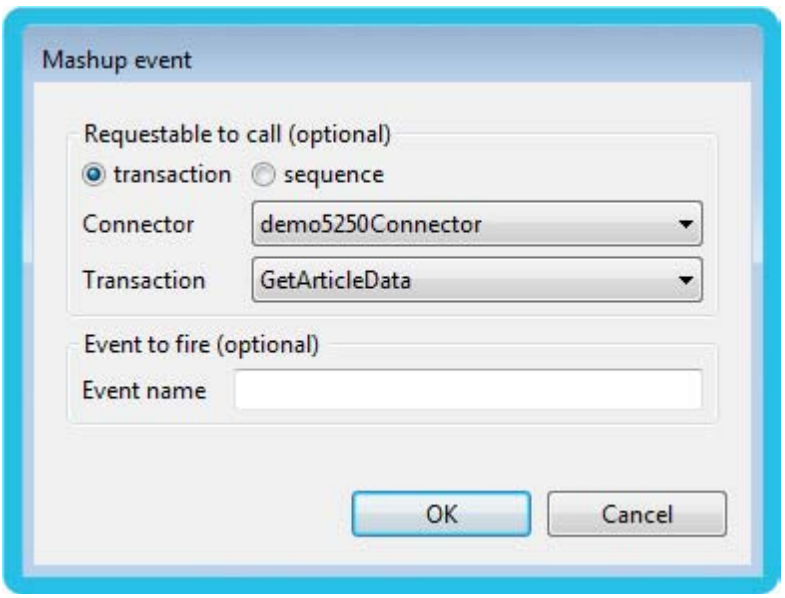

Figure 1 - 18: Configuration window: Mashup event editor

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

▸ the value is highlighted and a symbol appears on the right of the field:

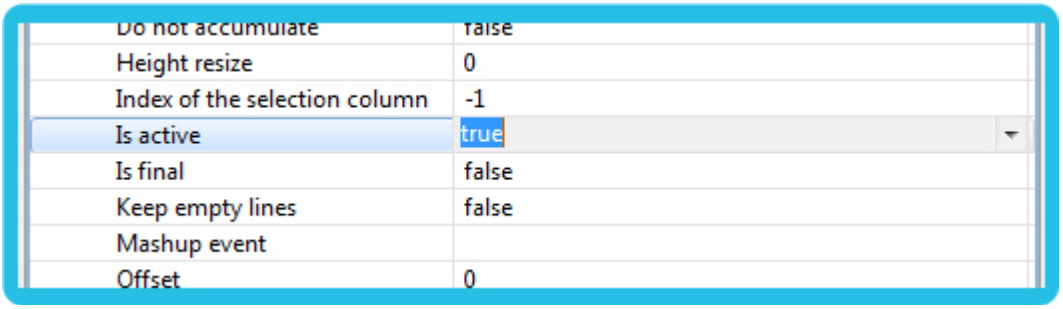

Figure 1 - 19: Property edition via drop-down menu

- Click on . A drop-down menu appears:

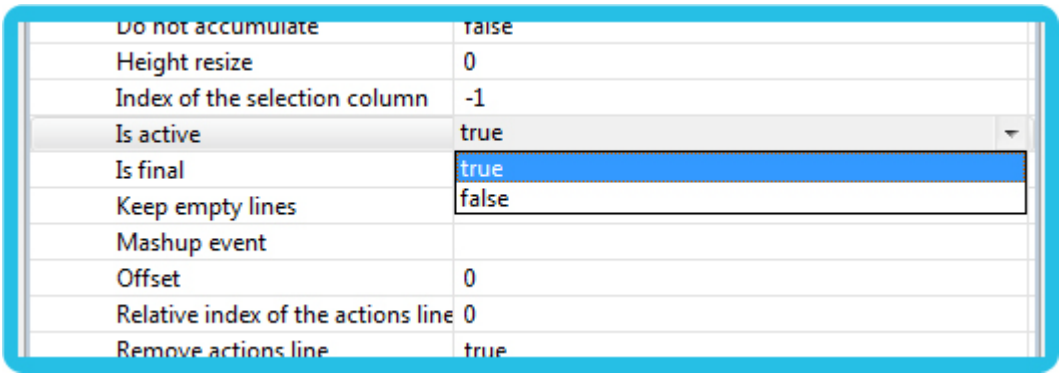

Figure 1 - 20: Selecting a value in the drop-down menu

- Select a value in the drop-down menu.
- Press `Enter`.

CONNECTOR VIEW

The **Connector** view displays any information related to project-specific connectors: HTML, HTTP, Legacy (Javelin), SQL, etc.

It can be divided into several parts depending on the type of connector displayed.

***Connector** views associated with specific connectors are described in greater details in dedicated Quick Start Guides.*

The **Connector** view can also contain **Document Editors**.

When double-clicking on a file to edit it, a **Document Editor** opens in a tab in the **Connector** view, next to already opened connectors.

Tabs are identified by specific icons:

Table 1 - 7: Tab icons in Connector View

The icon...	...identifies...
	a connector tab.

Table 1 - 7: Tab icons in Connector View

The icon...	...identifies...
	an editor tab.

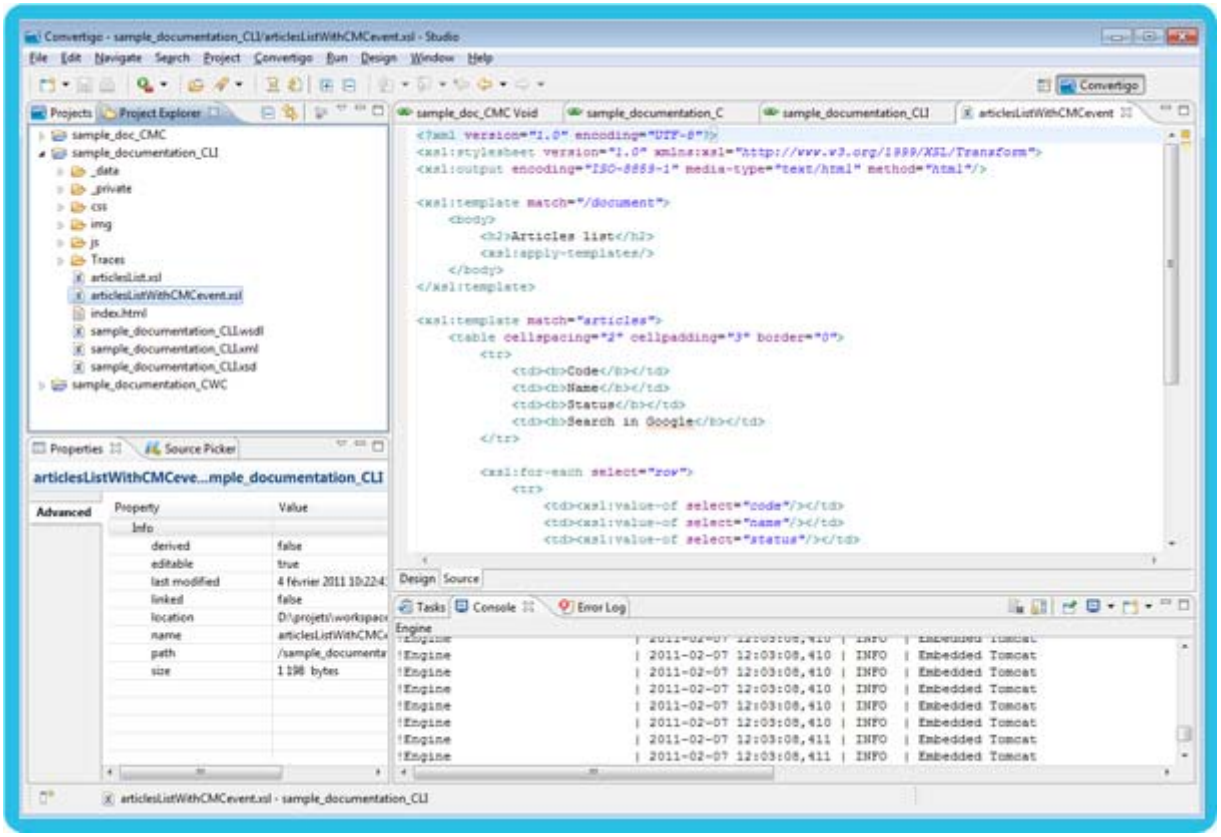

Figure 1 - 21: Style sheet in XSL format opened in Document Editor in the Connector view

1.4 Mashup Composer

The Mashup Composer is based on the open source GateIn portal.

For more information, consult the GateIn documentation at :
http://www.jboss.org/gatein/documentation.

2 My First Convertigo Mashup Composer Project

This chapter gives instructions on how to set up a simple CMC project, using first the *Convertigo Studio* then the *Mashup Composer*.

- Prerequisites - Key Quick Guide Projects
- Mashup Composer Project Description
- Setting Up the Convertigo Part of the Project - Creating Mashup Elements
- Setting Up the Mashup Composer Part of the Project - Developing the Composite Application
- Finalizing and Testing the Composite Application

2.1 Prerequisites - Key Quick Guide Projects

This Mashup Composer project (called `sample_documentation_CMC`) is based on two other Quick Guides projects:

- the **Convertigo Web Clipping** (**CWC**) `sample_documentation_CWC` project, which purpose is to:
  - *connect* to the Google search application,
  - *clip* search form from the Google search page,
  - *send* a keyword in the Google homepage search field,
  - *click* the Google Search button,
  - *clip* results as well as the navigation bar from the Google results page.

    In the context of the Mashup Composer project, the `GoGoogleCom` and `SearchGoogleClip` CWC transactions are used.
- the **Convertigo Legacy Integrator** (**CLI**) `sample_documentation_CLI` project, the purpose of which is to:
  - *connect* to an article management legacy application,
  - *navigate* through legacy screens,
  - *generate* an XML output containing a number of article information (article code, name, product group, etc.) collected in the process,

- *display* these data in an HTML presentation.

  In the context of the Mashup Composer project, the `GetArticleData` CLI transaction is used.

For more information, refer to the "Starting with Convertigo Web Clipper" and " Starting with Convertigo Legacy Integrator" Quick Guides.

2.2 Mashup Composer Project Description

The purpose of this Mashup Composer project (called `sample_documentation_CMC`) is to build a composite application displaying two widgets called `GoogleClip` and `ListOfArticles`, interacting with each other thanks to an interaction hub widget named `InteractionHub`.

The table below describes these widgets:

Table 2 - 1: Sample project widgets - Name and description

Widget name	Description
`GoogleClip`	Connects to the Google search engine
`ListOfArticles`	Shows data extracted from an article management legacy application
`InteractionHub`	Defines and performs interactions between the two other widgets.

Once created and laid out as required by the user on a specifically created page, widgets interact together following the interaction hub.

In fact, the `ListOfArticles` widget contains, for each piece of data extracted from the legacy application, a button for launching a keyword-based Google search in the `GoogleClip` widget:

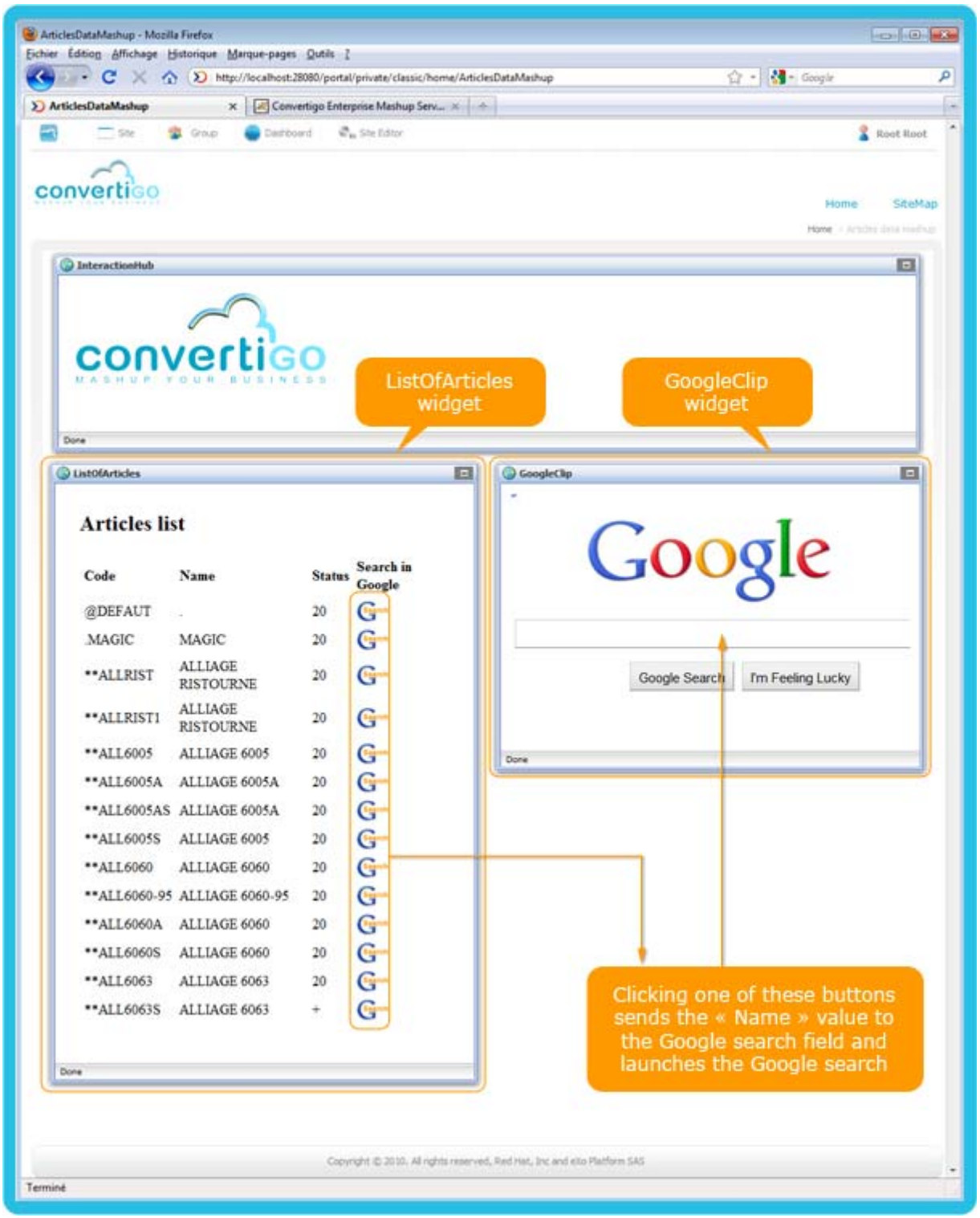

Figure 2 - 1: Widget interaction

The process can be summed up as follows:

1 *Create* in the ***Convertigo Studio*** all elements needed to compose the composite application: widgets, actions, events, interactions. Creating these elements means creating the following Convertigo projects:

A. *Create* the `sample_documentation_CWC` project described before, with all needed transactions (actions) so that it is able to connect to Google search

engine and to launch a specific search and clip results,

B. *Create* the `sample_documentation_CLI` project described before, defining an event through the XSL stylesheet so that the widget is able to trigger an event for each `articles` row generated by the `GetArticleData` transaction, thanks to a button dynamically added to each displayed row,

C. *Create* the `sample_doc_CMC` interaction hub project defining interactions.

2 *Deploy* the three projects from the ***Convertigo Studio*** to the ***Convertigo Server***.

3 *Declare* in the ***Mashup composer*** all the widgets based upon the previously defined projects thanks to the test platform:

A. *Add* in the Mashup composer a `GoogleClip` widget based upon the `GoGoogleCom` transaction set for the `sample_documentation_CWC` project. This transaction connects to the Google search engine (with empty search field).

When opened, widgets automatically execute the default action they are associated with (that is, the transaction they are based upon). The purpose of the `GoogleClip` widget is, in the end, to display all Google results returned after the Google search button included in the `ListOfArticles` widget has been clicked. So there is basically no need for the `GoogleClip` widget to display first an empty Google search form. This is done only to avoid having a "blank" widget when launched.

B. *Add* in the Mashup composer a `ListOfArticles` widget based upon the `GetArticleData` transaction set for the `sample_documentation_CLI` project. This transaction extracts a list of articles from a legacy application and displays the list in HTML.

C. *Add* in the Mashup composer an `InteractionHub` interaction widget based upon the `sample_doc_CMC` interaction hub project.

4 *Create* in the ***Mashup Composer*** the composite application using previously declared elements:

A. *Create* a page in the site to insert the `ListOfArticles`, `GoogleClip` and `InteractionHub` widgets.

B. *Add* widgets and *arrange* the layout on the page.

The setting-up is logically divided into two main parts:

- A first part dedicated to actions performed in the Convertigo Studio to create all elements needed afterwards in the Mashup Composer and deploying these elements to the Convertigo Server - Setting Up the Convertigo Part of the Project - Creating Mashup Elements.
- A second part dedicated to deploying the widgets from Convertigo to the Mashup Composer and creating the composite application strictly speaking in the Mashup Composer - Setting Up the Mashup Composer Part of the Project - Developing the Composite Application.

2.3 Setting Up the Convertigo Part of the Project - Creating

Mashup Elements

The following sections explain step by step how to create all mashup elements needed in the project to later compose the described composite application (see *"Mashup Composer Project Description"* on page 2-2):

- Creating projects in Convertigo Studio
- Creating all needed actions in CWC project
- Creating needed event in CLI project
- Creating interaction hub project
- Deploying all projects to Convertigo Server

2.3.1 Creating projects in Convertigo Studio

In this sample project, two widgets called `GoogleClip` and `ListOfArticles` are created. They are based upon transactions that have been developped in `sample_documentation_CWC` project set in the context of the *"Starting with Convertigo Web Clipper"* Quick Guide and `sample_documentation_CLI` project set in the context of the *Starting with Convertigo Legacy Integrator* Quick Guide.

The first step is to import these two projects in the Convertigo Studio, if they are not already present.

To import the sample projects in Convertigo Studio

The **Convertigo Studio** allows you to easily import a sample project for each Quick Guide. To do so, follow the procedure described in each Quick Guide:

1. for the `sample_documentation_CWC` project, refer to the *"Starting with Convertigo Web Clipper"* Quick Guide, see *"Opening the Sample Project from the Studio"* on page 2-4,
2. for the `sample_documentation_CLI` project, refer to the *"Starting with Convertigo Legacy Integrator"* Quick Guide, see *"Opening the Sample Project from the Studio"* on page 2-4.

*For more information on how to open a sample project, check the **Opening the Sample Project from the Studio** section (Chapter 2) of any existing Quick Guide.*

WHAT COMES NEXT?

According to the project description (see *"Mashup Composer Project Description"* on page 2-2), the `GoogleClip` widget should have the ability to connect to Google search engine and to launch a specific search and clip results.

This means the widget should be added **Actions**. This is done in the next section.

2.3.2 Creating all needed actions in CWC project

This section will manage the add of **Actions** to a widget. Actions a widget can execute are all

the transactions (or sequences) from a project, the same project as the start transaction defining the widget. In other words, the project should contain:

- a start transaction that connects to Google search engine,
- an additionnal transaction that launches a specific search and clip results.

All corresponding transactions have to be created in `sample_documentation_CWC` project. The sample project already contains both needed transactions:

- the `GoGoogleCom` transaction, declaring no variable, connects to Google search engine and clips search form,
- the `SearchGoogleClip` transaction, declaring one variable named `keyword`, searches the input keyword in Google search engine and clips results and navigation bar.

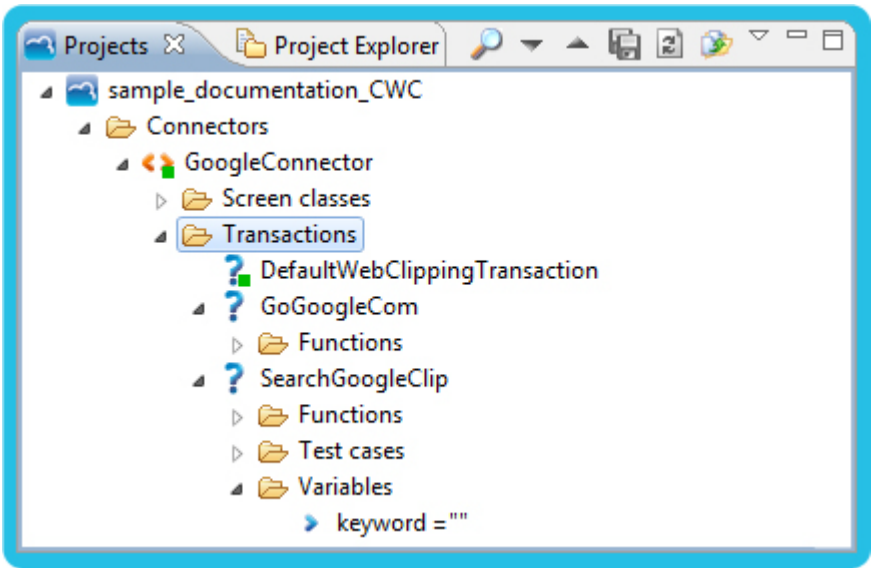

Figure 2 - 2: Transactions in sample_documentation_CWC project

There is no updates to make in the `sample_documentation_CWC` project.

For more information about developping transactions in CWC, refer to the "Starting with Convertigo Web Clipper" Quick Guide.

WHAT COMES NEXT?

According to the project description (see *"Mashup Composer Project Description"* on page 2-2), the `ListOfArticles` widget should have the ability to trigger an event for every displayed article row, thanks to a button dynamically added to each row. The updates to make to do so are explained in the next section.

2.3.3 Creating needed event in CLI project

The `ListOfArticles` widget displays the list of articles returned by the `GetArticleData` transaction.

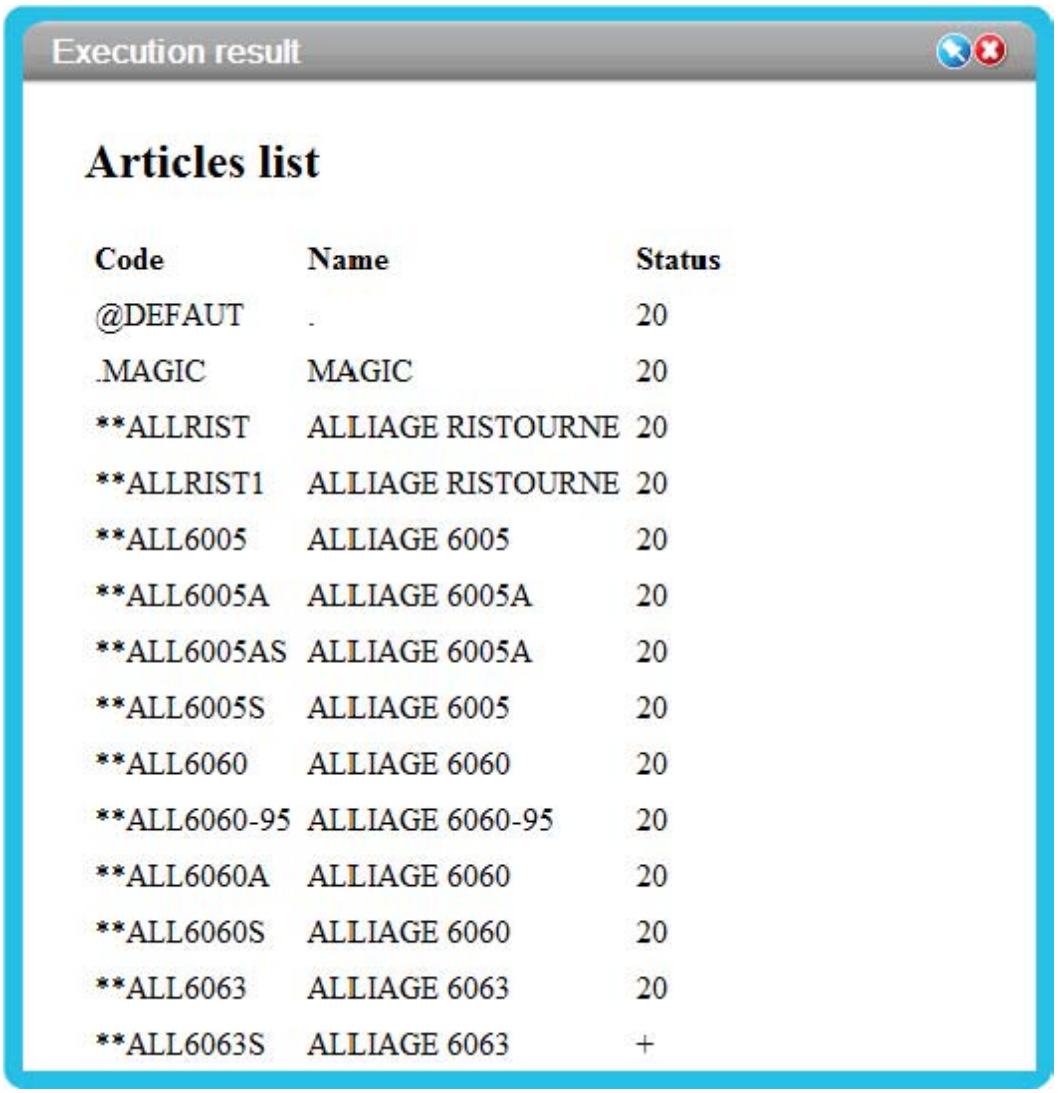

Figure 2 - 3: Displaying GetArticlesData transaction's result from sample_documentation_CLI project

For more information about developping transactions in CLI, refer to the "Starting with Convertigo Legacy Integrator" Quick Guide.

In the context of this sample project, the `ListOfArticles` widget also needs to be able to trigger an event for every `articles` row generated by the `GetArticleData` transaction, thanks to a button dynamically added next to each displayed row.

Adding the `ListOfArticles` widget the ability to trigger an event requires *improving* the widget display to dynamically add a button launching the event next to each displayed row (see Figure 2 - 1).

Improving the widget display requires *editing* the XSL file associated with the transaction upon which the widget is based, this means editing the XSL file associated with the `GetArticleData` transaction so that it meets our need of having a fourth column containing a dynamically created button with data to be sent with the event

To add an event-triggering button in a widget

1 In the **Projects** view, expand the transaction **Sheets** subfolder (for example

`GetArticleData` transaction).

2 Select the XSL style sheet object (for example `ArticlesList`).

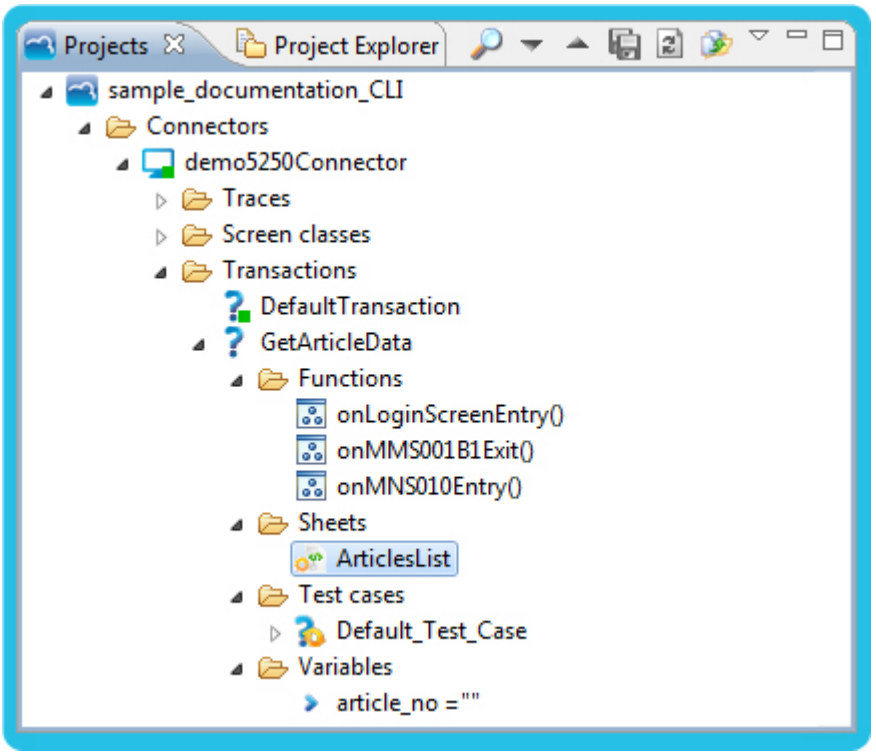

Figure 2 - 4: Selecting the transaction-related style sheet

The **Properties** view is automatically updated:

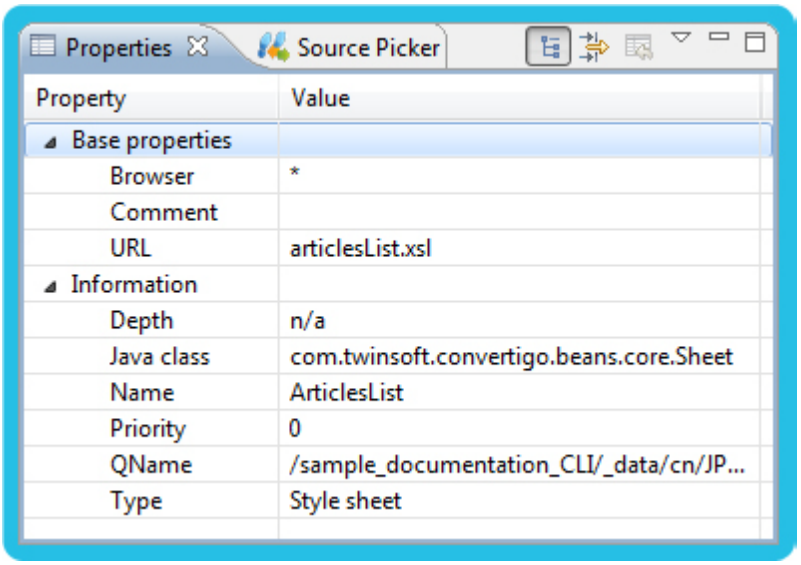

Figure 2 - 5: Style sheet properties

The XSL file URL is indicated in the **URL** property's **Value**.

3 In the **Projects** view, click on the **Project Explorer** tab.

4 Expand the `sample_documentation_CLI` project folder (see Figure 2 - 6).

5 Find the corresponding XSL file (`articlesList.xsl`).

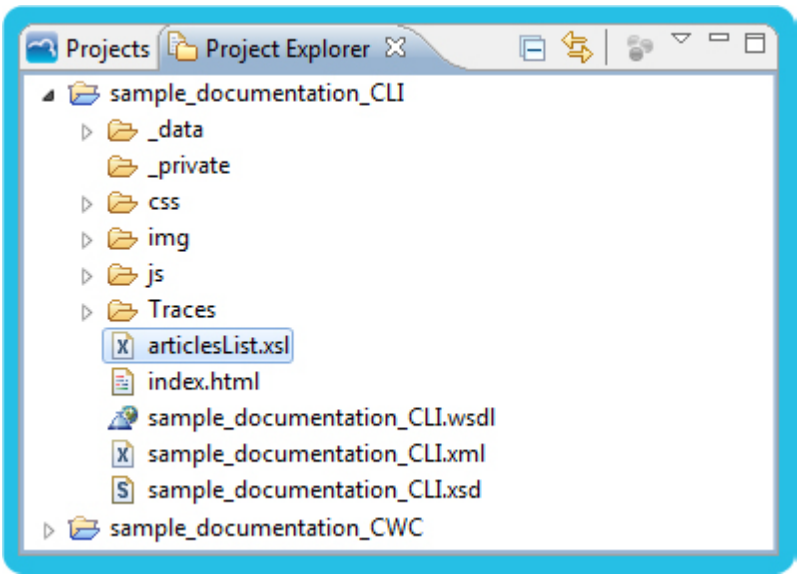

Figure 2 - 6: CLI project file tree

6 In the **Project Explorer**, double-click on the XSL file (for example `articlesList.xsl`) to open it in an editor.

*Depending on your environment, `.xsl` files open either in the **Source** or in the **Design** tab of **Document Editors**, in the same pane as connector views. Should it open in the **Design** tab, please switch to the **Source** tab by clicking on it before continuing.*

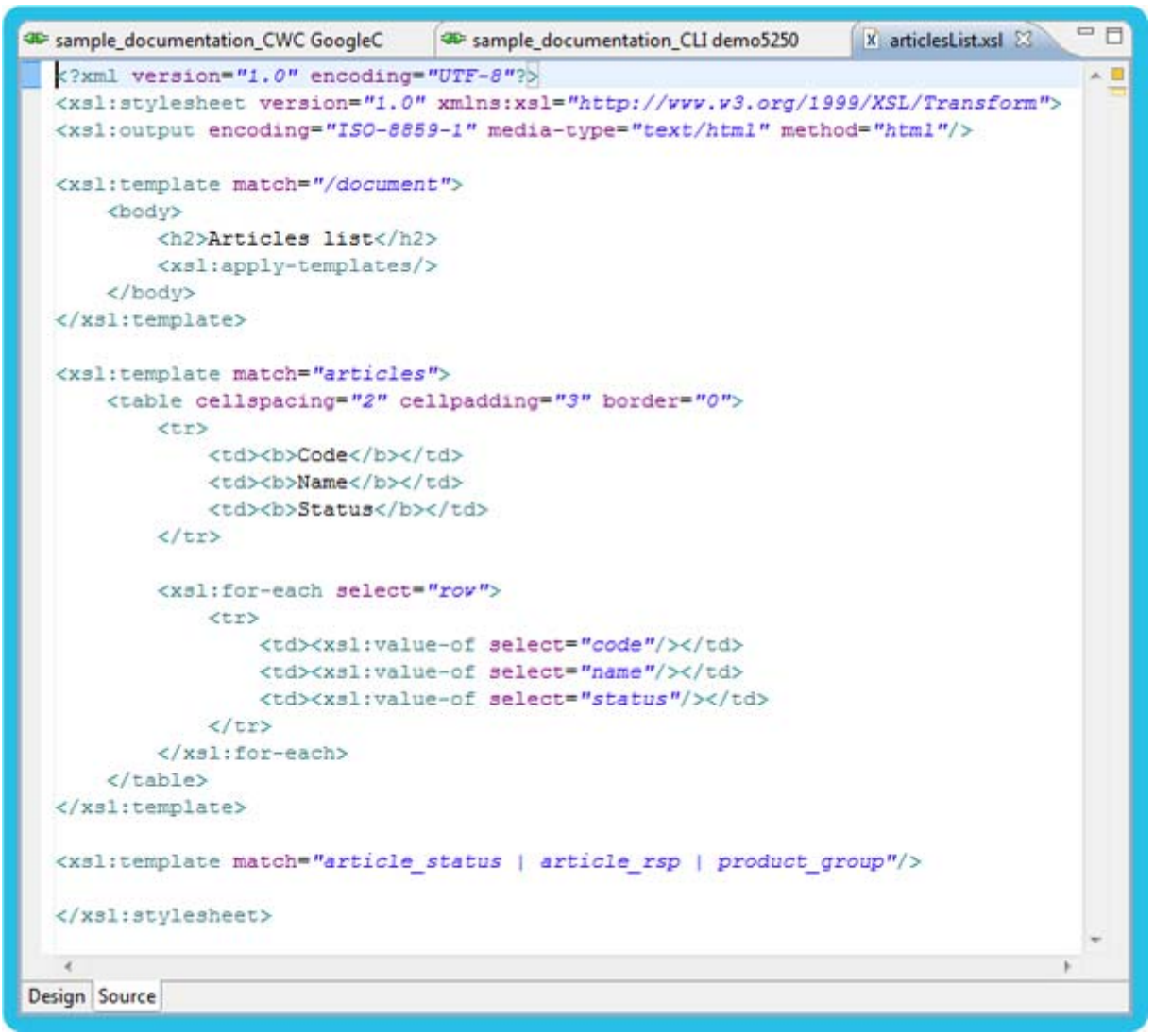

Figure 2 - 7: articlesList.xsl file in editor

We will now edit this XSL file to add to the template rule matching the `articles` table a fourth column containing a dynamically created button, with data to be sent with the event.

To this end, two additional `TD` elements (corresponding to the fourth column title and content) are needed in each `TR` element (corresponding to rows).

7 In titles `TR`, add a `TD` element with a title for the new column (for example `Search in Google`, see Figure 2 - 8).

8 In article rows `TR`, add a `TD` element containing an image (`IMG` element, see Figure 2 - 8). This image represents the item to be clicked for the event to be launched. The `<img>` tag needs:

- *required* attributes - `src` and `alt`;
- *standard* attributes - `style` and `title`;
- an *event* attribute - `onclick`, JavaScript expression allowing to trigger the event.

Set these attributes according to the following table:

Table 2 - 2: `IMG` element attributes

Attribute	Description	Value
`src`	Source attribute. Relative path to image.	(for example) `"img/googleSearch_logo.png"`
`title`	Image title. Text displayed as the mouse pointer moves over the object.	(for example) `"please click here to search for article"`
`alt`	Alternative text. Text displayed when the `IMG` element cannot be displayed.	(for example) `"please click here to search for article"`
`style`	Style attribute. Here, we use this attribute to define the type of cursor to display as the mouse pointer moves over the object.	`"cursor: pointer;"`
`onclick`	Event attribute, containing a JavaScript expression to be run on a mouse click. Here,it calls a Convertigo library function triggerring an event. This function takes two parameters: • the event name (for example `event_articleClicked`), and • the event payload (or parameters). Here the event sends a structure containing the article's three pieces of data. Beware this value must be in proper XSL syntax (`{tagName}` to get the `tagName` element value).	`"C8O.doMashupEvent( 'event_articleClicked', {{ code:'{code}', name:'{name}', status:'{status}' }} );"`

For more information on the required, optional, standard and event attributes of the HTML <img> tag, see the W3Schools dedicated page at: http://www.w3schools.com/tags/tag_IMG.asp

*To help you edit the XSL file, a ready-to-use file called `articlesListWithCMCevent.xsl` is stored in the project. You can access it via the **Project Explorer** tab. You can either:*
- copy its content and paste it in the `articlesList.xsl` file (see Figure 2 - 8), or
*- set the **URL** property of the `ArticlesList` style sheet object (see Figure 2 - 5) to `articlesListWithCMCevent.xsl`.*

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

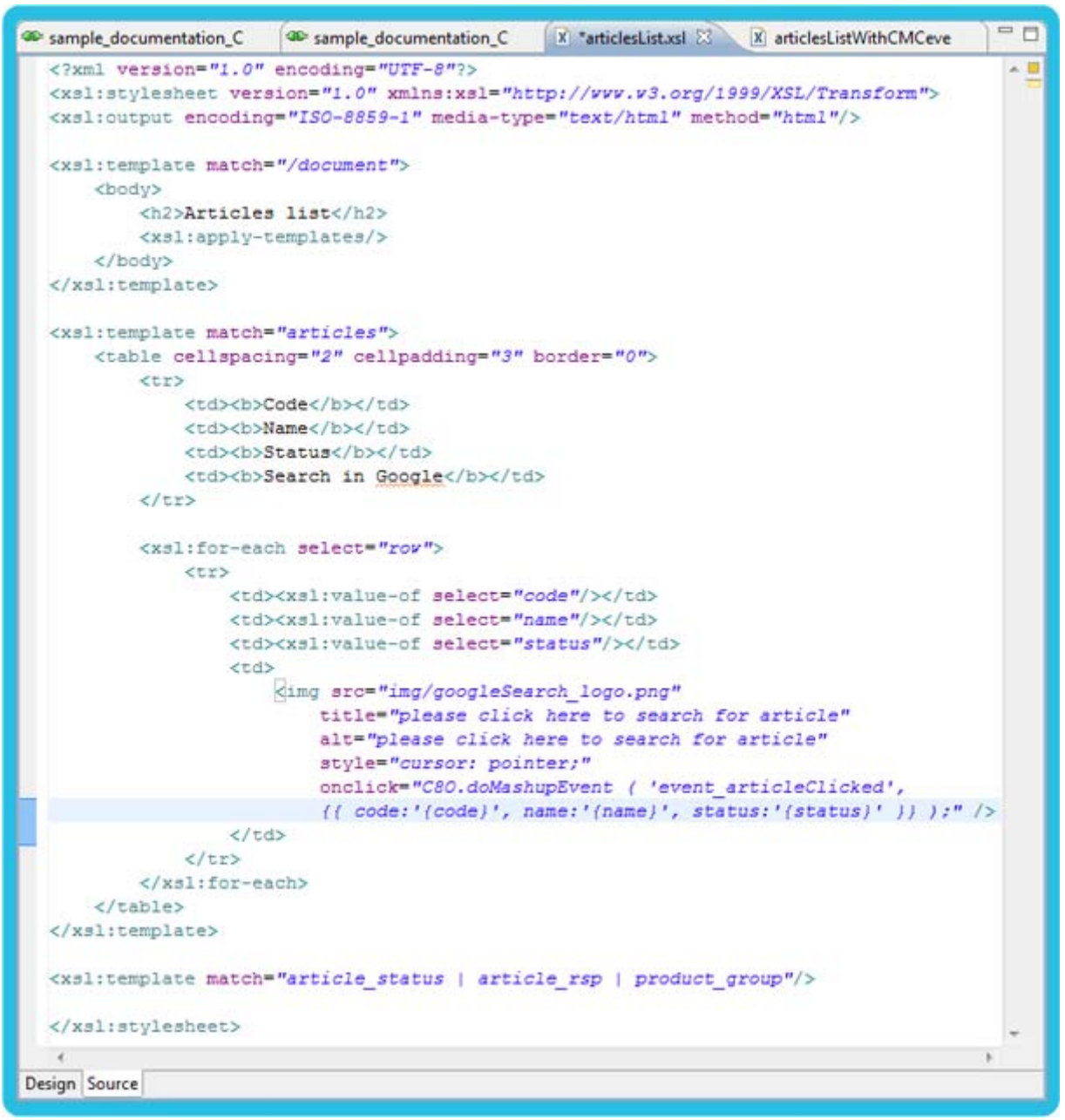

Figure 2 - 8: Edited articlesList.xsl

9 Save it by clicking on the **Save** icon of the Convertigo Studio or by pressing `Ctrl+S`.

WHAT COMES NEXT?

The ***Convertigo Studio*** has been used so far in the project to:

- create projects serving as basis for creating widgets;
- give a widget the ability to execute several actions;
- give a widget the ability to trigger an event dynamically sending data.

We will now set up the interaction hub project that will build the composite application by defining the interactions between the two widgets.

2.3.4 Creating interaction hub project

In the context of this sample composite application, one widget interaction is implemented. It can be defined as follows:

- it is triggered by the `event_articleClicked` event associated with the `ListOfArticles` widget,
- it launches the `SearchGoogleClip` action associated with the `GoogleClip` widget.

The `keyword` needed as parameter of the Google search *action*, in the `GoogleClip` widget, will be bound to the `name` data sent by the *event*, from the `ListOfArticles` widget.

The process to create interaction hub can be summed up as follows:

1 *Create* in the ***Convertigo Studio*** a `sample_doc_CMC` interaction hub project.

2 *Define* interaction, with involved widgets, event, action and mapping function, in the configuration file.

A completed sample interaction hub project is stored in Convertigo 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 Convertigo Studio

1 Select ***File > New > Project*** menu or in the **Studio** toolbar, click on (***New***) then select ***Project***.

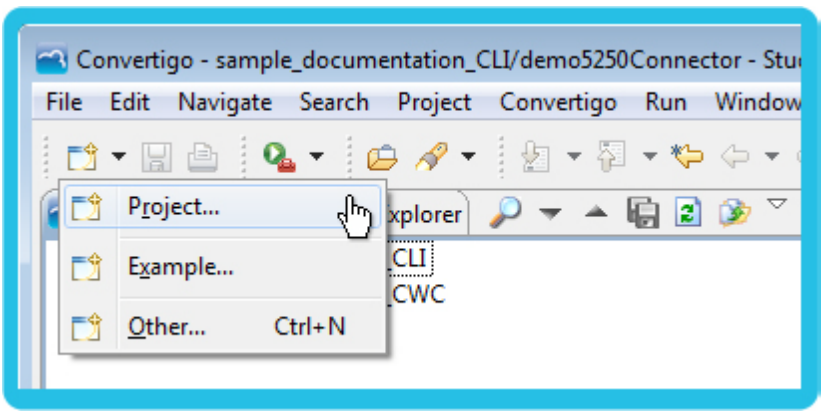

Figure 2 - 9: Launching the New Project wizard

A **New Project** wizard opens:

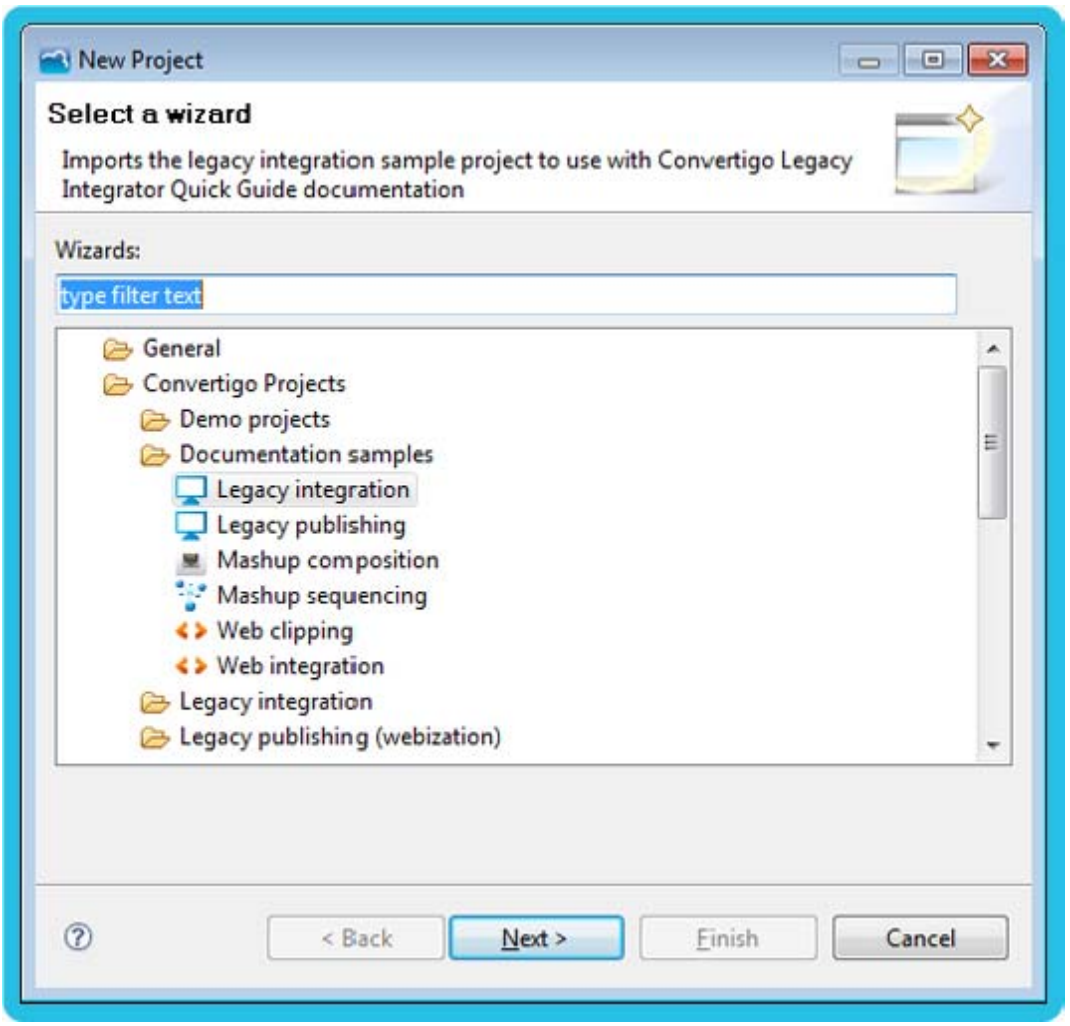

Figure 2 - 10: New Project Wizard

2 Expand **Convertigo Projects**, then **Documentation samples**, and select **Mashup composition**:

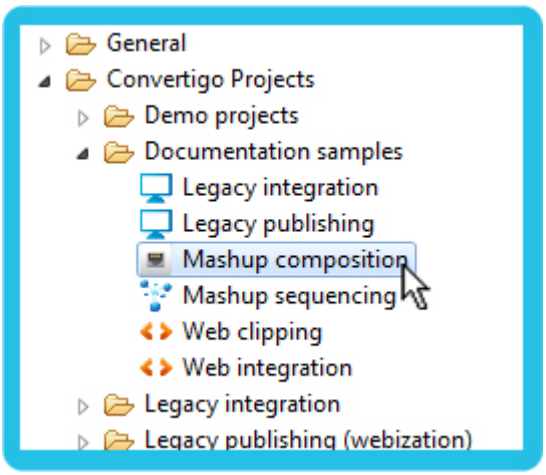

Figure 2 - 11: Selecting the Mashup Composition sample project

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

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

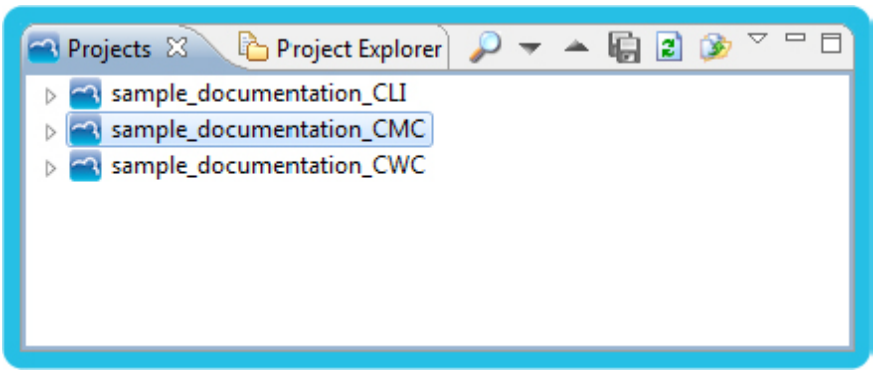

Figure 2 - 12: Sample project in Projects view

You can now start setting up your own interaction hub project, `sample_doc_CMC`.

To create interaction hub project

1 Select ***File > New > Project*** or in the **Studio** toolbar, click on (***New)*** then select ***Project***.

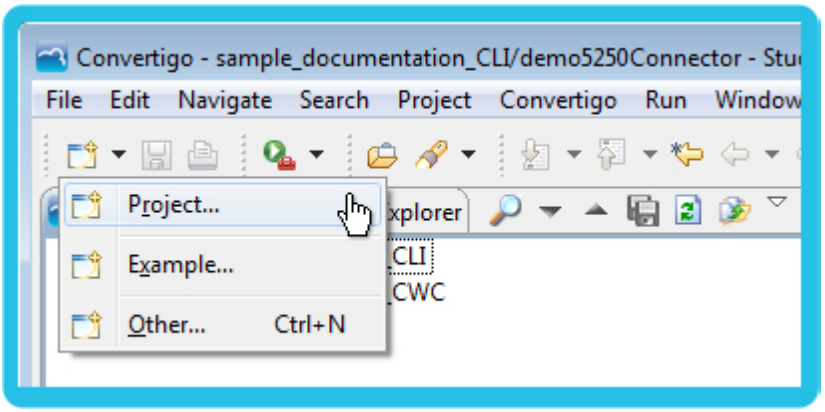

Figure 2 - 13: Launching the New Project wizard

A **New Project** wizard opens:

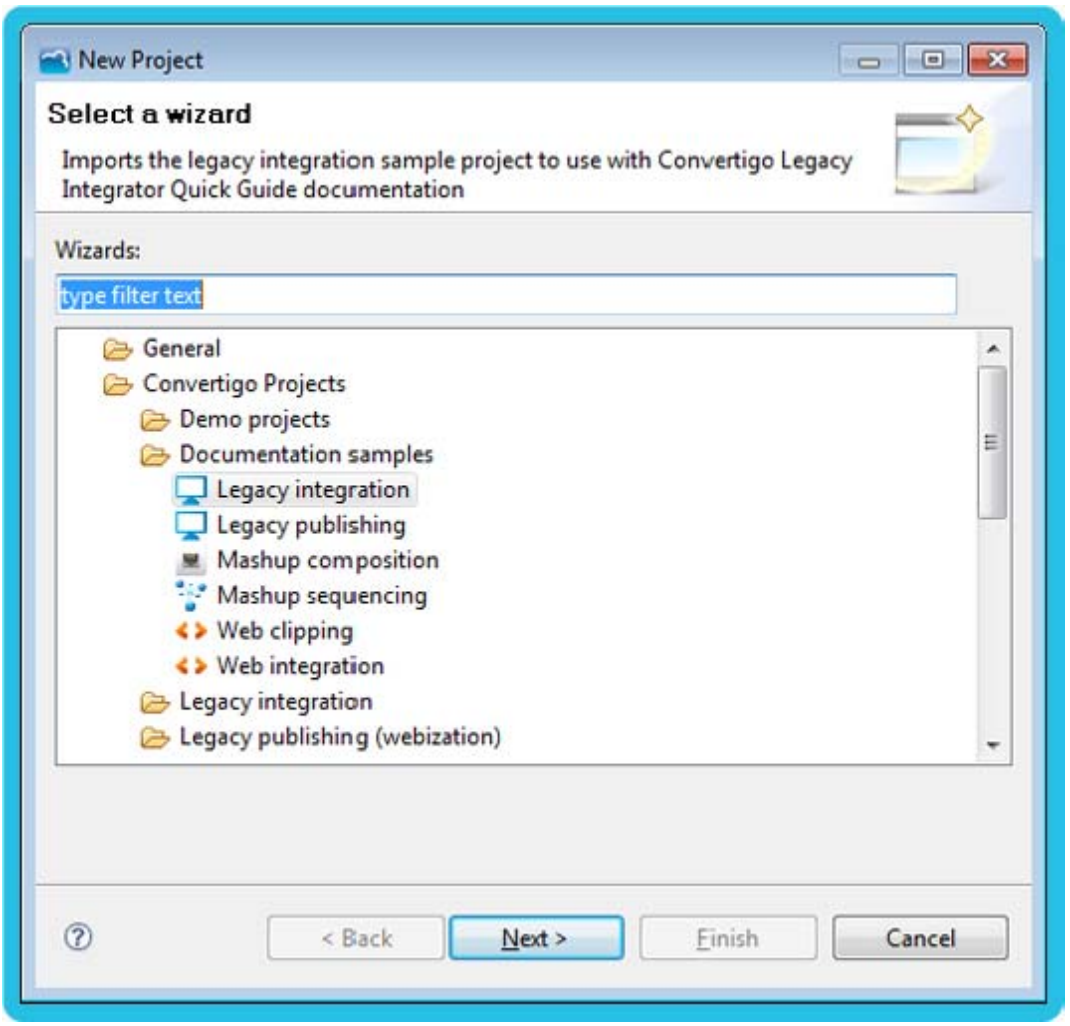

Figure 2 - 14: New Project Wizard

2 Expand **Convertigo Projects**, then **Mashup composition**, and select **Interaction hub**:

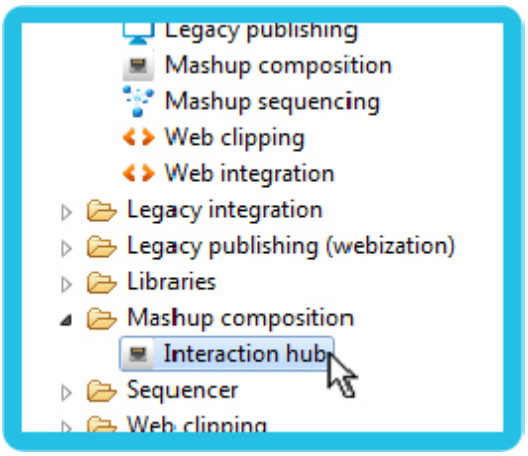

Figure 2 - 15: Selecting a project type

3 Click on **Next**:

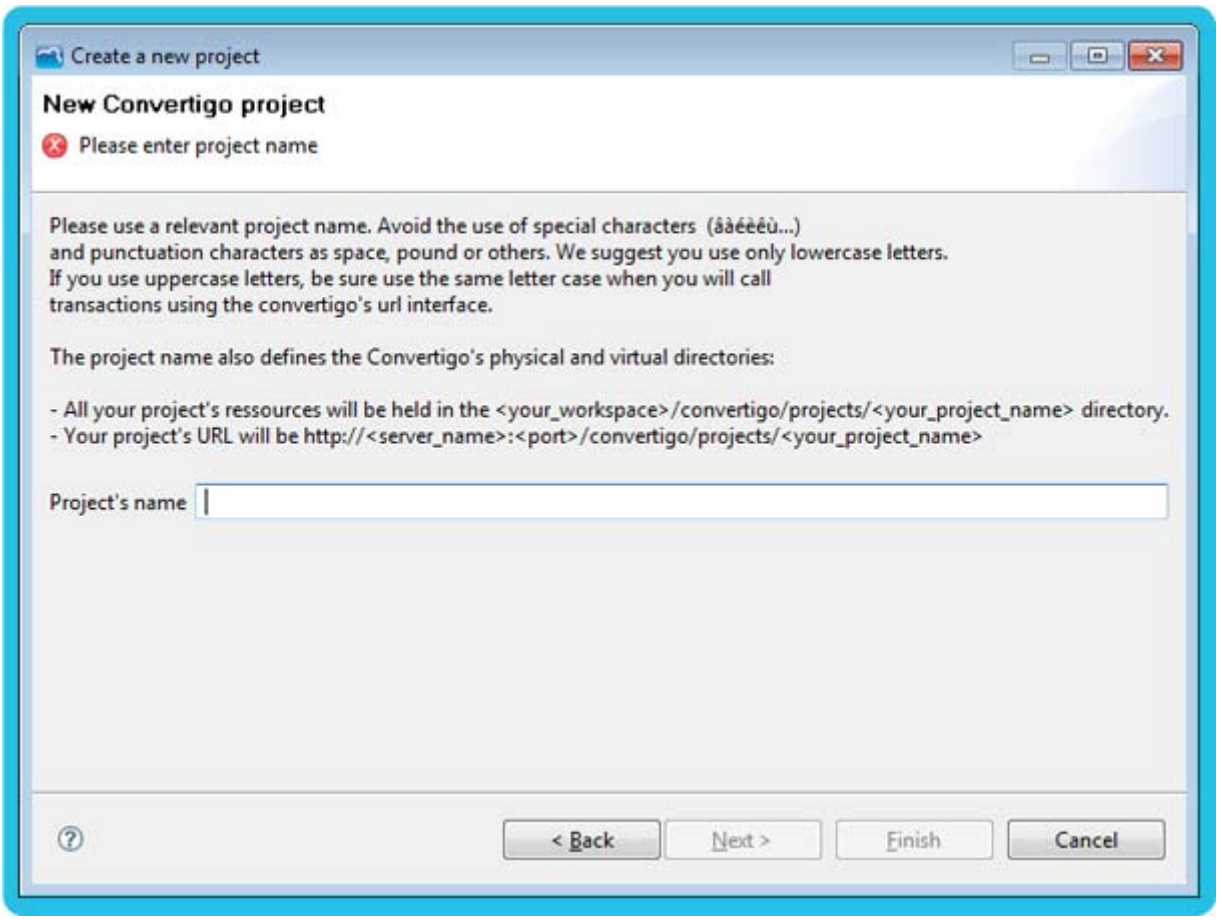

Figure 2 - 16: Entering a project name

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

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.

5 Click on **Finish**. The project now appears in the **Projects** view:

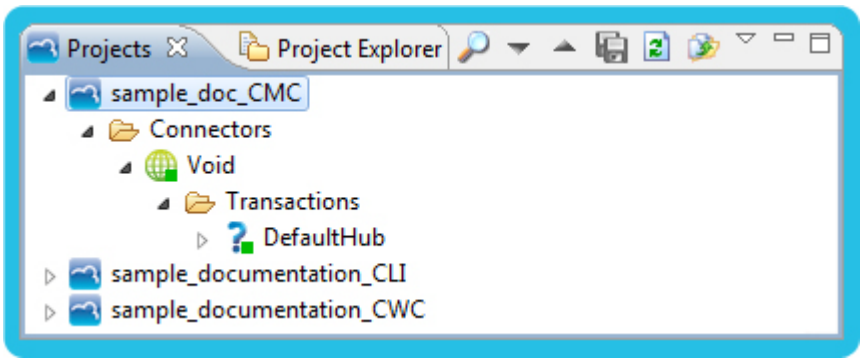

Figure 2 - 17: sample_doc_CMC project appearing in the Projects view

It contains by deafult a *Void* connector, that doesn't run, but its editor opens in the **Connector** view:

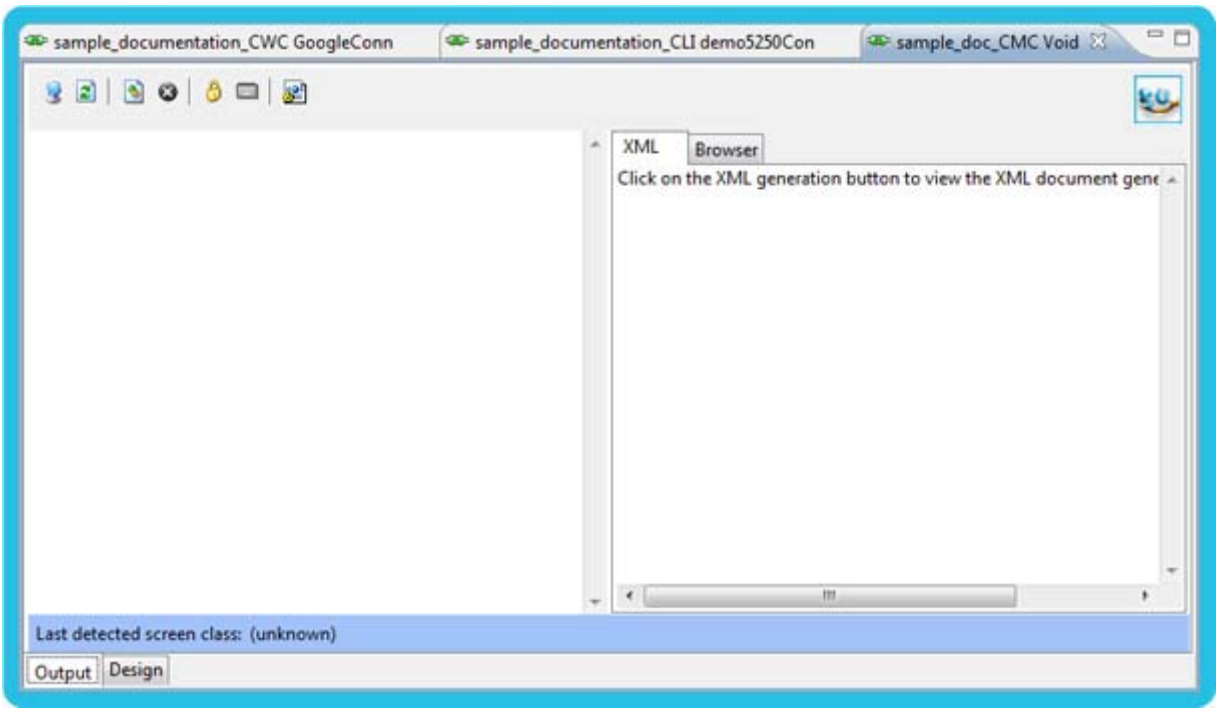

Figure 2 - 18: Connector view displaying CMC Void connector

This is the starting point of the project.

To define widget interaction on the configuration file

1. In the **Projects** view, click on the **Project Explorer** tab.
2. Expand the `sample_doc_CMC` project folder, then the `hub` folder.
3. Find the mapping configuration XML file (`default_hub.xml`).

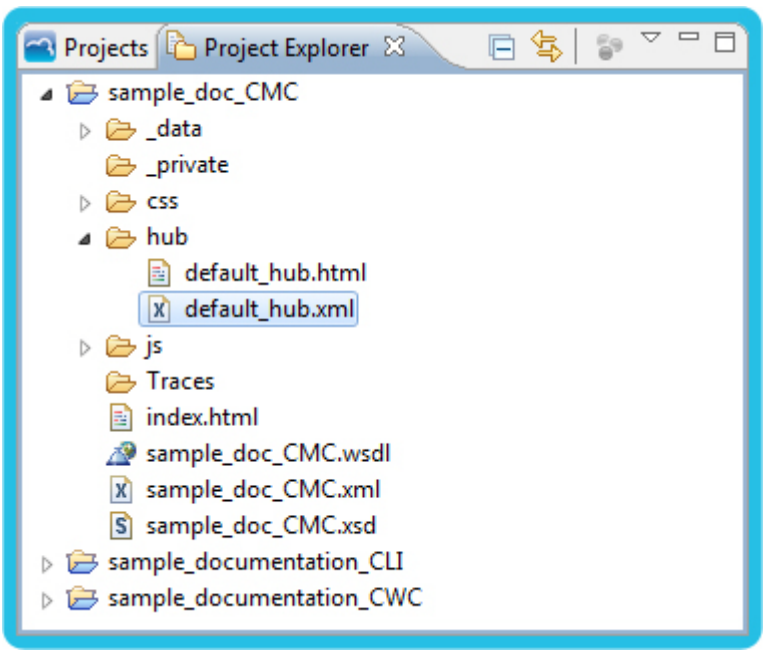

Figure 2 - 19: CMC project file tree

4. In the **Project Explorer**, double-click on the XML file `default_hub.xml` to open it in an editor.

*Depending on your environment, `.xml` files open either in the **Source** or in the **Design** tab of **Document Editors**, in the same pane as connector views. Should it open in the **Design** tab, please switch to the **Source** tab by clicking on it before continuing.*

Figure 2 - 20: default_hub.xml file in editor

This configuration file presents the interaction mapping in commented code.

We will now edit this XML file so that it meets our interaction need: an interaction between the two previously defined widgets.

The interaction is modelled in the form of a sentence:
On <*source_widget_name*> widget,
When <*event_name*> occurs,
On <*target_widget_name*> widget,
Do <*action_name*> action,
Mapping parameters thanks to <*function_content*> mapping function.

In our case, the interaction is defined as:
On `ListOfArticles` widget,
When `event_articleClicked` occurs,
On `GoogleClip` widget,
Do `searchGoogleClip` action,
Mapping parameters thanks to `myFunctionContent` mapping function (with `myFunctionContent` representing the mapping script, detailed later in this Quick Guide).

We will now set this interaction in the configuration XML file.

5 Uncomment the XML code that is present in the mapping configuration file. The XML sample introduces the way to bind events and actions, and their parameters, thanks to JavaScript code.

6 An `origin` element represents the source widget. The *source widget name* is set in the `name` attribute of this element. Set `ListOfArticles` widget name in this attribute:

```
<origin name="ListOfArticles">
    <event name="eventB1">
        <target name="WidgetA">
            <mapping><![CDATA[
                var today = new Date();
                return {
                    // action
                      transaction : "AddR
```

Figure 2 - 21: Editing source widget name in the mapping configuration file

7 An `event` element represents the event triggering in the interaction. The *event name* is set in the `name` attribute of this element. Set `event_articleClicked` event name in this attribute:

```
<origin name="ListOfArticles">
    <event name="event_articleClicked">
        <target name="WidgetA">
            <mapping><![CDATA[
                var today = new Date();
                return {
                    // action
                      transaction : "AddR
```

Figure 2 - 22: Editing triggered event name in the mapping configuration file

8 A `target` element represents the target widget. The *target widget name* is set in the `name` attribute of this element. Set `GoogleClip` widget name in this attribute:

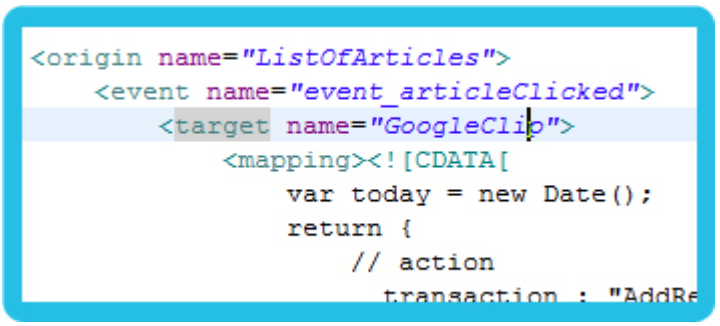

Figure 2 - 23: Editing target widget name in the mapping configuration file

Now that the target widget is defined, the next step is to define the action to launch in this target widget. No XML element is used to defined the action to launch, this is directly described in the mapping *function content*, next to the parameters mapping.

Several samples of JavaScript mapping code are proposed:

- one creating a new data structure returned to be passed to the action's input,
- two re-using the event's payload structure, editing it and returning it to be passed to the action's input.

*Returning the `payload` object at the end of the **mapping function**, when this structure is re-used to be passed to the **action**, is not mandatory as the `payload` structure is used by default if nothing else is returned from the **mapping function**.*

In the first sample, the new data structure is created directly in the `return` statement, JavaScript code can be used before return statement. In the returned structure, comments isolate action definition (corresponding to the transaction or the sequence to call in Convertigo project) and parameters mapping (corresponding to the transaction's or the sequence's variables to send).

9 In the context of this sample project, we are going to create a new data structure to define the action and bind its parameters with event's data. Using the first mapping sample, remove all unnecessary JavaScript code before the `return` statement:

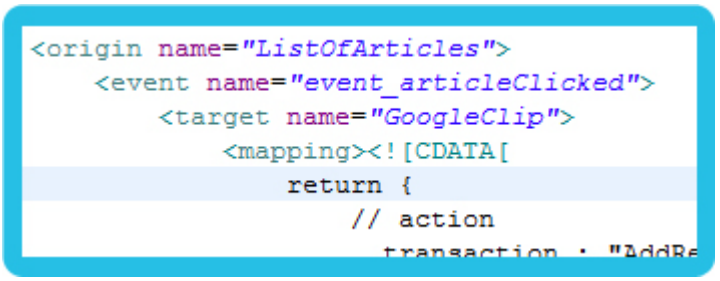

Figure 2 - 24: Removing unused JavaScript code

10 Modify the action definition code section by mapping `__transaction` variable with the name of the transaction to launch, `SearchGoogleClip`:

```
<origin name="ListOfArticles">
    <event name="event_articleClicked">
        <target name="GoogleClip">
            <mapping><![CDATA[
                return {
                    // action
                    __transaction : "SearchGoogleClip",
```

Figure 2 - 25: Mapping action name to __transaction variable

If the action to launch is a sequence, we would map the `__sequence` variable.

11 The `SearchGoogleClip` transaction has one defined variable, `keyword`. We want the interaction to pass a piece of data from the clicked article (`name`) into this variable.

Modify parameters mapping code section by mapping the `keyword` variable with the `name` parameter of the event payload, and remove all the other sample mapping variables:

```
<origin name="ListOfArticles">
    <event name="event_articleClicked">
        <target name="GoogleClip">
            <mapping><![CDATA[
                return {
                    // action
                    __transaction : "SearchGoogleClip",

                    // parameters
                    keyword : payload.name
                }
            ]]></mapping>
```

Figure 2 - 26: Mapping keyword variable

As the interaction launches only one action, and there is no other event that can trigger another interaction, the mapping is over.

Several actions can be triggered by an interaction but only one is set up in this interaction.

12 To finish, remove all unwanted sample mapping XML elements from the mapping configuration file:

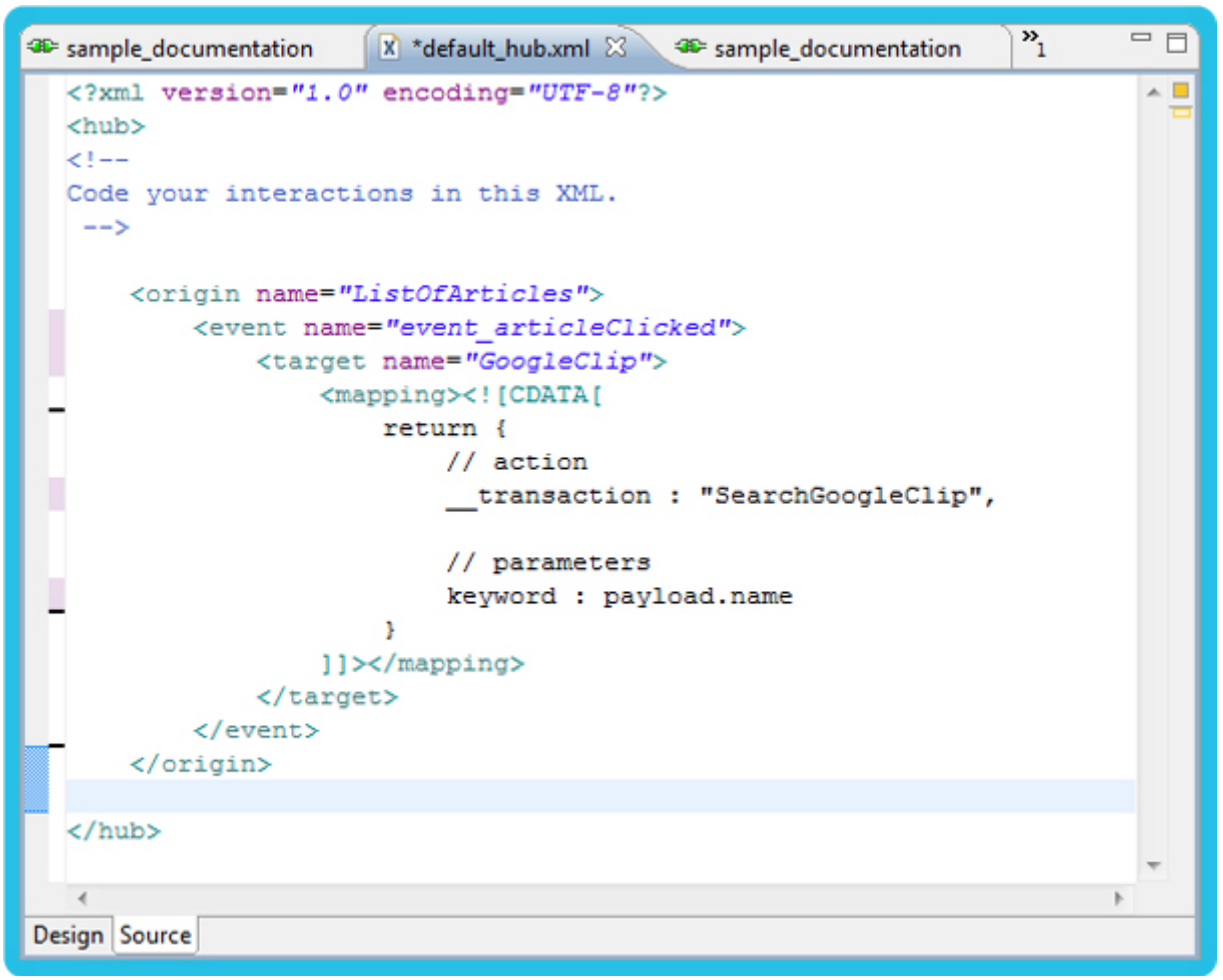

Figure 2 - 27: Finished mapping configuration file

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

13 Save it by clicking on the **Save** icon of the CMC application or by pressing `Ctrl+S`.

WHAT COMES NEXT?

According to the project description (see *"Mashup Composer Project Description"* on page 2-2), the projects should now be deployed from the Convertigo Studio to the Convertigo Server containing the Mashup Composer. This is done in the next section.

2.3.5 Deploying all projects to Convertigo Server

This section will explain the deployement of projects from the Convertigo Studio to the Convertigo Server that contain the Mashup Composer. In this way, it will be possible to deploy the widgets to the Mashup Composer and the Mashup Composer will be able to execute the projects transactions (see *"Setting Up the Mashup Composer Part of the Project - Developing the Composite Application"* on page 2-26).

All three involved projects have to be deployed from the Convertigo Studio to the Convertigo Server. Repeat the following procedure for each project:

To deploy a project from Convertigo Studio to Convertigo Server

1 In the **Projects** view, right-click on the project (for example `sample_documentation_CLI` project), then select ***Deploy***.

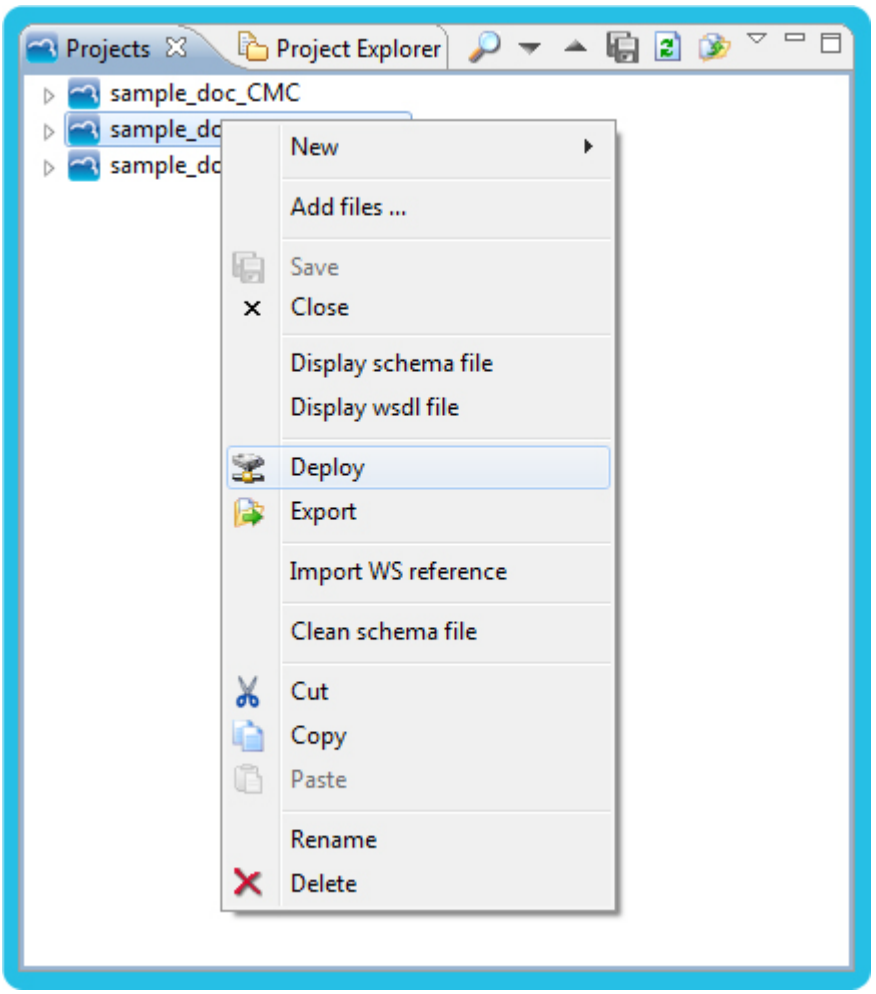

Figure 2 - 28: Deploying a Convertigo project to a Convertigo Server

A **Deploy a Convertigo project** window opens.

2 Enter the convertigo server address in the **Convertigo server** field (for example `localhost:28080` if the Server is installed on the same computer as the Studio) or select a previously used address in the drop-down list (if drop-down list is not empty).

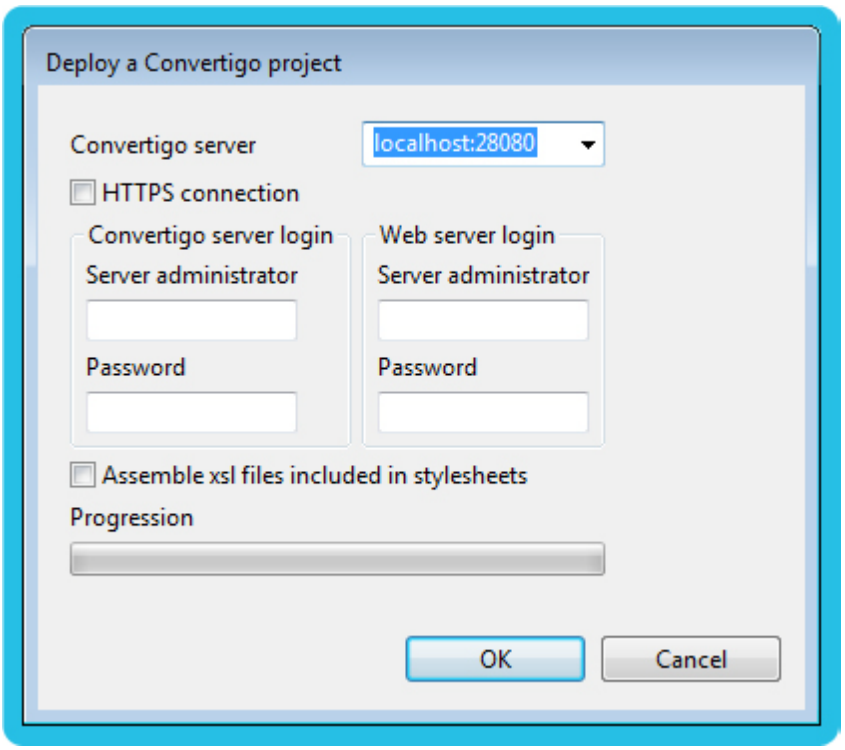

Figure 2 - 29: Entering a Convertigo Server address

3 Then, enter the Convertigo Server login credentials in the **Server administrator** and **Password** fields (for example `admin` and `admin` for a default local installation).

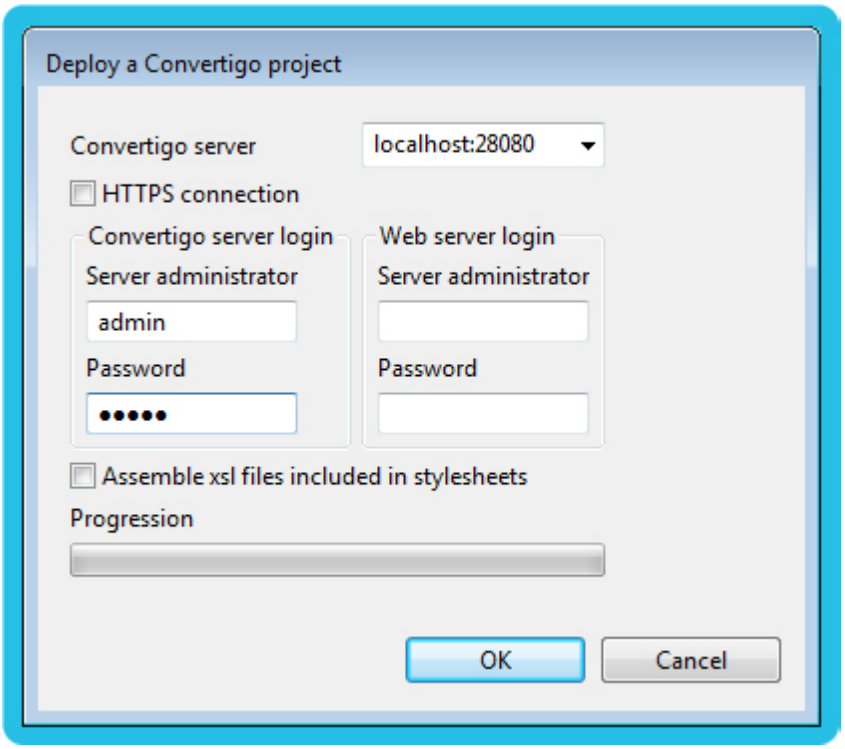

Figure 2 - 30: Entering the Convertigo Server login

4 Click on **OK**.

The progression status bar progresses as the project is deployed.

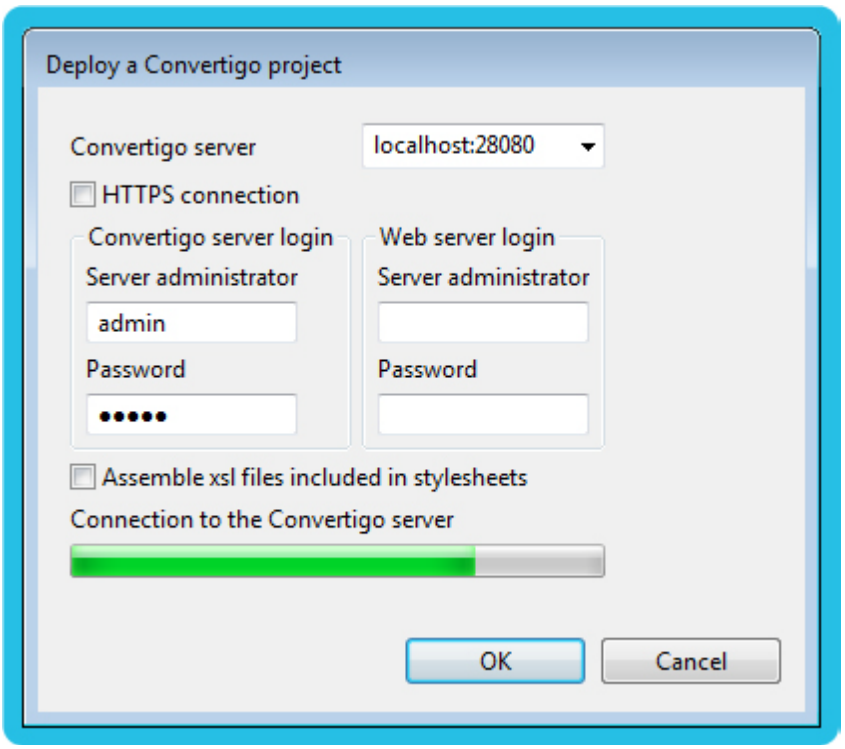

Figure 2 - 31: Progression status bar

A message pops-up, indicating that the project has been successfully deployed to the Convertigo Server.

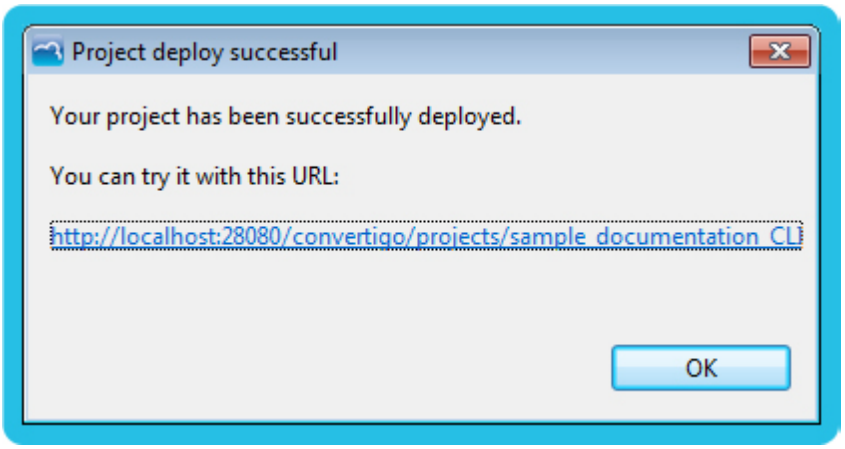

Figure 2 - 32: Confirmation message of deployed project

Repeat this procedure for the two left projects `sample_documentation_CWC` and `sample_doc_CMC`.

2.4 Setting Up the Mashup Composer Part of the Project - Developing the Composite Application

The following sections explain step by step how to set up your first Composite Application in Convertigo Mashup Composer:

- Switching to the Mashup Composer
- Declaring Widgets in Mashup Composer

- Creating a Page Including Three Widgets

2.4.1 Switching to the Mashup Composer

The following procedures describe how to access the Mashup Composer.

To access the Mashup Composer

1 Open a Web browser, then enter the following URL

```
http://<HostName>:<ConvertigoPortNumber>/portal
```

For example, in our project:

- *HostName* = `localhost` (local server installed);
- *ConvertigoPortNumber* = `28080` (default port for server).

For example, in a browser URL address bar:

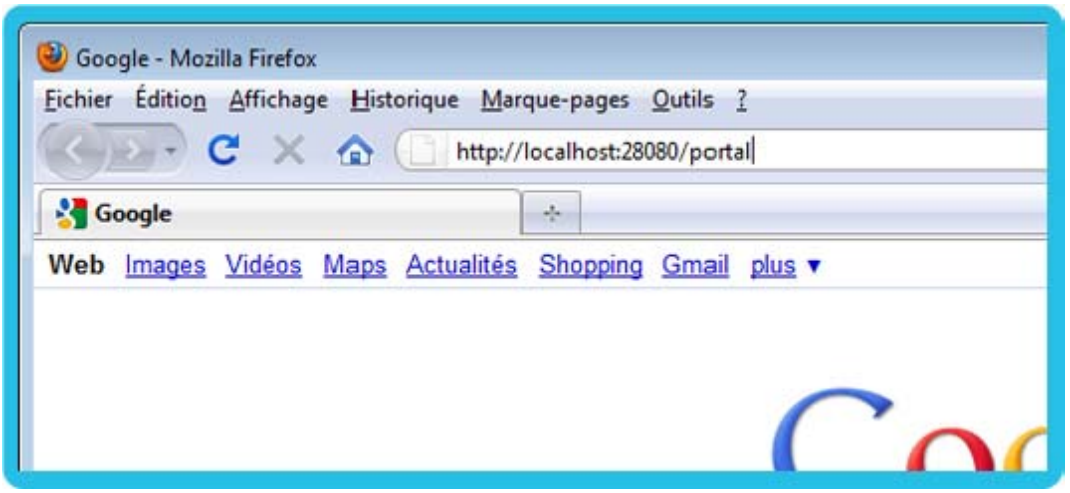

Figure 2 - 33: Mashup Composer URL in a Web browser

The Mashup Composer **Home** page opens:

Figure 2 - 34: Mashup Composer Home page

2 Click on the *Administrator* link to log in with an administrator user ID.

Figure 2 - 35: Logging in to the Mashup Composer as administrator

You must be logged in as administrator to create pages in Mashup Composer and compose your composite application.

A toolbar appears at the top of the page, with five menu items:

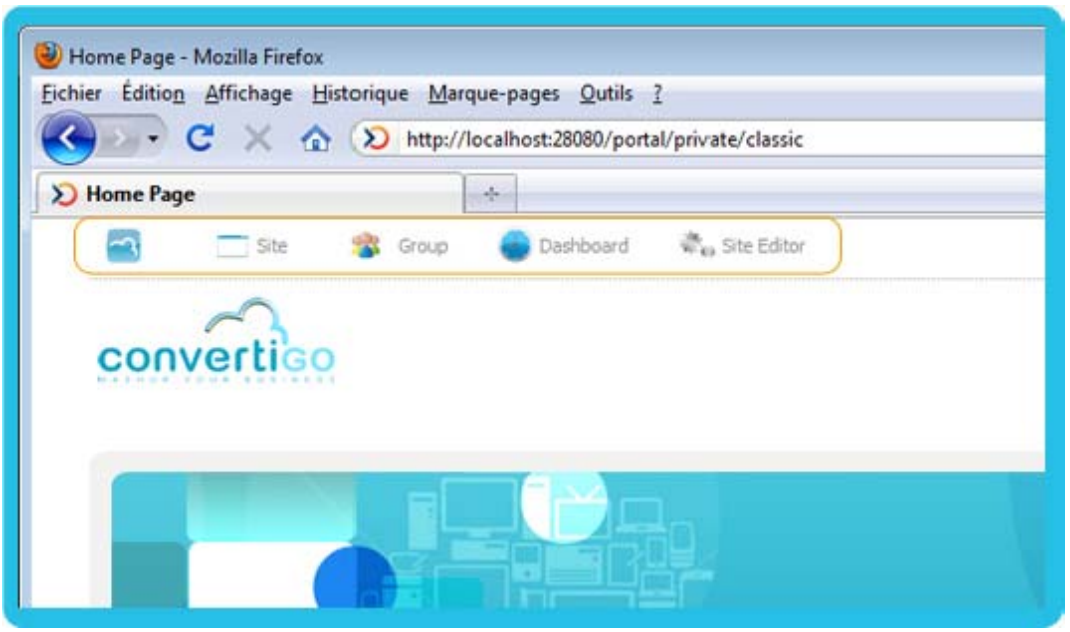

Figure 2 - 36: Toolbar available while logged in

You are now connected to the Mashup Composer.

WHAT COMES NEXT?

We have now logged in to the Mashup Composer.

The next step consists in declaring the widgets in Mashup Composer from the projects deployed in the Convertigo Server.

2.4.2 Declaring Widgets in Mashup Composer

This section explains the way to declare widgets in the Mashup Composer from the Convertigo Server test platform.

The widgets to declare are based upon some transactions of the projects that were previously created in the Convertigo Studio and deployed to the Convertigo Server.

Table 2 - 3: Sample project widgets - Transaction / project

Widget...	...is based upon ...	...set for the...
`GoogleClip`	`GoGoogleCom` transaction	web clipping `sample_documentation_CWC` project
`ListOfArticles`	`GetArticleData` transaction	legacy integration `sample_documentation_CLI` project
`InteractionHub`	the default transaction (named `DefaultHub`)	interaction hub `sample_doc_CMC` project

These data widgets are based upon transactions differing on a major point: one transaction declares a variable, one does not, and the last one is a transaction from a hub project (not a real executable transaction).

For any widget's creation, the process can be summed up as follows:

1 *Open* the ***Convertigo Server test platform*** in the a browser, pointing on the project containing the transaction used as basis for the widget's creation.

2 *Declare* the widget from the test platform to the Mashup Composer.

3 *Configure* the created widget.

The following procedures show how to create the above mentioned widgets, based upon previoulsy developped transaction / project, depending on whether the transaction uses a variable or not.

GoogleClip widget

ListOfArticles widget

InteractionHub widget

GOOGLECLIP WIDGET

The *GoogleClip* widget is based upon the `GoGoogleCom` transaction, a variable-free transaction. The first step is to open the test platform on the Convertigo Server for the `sample_documentation_CWC` project containing the relevant transaction for creating this widget.

To open the project's test platform on the server

1 Launch a standard Web browser.

2 In the URL address field, type in the project's URL in the following format:

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

For example, in our project:

- *`HostName`* = `localhost` (default installed server);
- *`ConvertigoPortNumber`* = `28080` (default port for server);
- *`PjName`* = `sample_documentation_CWC` (in our sample).

For example, in a browser URL address bar:

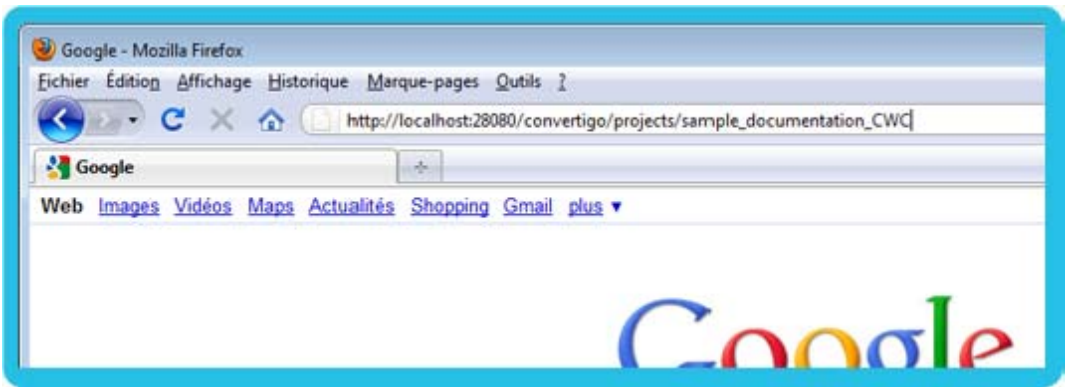

Figure 2 - 37: Convertigo Server test platform URL

The test platform of the project opens:

Figure 2 - 38: Test platform page for sample_documentation_CWC project

It displays:

- a left menu leading to Convertigo server administration, to several documentation sources, to the list of projects of the server, etc.

Figure 2 - 39: Test platform left menu

- a left part listing:
  - the **project name** (in our example, `sample_documentation_CWC`);
  - the **project description**;

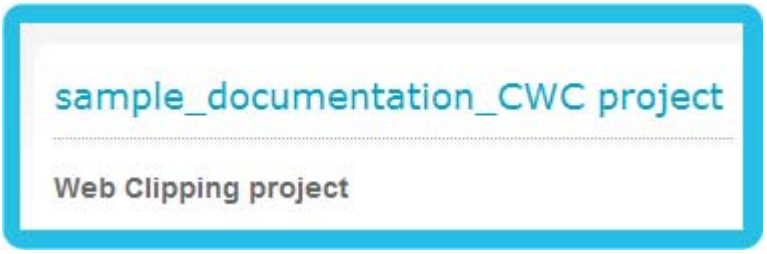

Figure 2 - 40: Test platform part showing project's details

  - the ***connector*** defined for the project (in our example, `GoogleConnector`);
  - all ***transactions*** defined for the connector (in our example, `DefaultWebClippingTransaction`, `GoGoogleCom` and `SearchGoogleClip`).

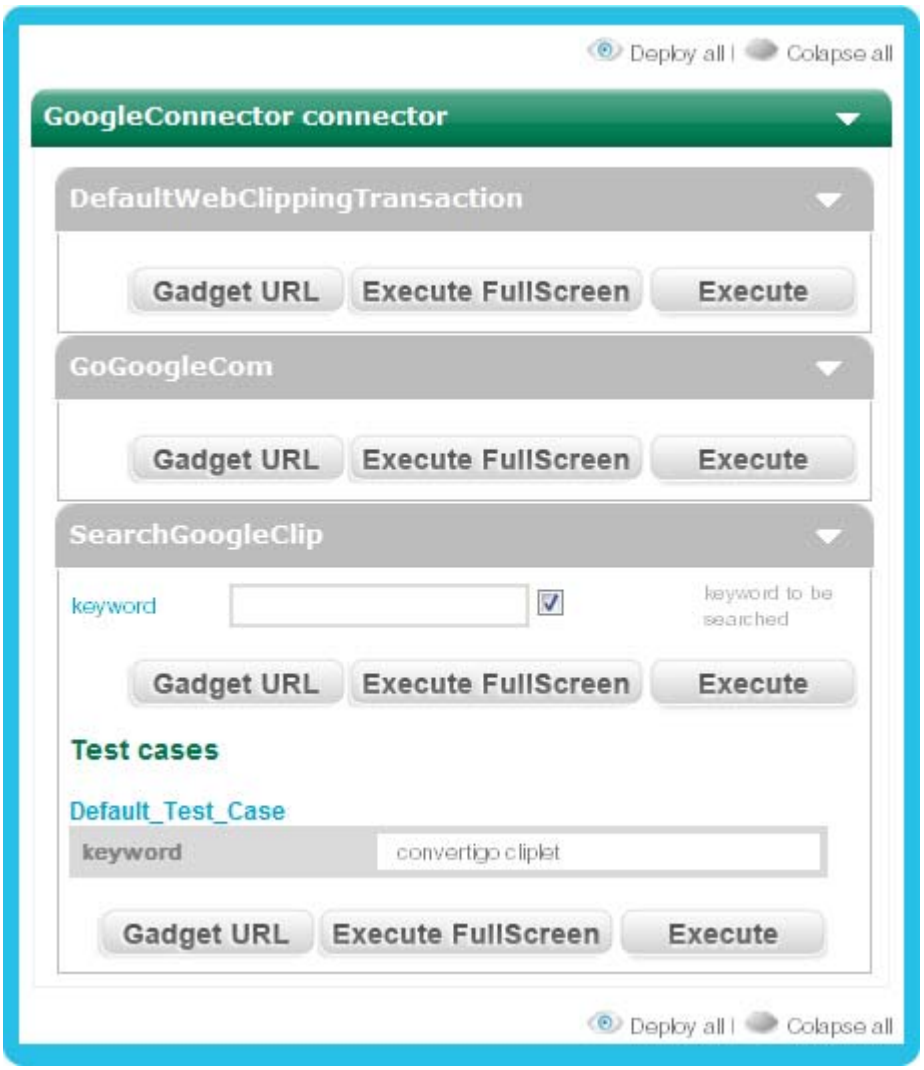

Figure 2 - 41: Test platform part showing project's transactions

- a rigth part welcoming the executed transaction's result (when such content exists).

Figure 2 - 42: Test platform part displaying transaction's result

For more information on how to open a project's t'est plateform and to test transactions in the test platfrom, refer to the "To test the transaction using the test platform" procedure (Chapter 2) of any existing Quick Guide.

The next step is to declare the widget based on the variable-free transaction in the Mashup Composer from the test platform.

To declare a widget based on a variable-free transaction from the test platform

1 In the test platform, when all content is expanded, each transaction is associated with three buttons: *Gadget URL*, *Execute FullScreen* and *Execute*. Expand the relevant transaction (for example `GoGoogleCom` transaction) and click on the ***Gadget URL*** button.

A result is created and displayed in the *Execution result* part of the test platform:

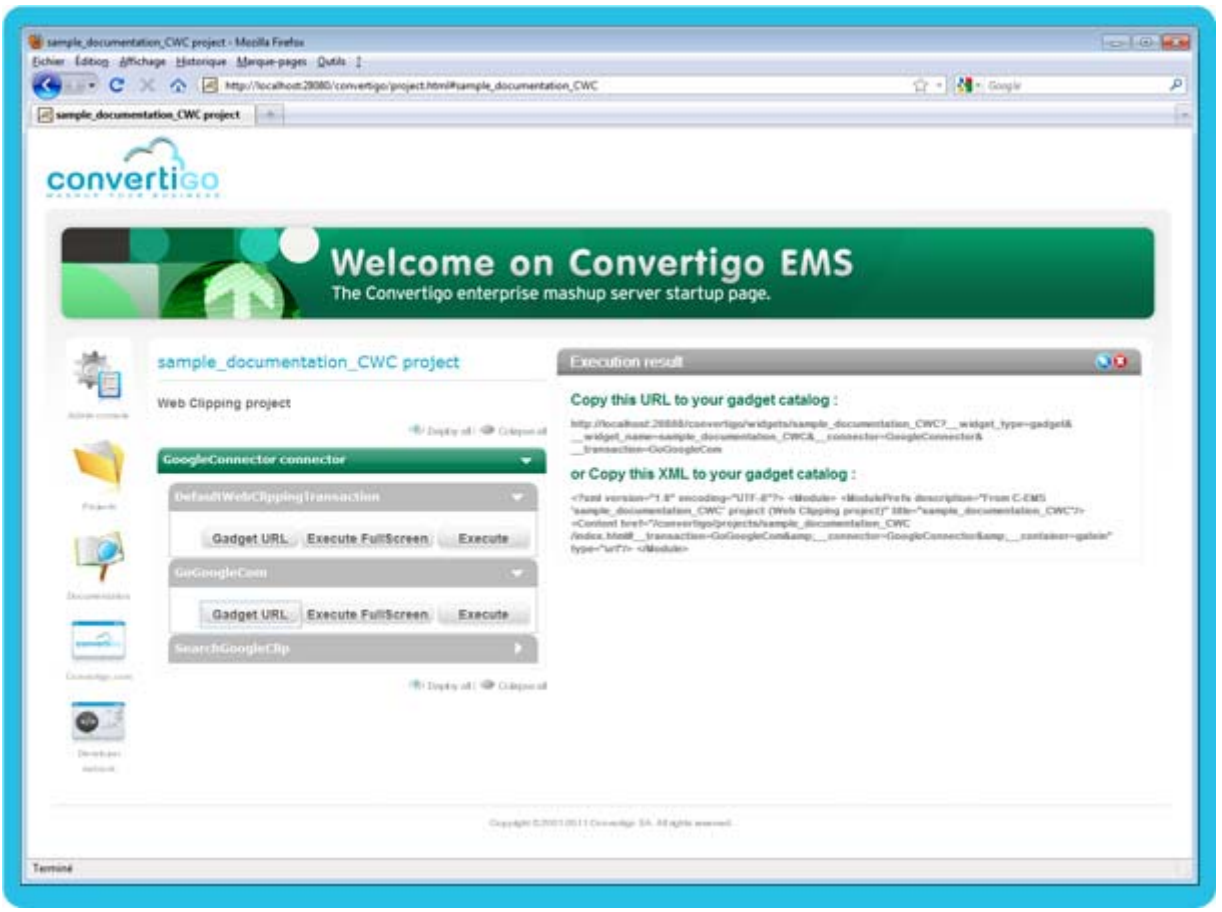

Figure 2 - 43: Declaring a widget from the GoGoogleCom transaction

2 There are two available contents in this result:

- a gadget URL,
- a gadget XML.

Copy the gadget XML content by selecting it and pressing the `Ctrl+C` key combination.

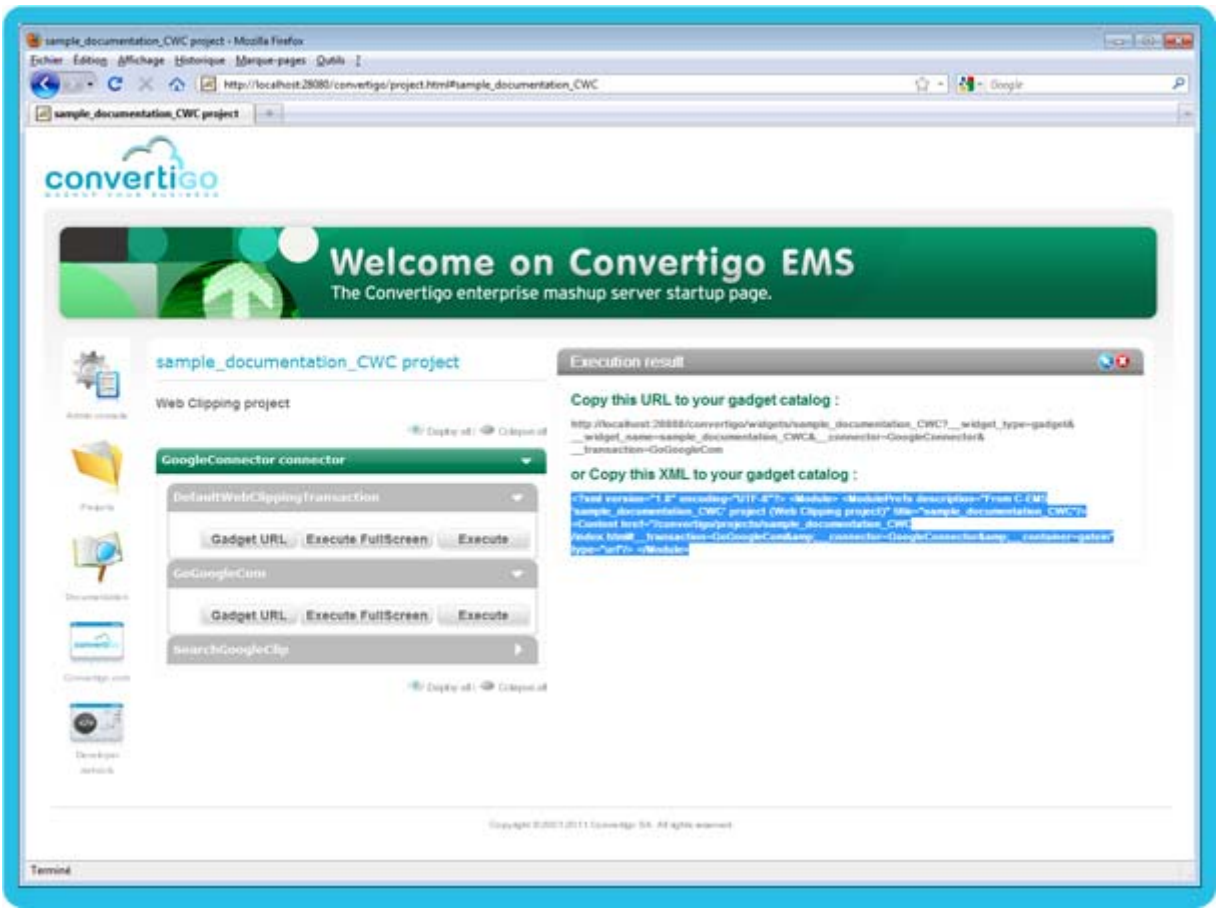

Figure 2 - 44: Copying GoGoogleCom gagdet XML

3 Switch to the Mashup Composer window (already opened).

To open the Mashup Composer (if not already opened), see "Switching to the Mashup Composer" on page 2-27.

4 Open the *Application Registry* administration page by clicking on the ***Group > Administration > Application Registry*** menu.

Figure 2 - 45: Opening the Application Registry administration page

The *Application Registry* administration page opens:

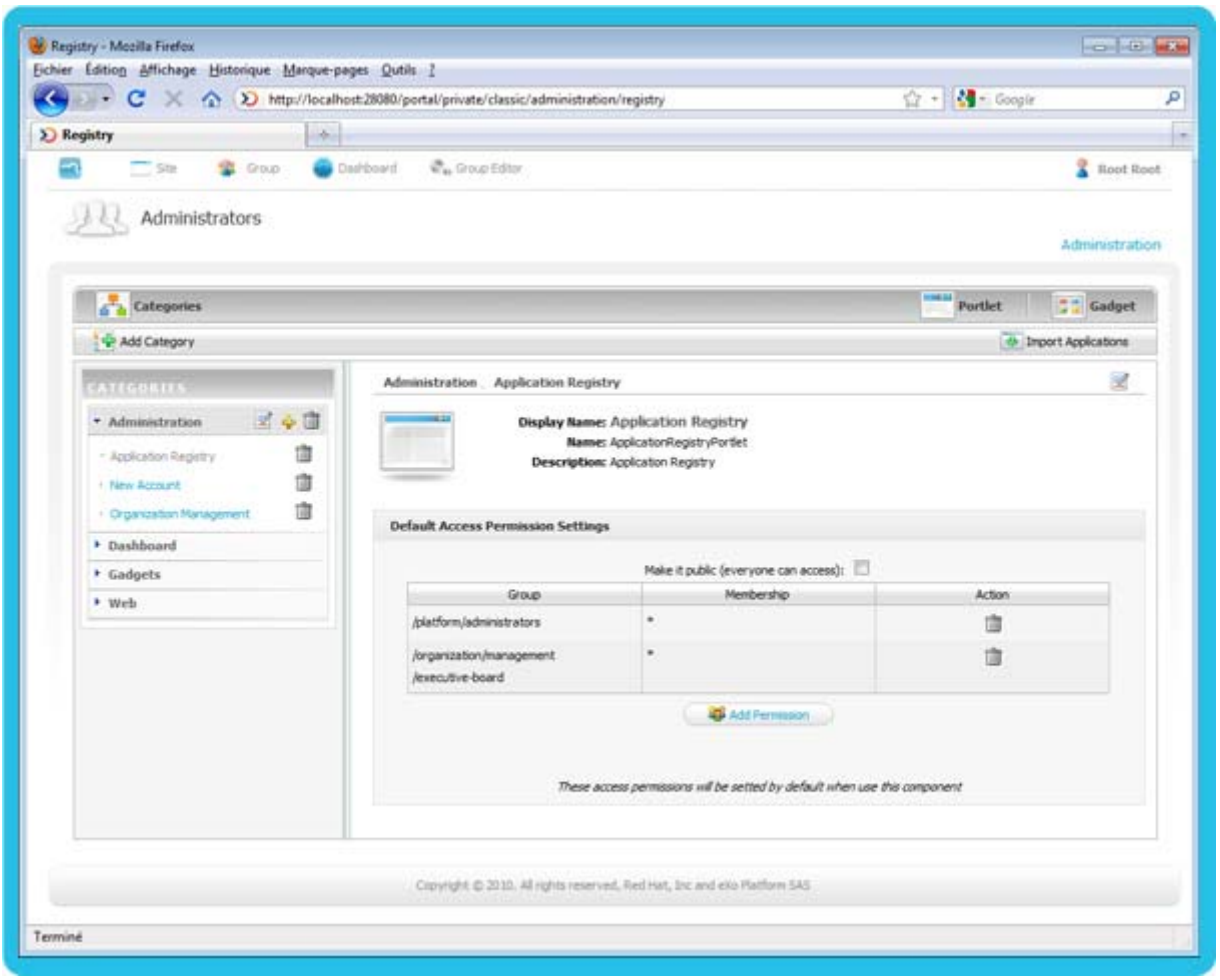

Figure 2 - 46: Application Registry administration page

5 In the *Application Registry*, click on the ***Gadget*** button, on the top right corner of the table.

The list of available gagdets is displayed on the left part of the page:

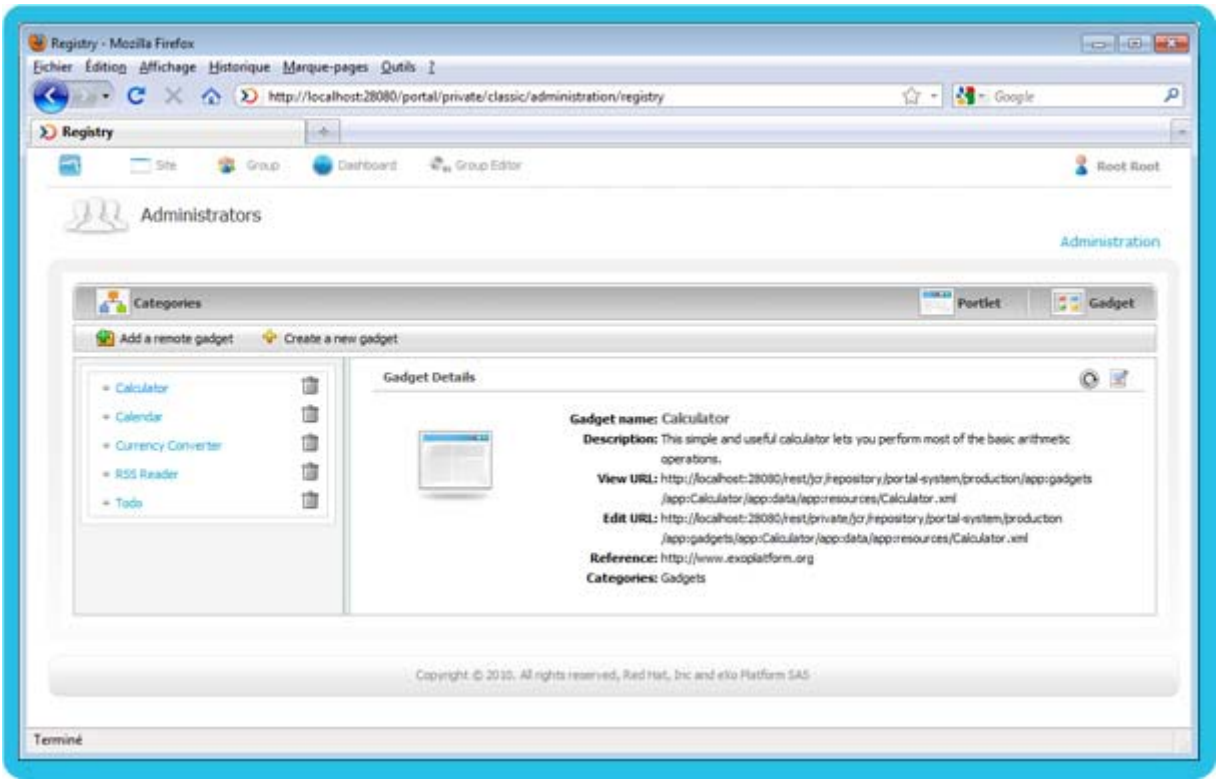

Figure 2 - 47: Accessing Gadget registry

6 Click on the ***Create a new gadget*** button.

A form is displayed, showing a sample gadget content in a *Source* text area.

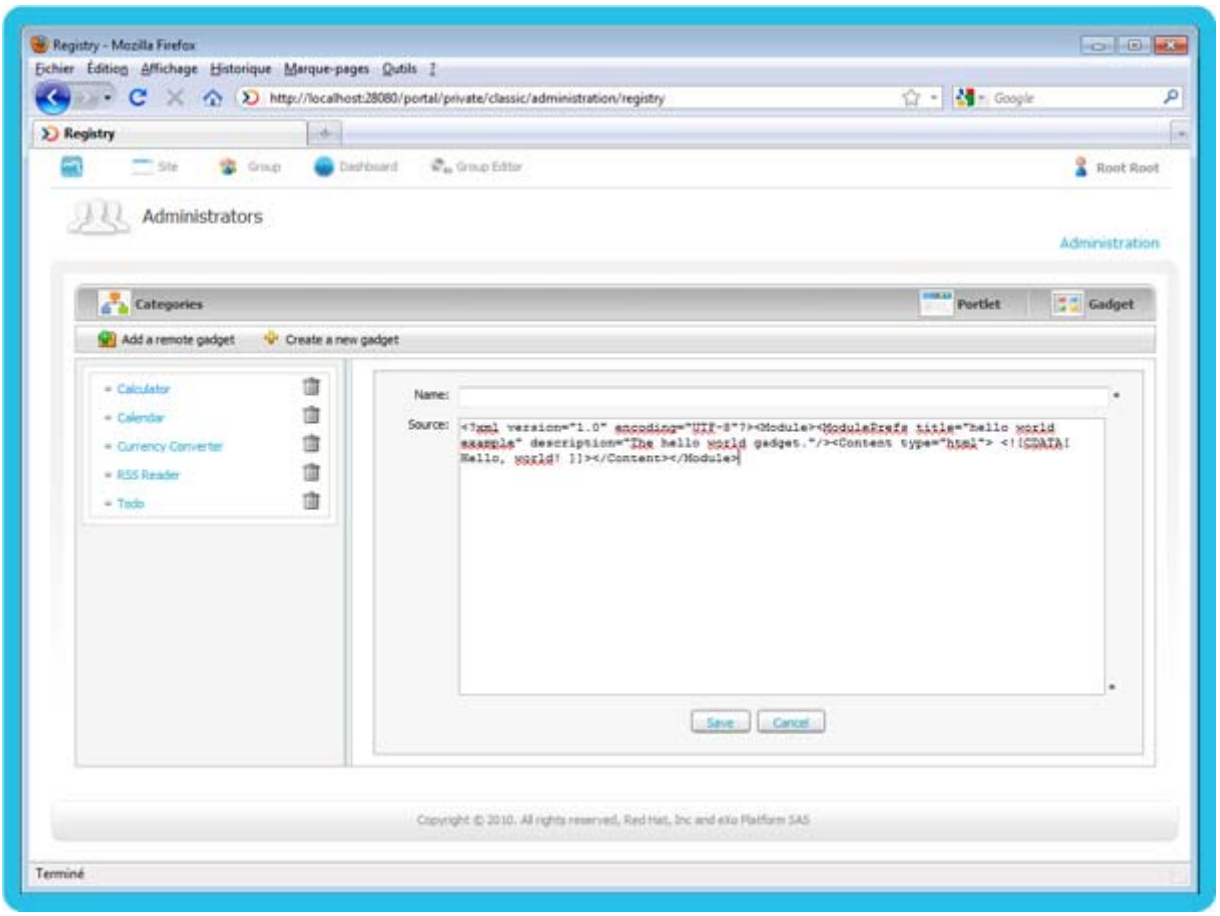

Figure 2 - 48: Creating a new gadget

7 Remove all content from *Source* text area and paste the gadget XML content from the test platform by pressing the `Ctrl+V` key combination.

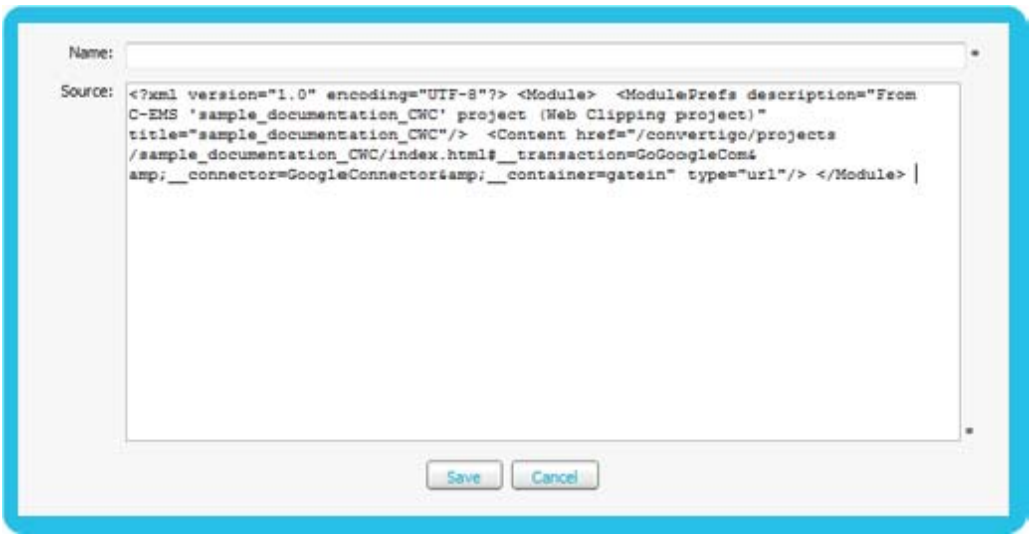

Figure 2 - 49: Pasting gadget content

8 The gadget XML content includes a widget's title set in the `title` attribute. Replace this `title` attribute's content and fill the **Name** field of the form with the widget's name (for example `GoogleClip`).

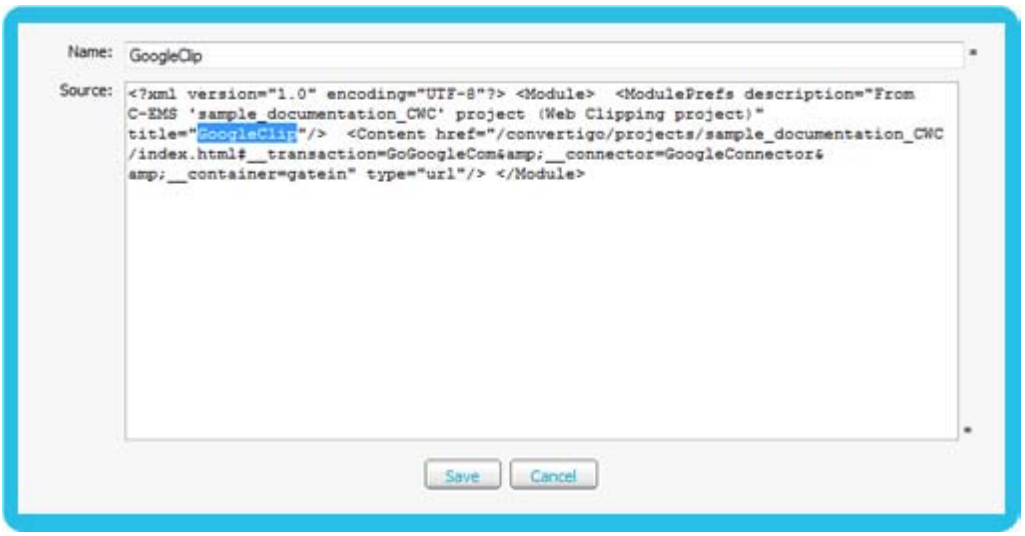

Figure 2 - 50: Setting gadget (widget) name

9 Validate the widget's declaration by clicking on the *Save* button.

A ***Gadget Details*** panel is displayed, summarizing the widget that has been created.

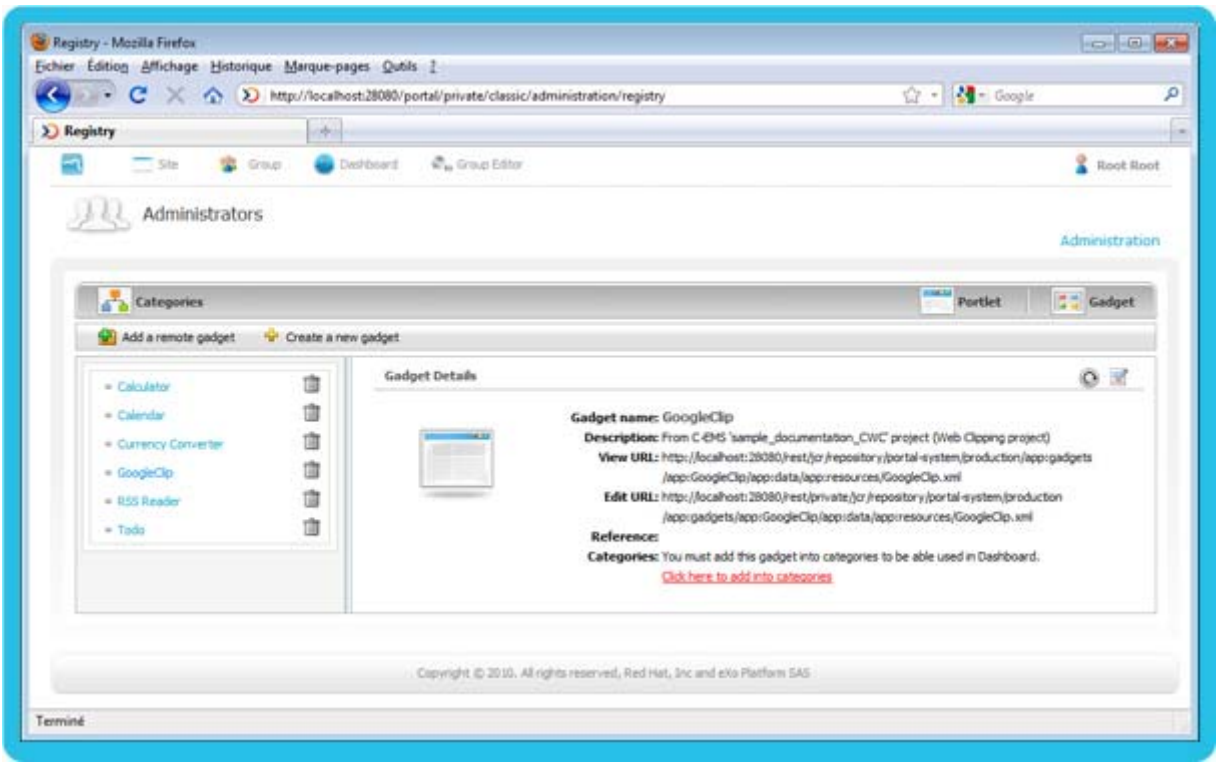

Figure 2 - 51: Gadget Details summary

The next step is to add the widget into at least one category to be able to use it later in a page.

To add a widget to a category

1 In the ***Gadget Details*** panel, click on the *Click here to add into categories* link.

2 A table is dynamically added at the bottom of the gadget summary, displaying available Categories.

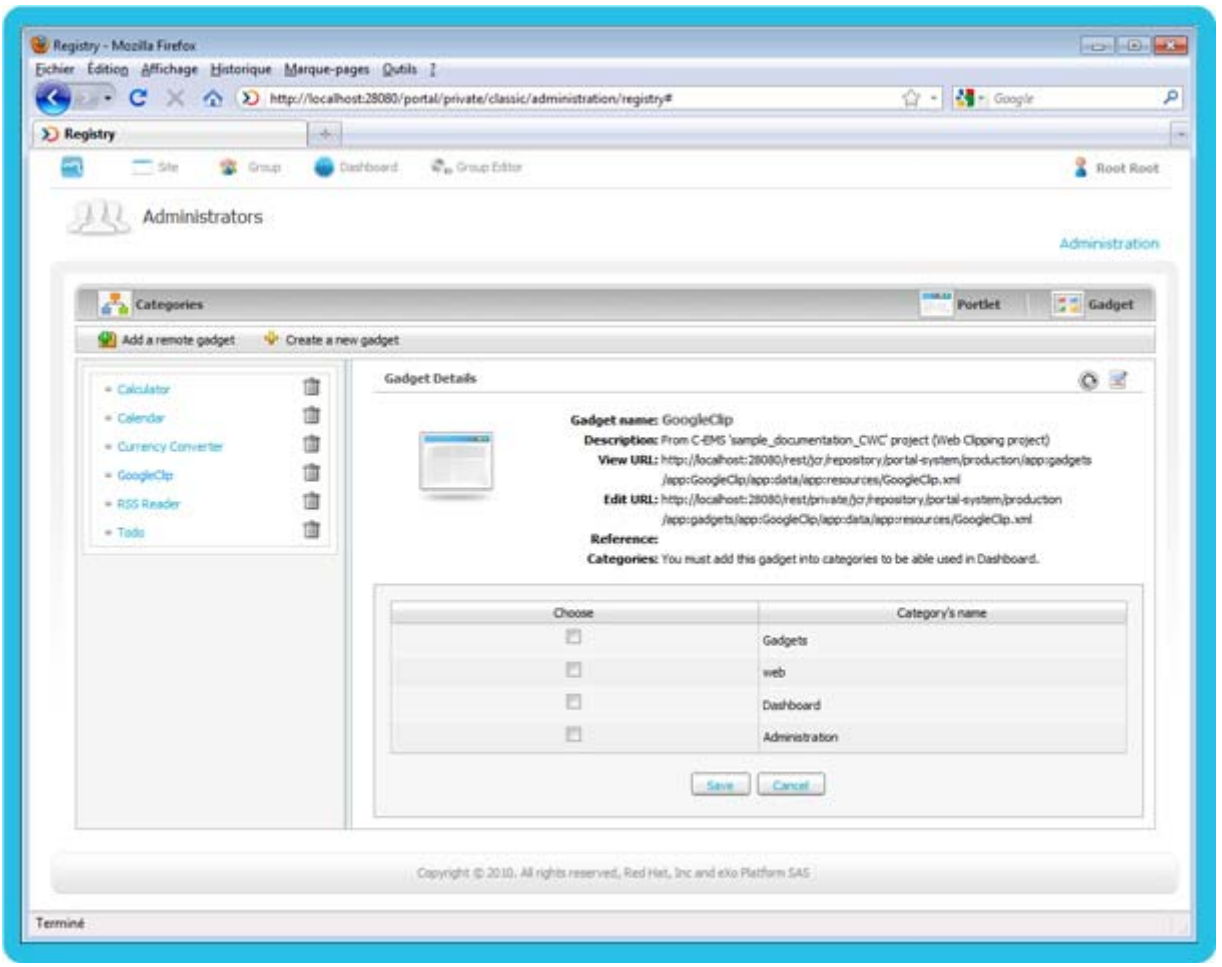

Figure 2 - 52: Gadget Details summary with available Categories

3 Check the *Choose* check box of the relevant categories (for example `Gadgets`).

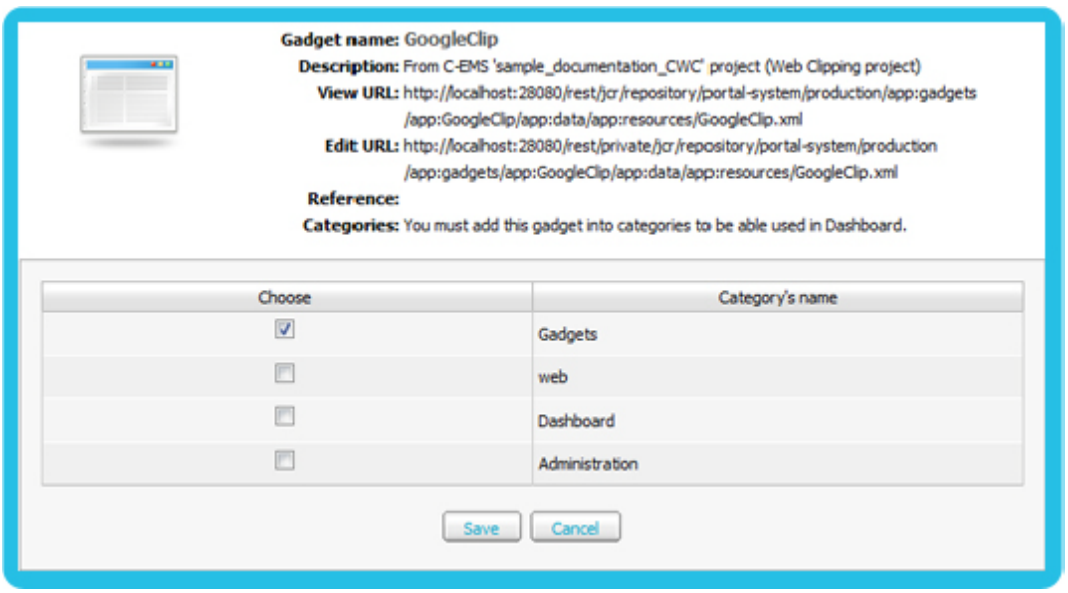

Figure 2 - 53: Adding Gadget to Categories

4 Click on the *Save* button to validate.

The *GoogleClip* widget is added in the Mashup Composer and correctly configured.

LISTOFARTICLES WIDGET

The *ListOfArticles* widget is based upon the `GetArticleData` transaction, a variable-based transaction. The first step is to open the test platform on the Convertigo Server for the `sample_documentation_CLI` project containing the relevant transaction for creating this widget.

To do so, repeat the steps of preceding procedure *"To open the project's test platform on the server"* on page 2 - 30 for the `sample_documentation_CLI` project.

The next step is to declare the widget based on the variable-based transaction in the Mashup Composer from the test platform.

To declare a widget based on a variable-based transaction from the test platform

In the test platform, when all content is expanded, each transaction is associated with three buttons: *Gadget URL*, *Execute FullScreen* and *Execute*. These buttons are also present for test cases defined on transactions.

1 Expand the relevant transaction (for example `GetArticleData` transaction). A test case named `Default_Test_Case` is defined for the transaction, fixing a value for the `article_no` variable.

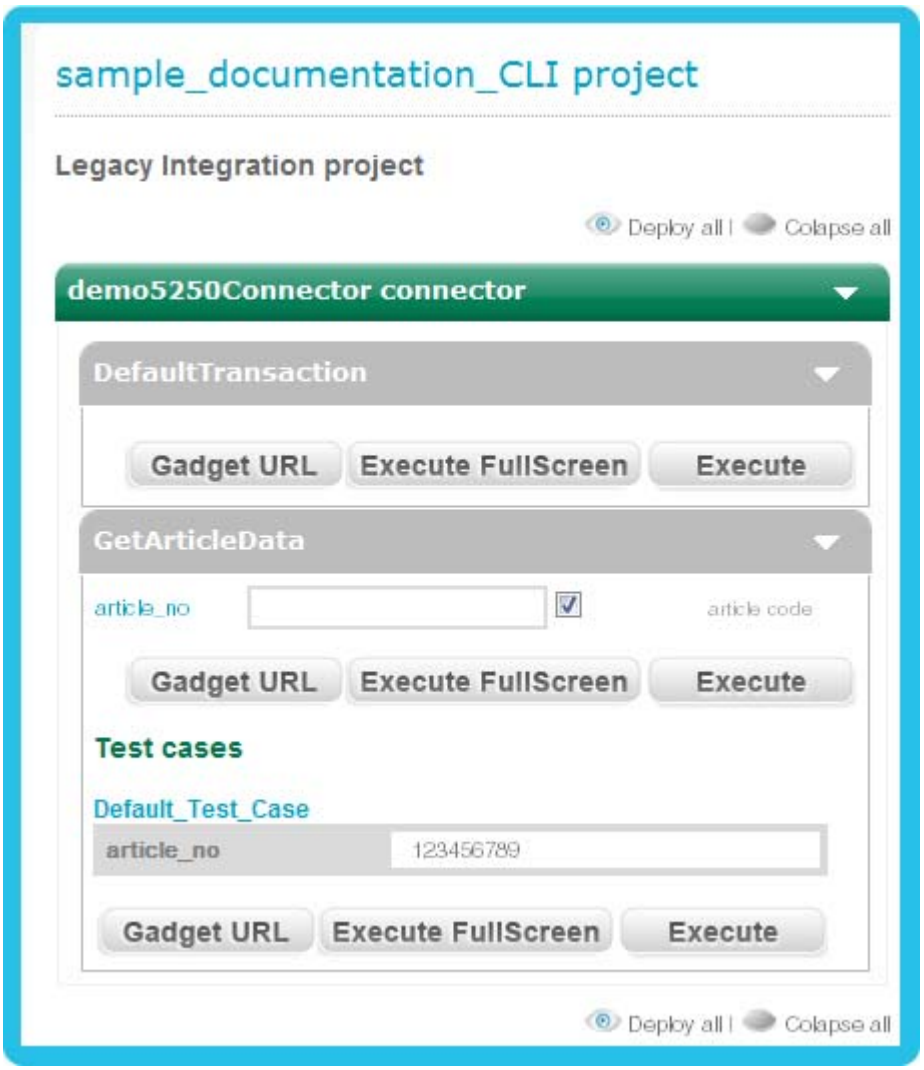

Figure 2 - 54: GetArticleData transaction in test platform

2 Then, two similar actions are possible:

- fill the `article_no` input and click on the ***Gadget URL*** button of the transaction, or
- click directly on the ***Gadget URL*** button of the defined test case.

A result is created and displayed in the *Execution result* part of the test platform:

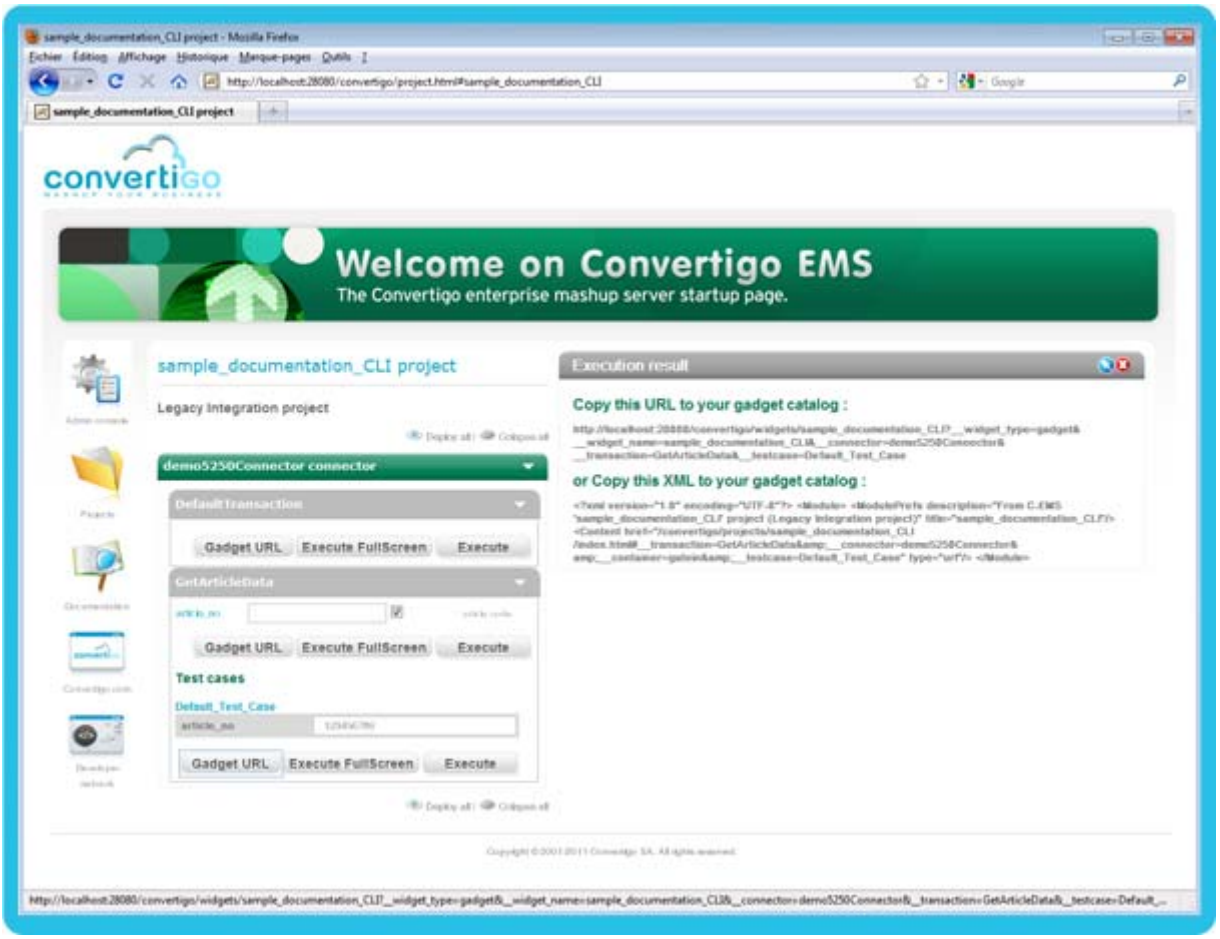

Figure 2 - 55: Declaring a widget from the GetArticleData transaction

3 Then follow steps 2 to 9 of the procedure *"To declare a widget based on a variable-free transaction from the test platform"* on page 2 - 34.

The next step is to add the widget into at least one category to be able to use it later in a page. To do so, follow the procedure *"To add a widget to a category"* on page 2 - 40.

The *ListOfArticles* widget is added in the Mashup Composer and correctly configured.

INTERACTIONHUB WIDGET

The *InteractionHub* widget is based upon the default transaction of an interaction hub project. The first step is to open the test platform on the Convertigo Server for the `sample_doc_CMC` project, interaction hub project used as basis for creating this widget.

To do so, repeat the steps of preceding procedure *"To open the project's test platform on the server"* on page 2 - 30 for the `sample_doc_CMC` project.

The next step is to declare the widget based on the default transaction in the Mashup Composer from the test platform.

To declare a widget based on an interaction hub default transaction from the test platform

1 In the test platform, when all content is expanded, each transaction is associated with three buttons: *Gadget URL*, *Execute FullScreen* and *Execute*. The interaction hub project contains at least one default transaction named `DefaultHub`. Expand the transaction and click on the ***Gadget URL*** button.

A result is created and displayed in the *Execution result* part of the test platform:

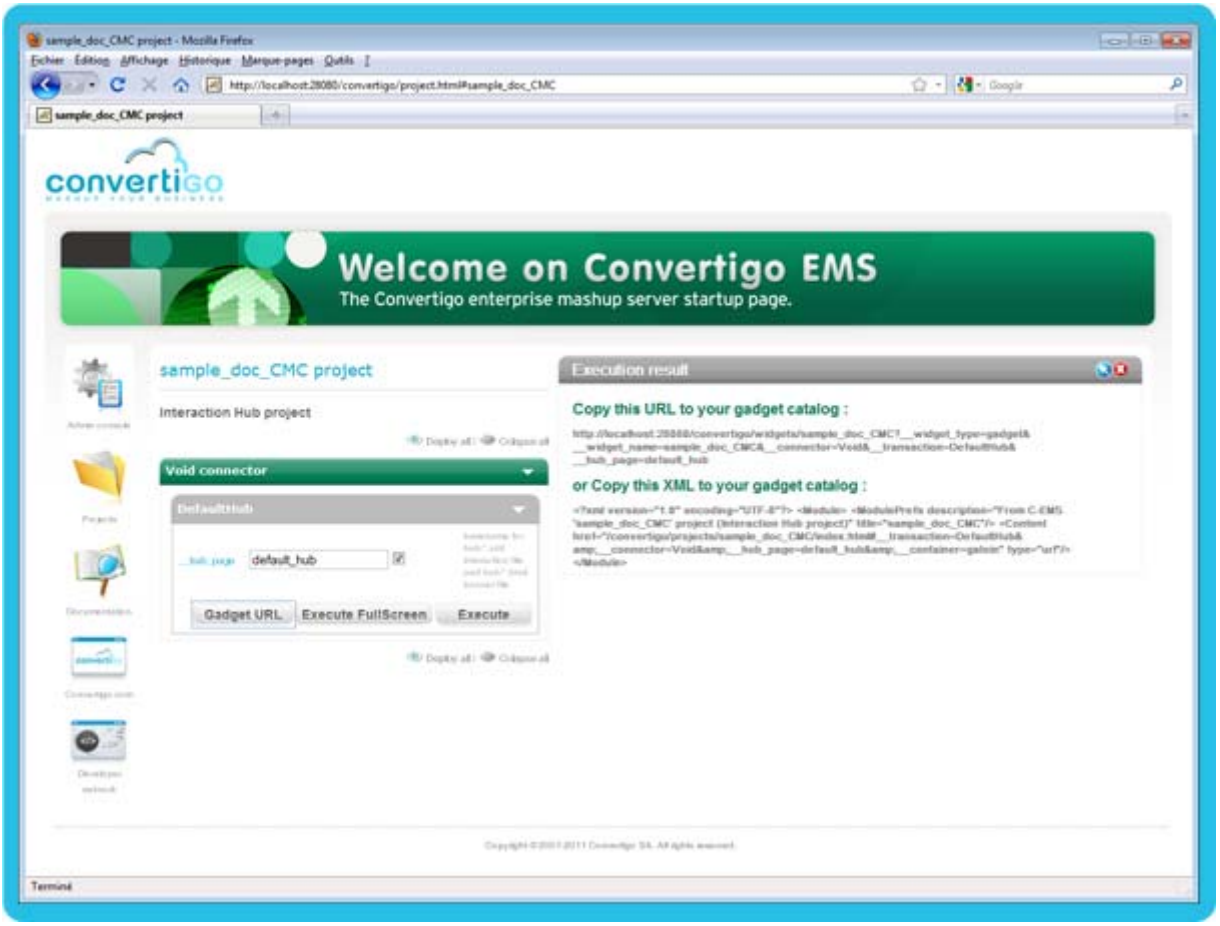

Figure 2 - 56: Declaring a widget from the DefaultHub transaction

Then follow steps 2 to 9 of the procedure *"To declare a widget based on a variable-free transaction from the test platform"* on page 2 - 34.

The next step is to add the widget into at least one category to be able to use it later in a page. To do so, follow the procedure *"To add a widget to a category"* on page 2 - 40.

The *InteractionHub* widget is added in the Mashup Composer and correctly configured.

WHAT COMES NEXT?

We have now declared the widgets in the Mashup Composer.

The next step consists in creating a page displaying the three previously created widgets, and arranging its layout.

2.4.3 Creating a Page Including Three Widgets

To create a properly laid out page including widgets, three steps must be completed:

- *Create* a new empty page.
- *Add* widgets (for example `InteractionHub`, `ListOfArticles` and `GoogleClip`) in the created page.
- *Arrange* widget layout on the page in a user-friendly way.

This is done in the next procedures.

To create a new empty page in the Mashup Composer

1 In the Mashup Composer, display the site's *Home* page by clicking on the ***Site > Classic > Home*** menu:

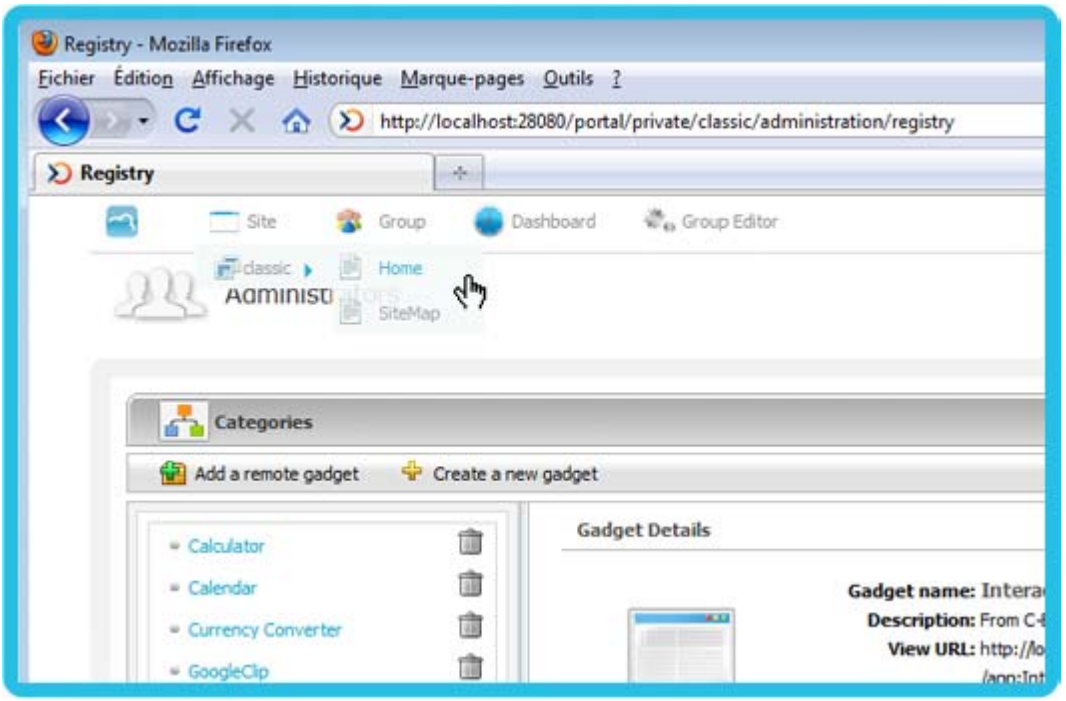

Figure 2 - 57: Navigating to site's Home page

2 Add a new page by clicking on the ***Site Editor > Add New Page*** menu:

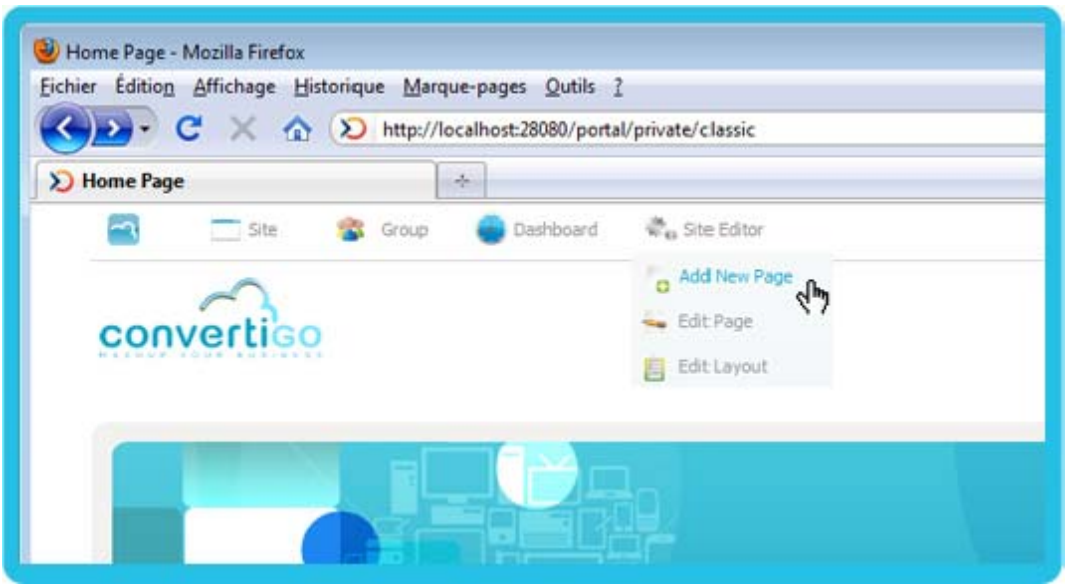

Figure 2 - 58: Adding a new page

The *Page creation wizard* opens:

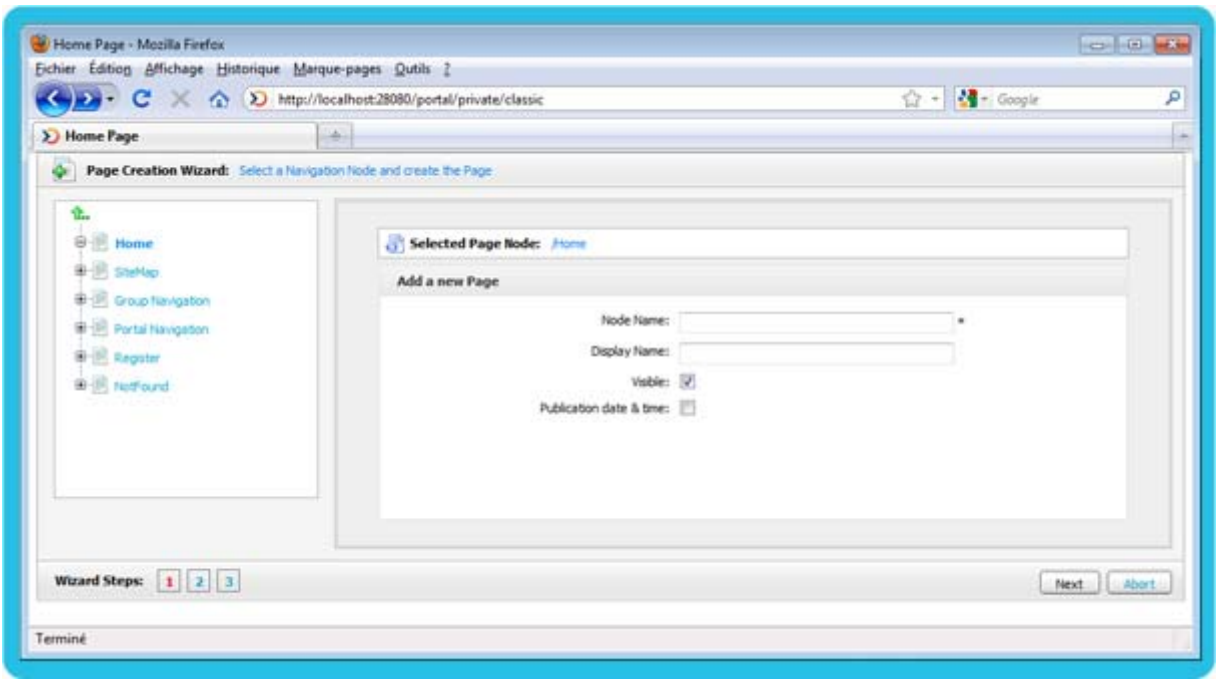

Figure 2 - 59: Page creation wizard

3 In the left panel of the wizard's first page, select a *Navigation Node* defining where the page has to be added (for example select **Home** node).

4 In the right panel of the wizard's first page, enter a page name in the **Node Name** field (for example `ArticlesDataMashup`) and in the **Display Name** field (for example `Articles data mashup`).

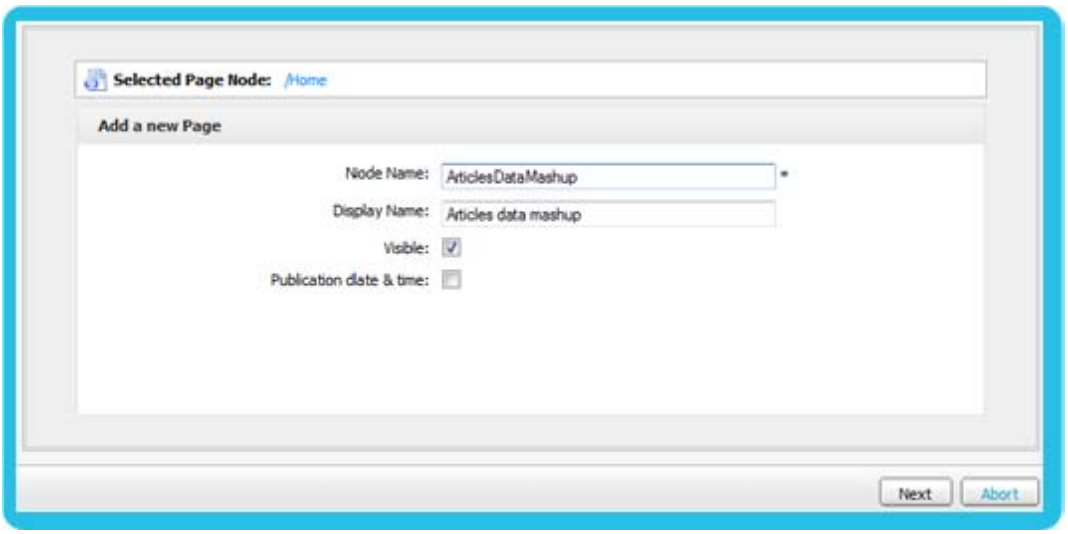

Figure 2 - 60: Filling page name and display name

The node name is an identifier, it is not displayed on screen. Only the display name is displayed.

5 Click on the **Next** button.

6 On the wizard's second page, select a layout for the created page. As we want to create an empty page, select **Empty layout**:

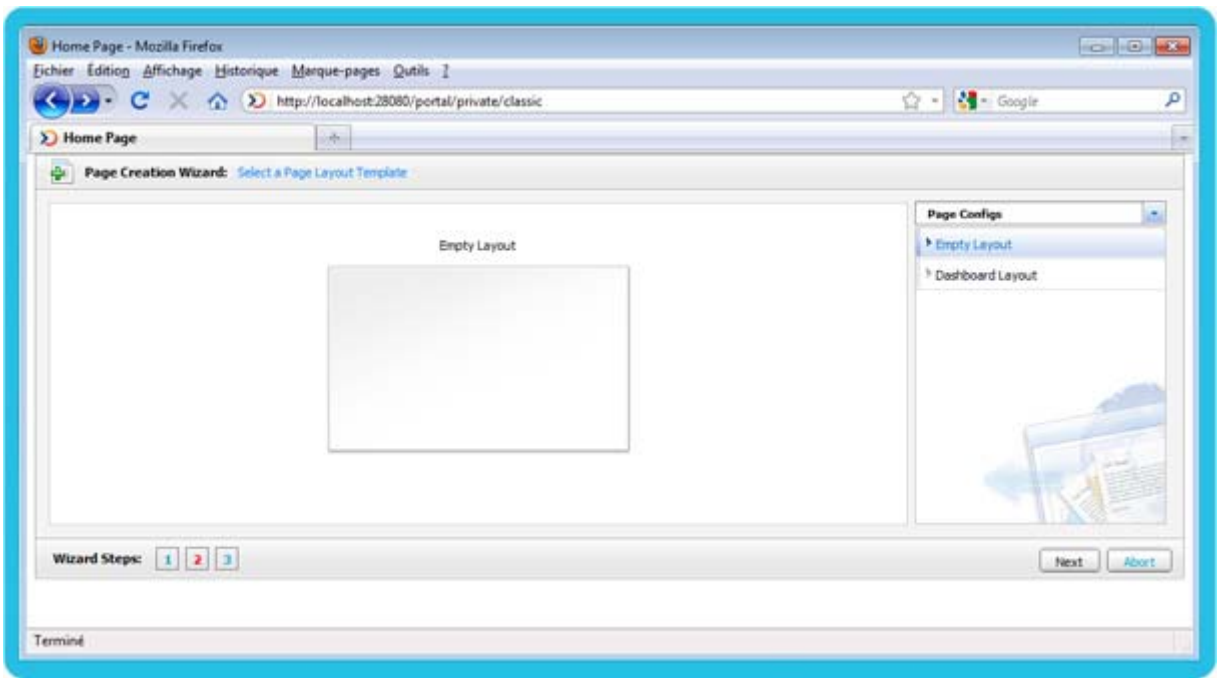

Figure 2 - 61: Selecting empty layout for the page

7 Click on the **Next** button.

The wizard's third page allows you to add widgets on the newly created page and to arrange the page's layout. These actions will be realized in next procedure.

8 Click on the **Finish** button of the ***Page Editor*** to finish creating the new empty page:

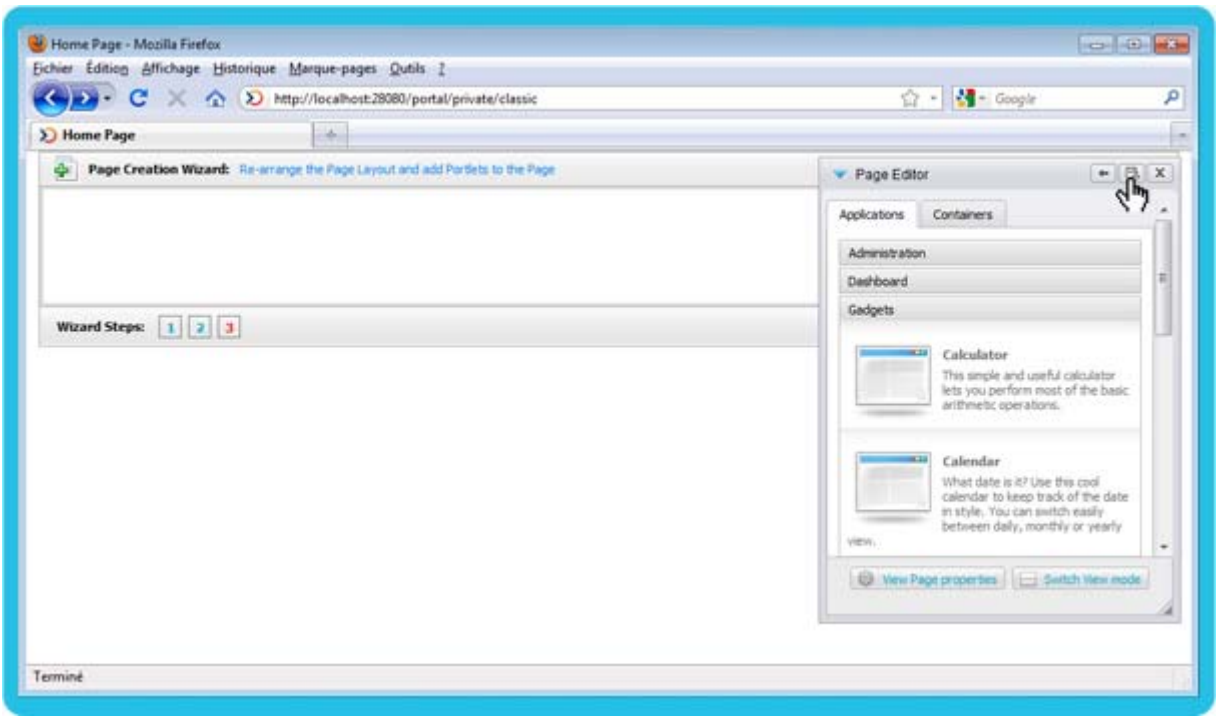

Figure 2 - 62: Clicking on finish button

The new empty page is created and displayed.

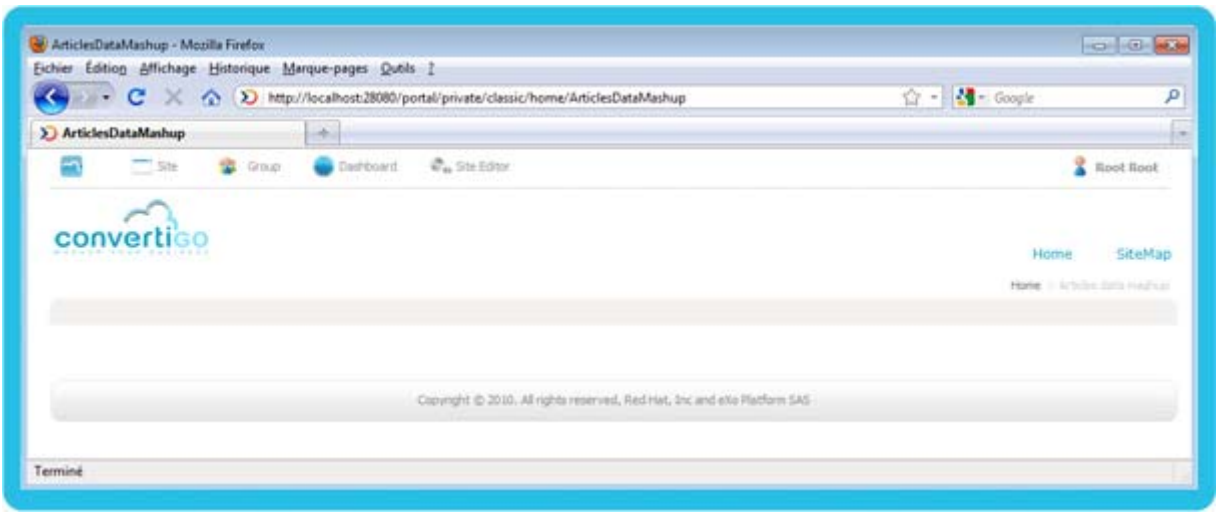

Figure 2 - 63: Newly created empty page

The *Articles data mashup* page is created and empty. The following procedures explain how to add widgets to the page and how to arrange the widgets' layout in the page using the ***Page Editor***.

To open the Page Editor to edit a page

1 On the desired page, open the ***Page Editor*** by clicking on the ***Site Editor > Edit Page*** menu:

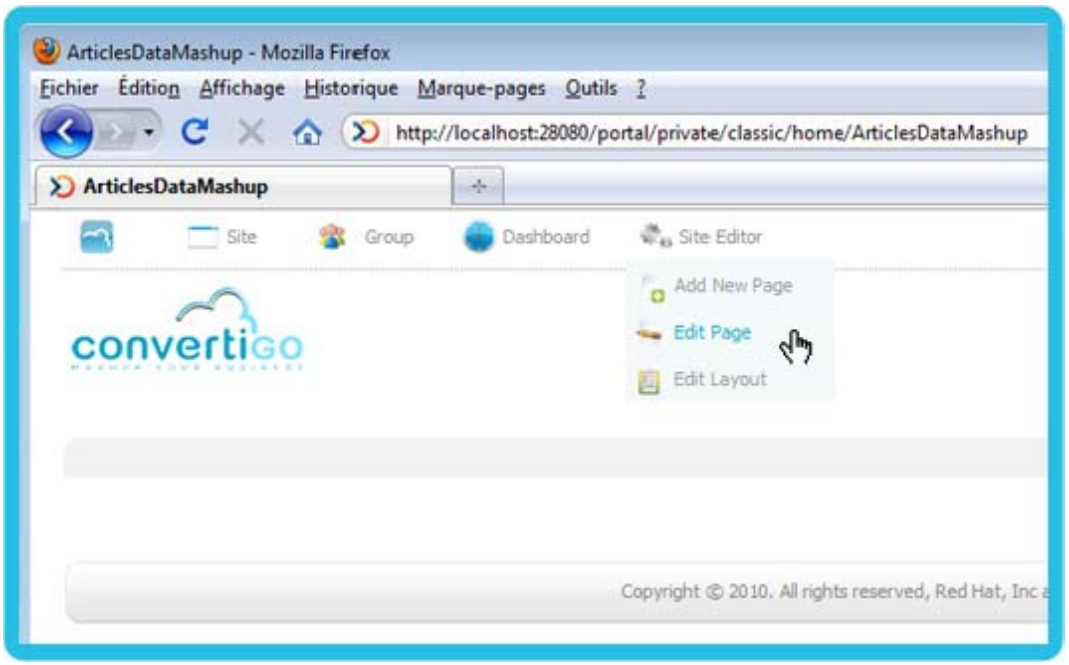

Figure 2 - 64: Opening Page Editor

The ***Page Editor*** opens. The left part of the ***Page Editor*** shows a representation of the future page. The right part of the ***Page Editor*** displays the ***Page Editor*** toolbar. This toolbar can be moved in the screen, it contains every necessary tool to add widgets on the page and to arrange the widgets' layout in the page.

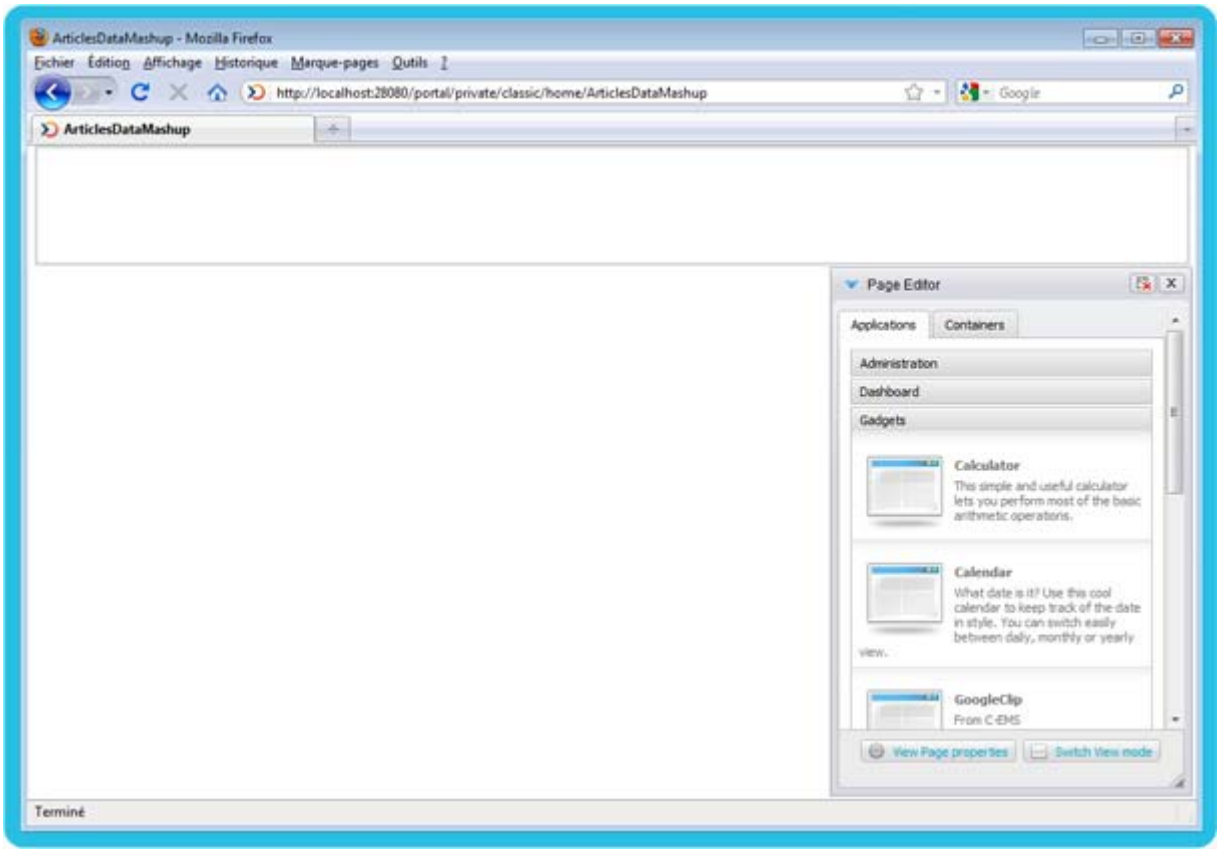

Figure 2 - 65: Page Editor

To add widgets to a page

1 In the toolbar, expand **Gadgets** category, and find the previously added widgets (`GoogleClip`, `ListOfArticles` and `InteractionHub`).

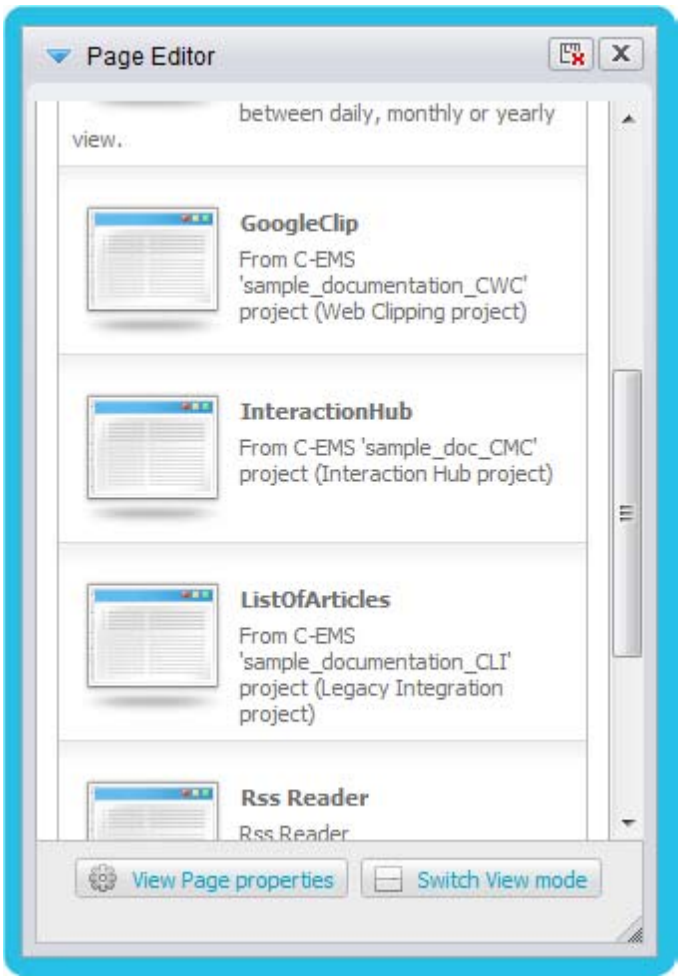

Figure 2 - 66: Finding widgets in Gadgets category

2 Drag-and-drop a first widget to the page to add it (for example drag-and-drop `GoogleClip` widget):

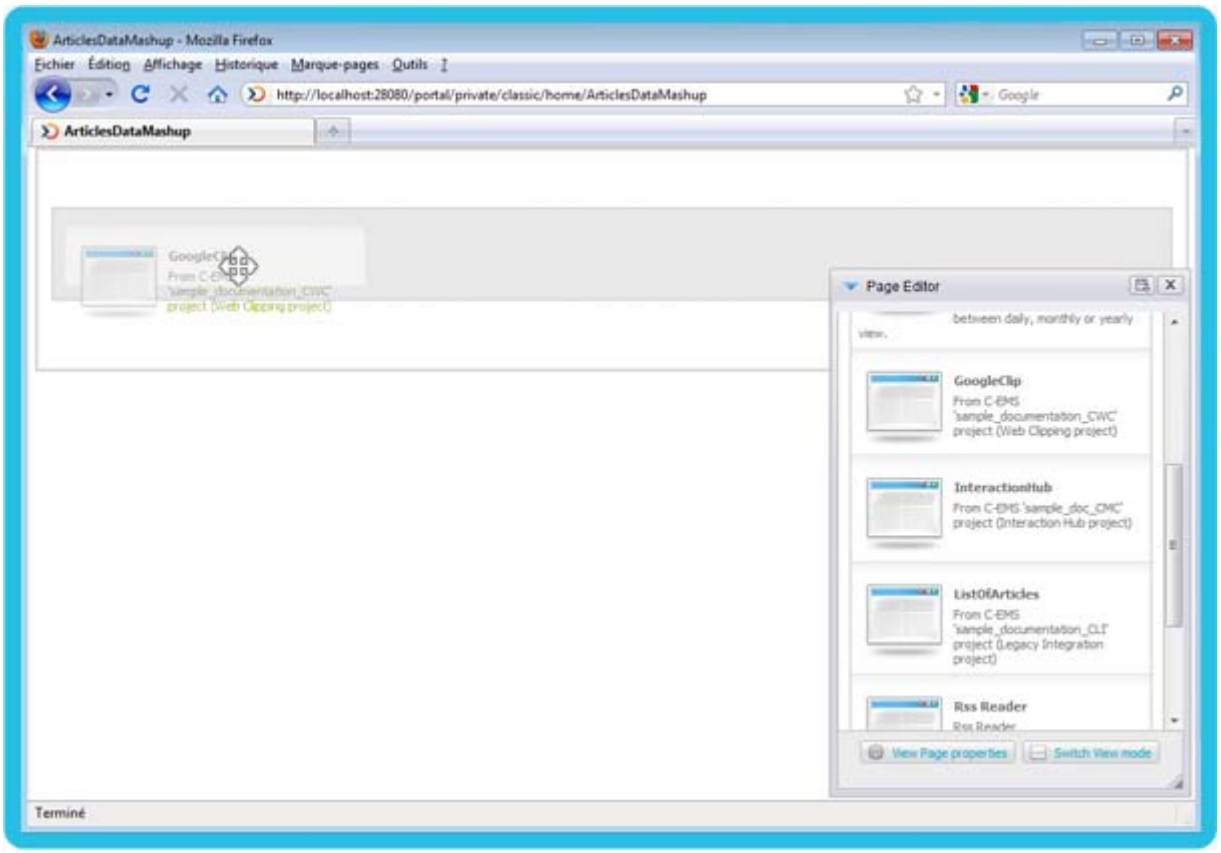

Figure 2 - 67: Adding GoogleClip widget to the page

The widget is added in the page, without any specific layout for now.

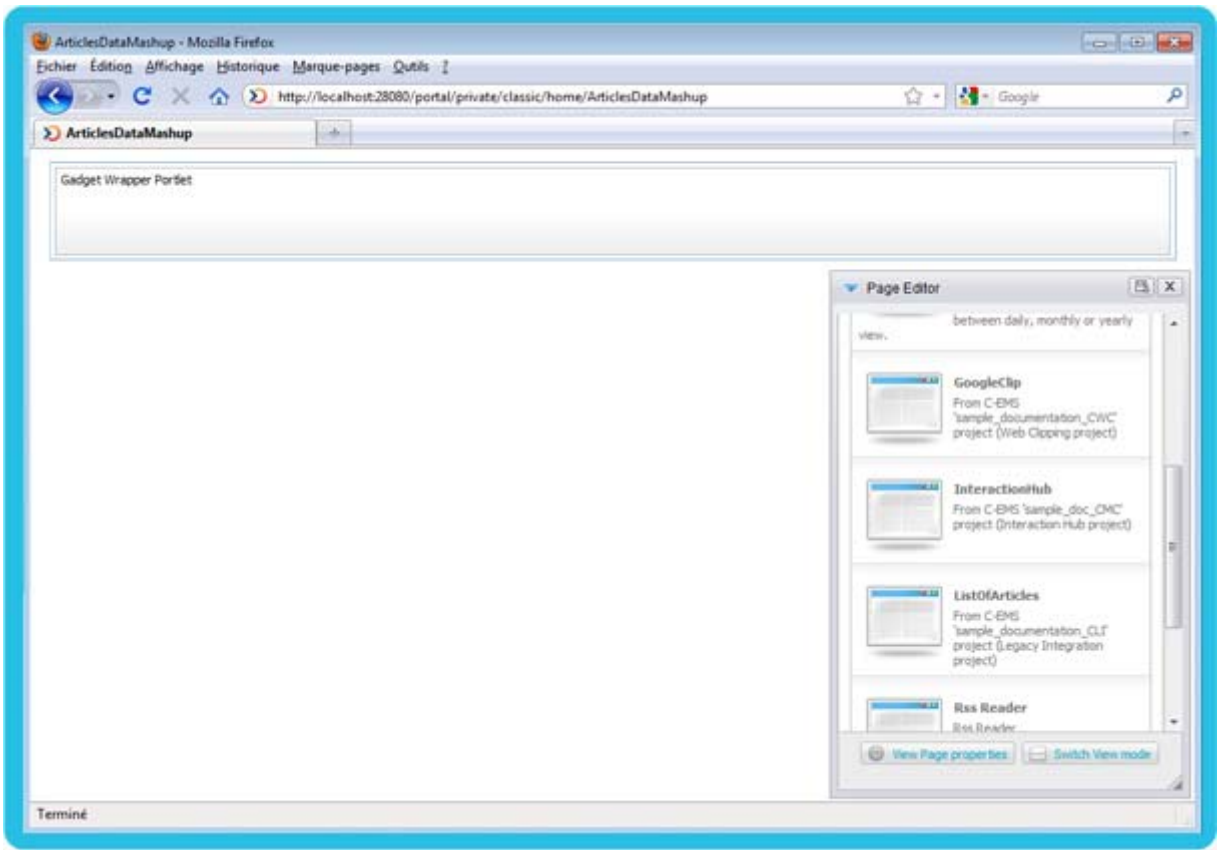

Figure 2 - 68: GoogleClip widget in the page

The next step consists in adding the second and third widgets to the page.

3 Repeat step 2 of this procedure for:

- the `ListOfArticles` widget,
- the `InteractionHub` widget.

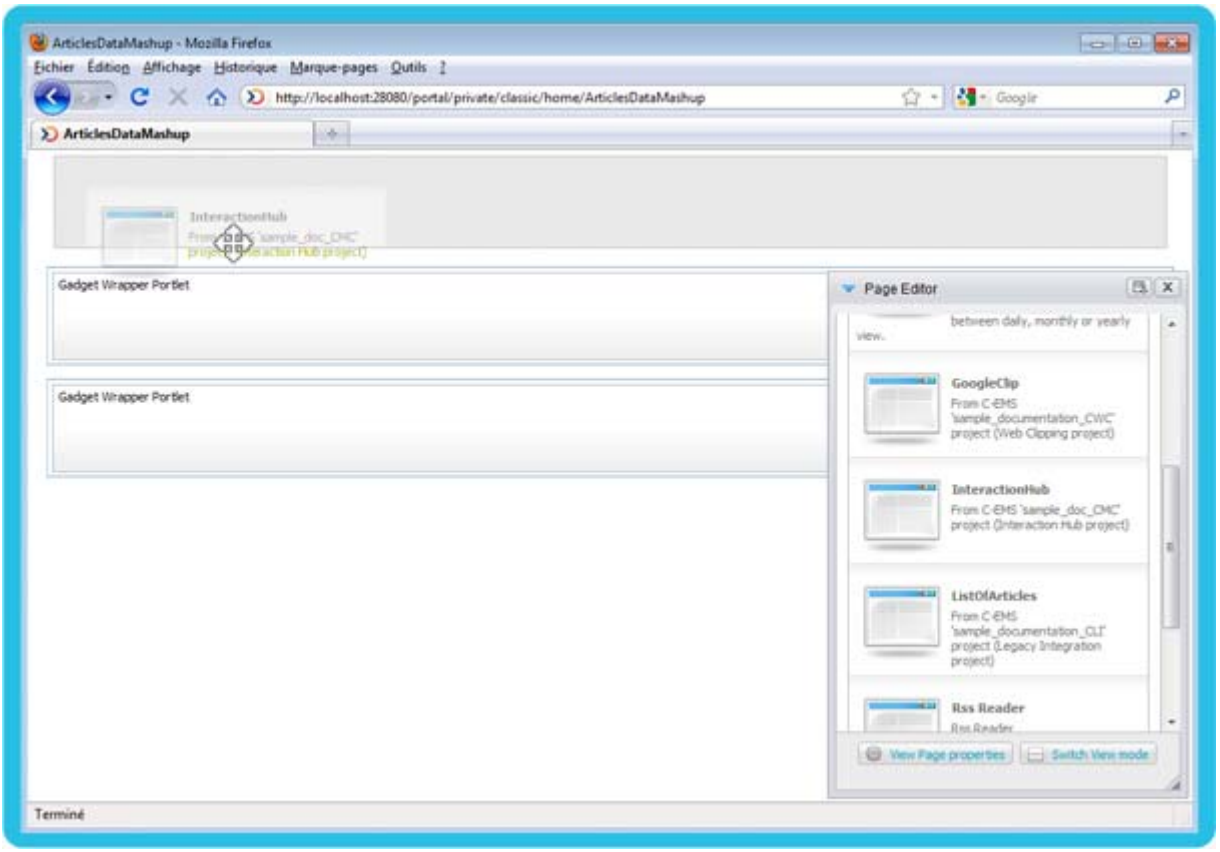

Figure 2 - 69: Adding interactionHub widget to the page

The three widgets are added to the page.

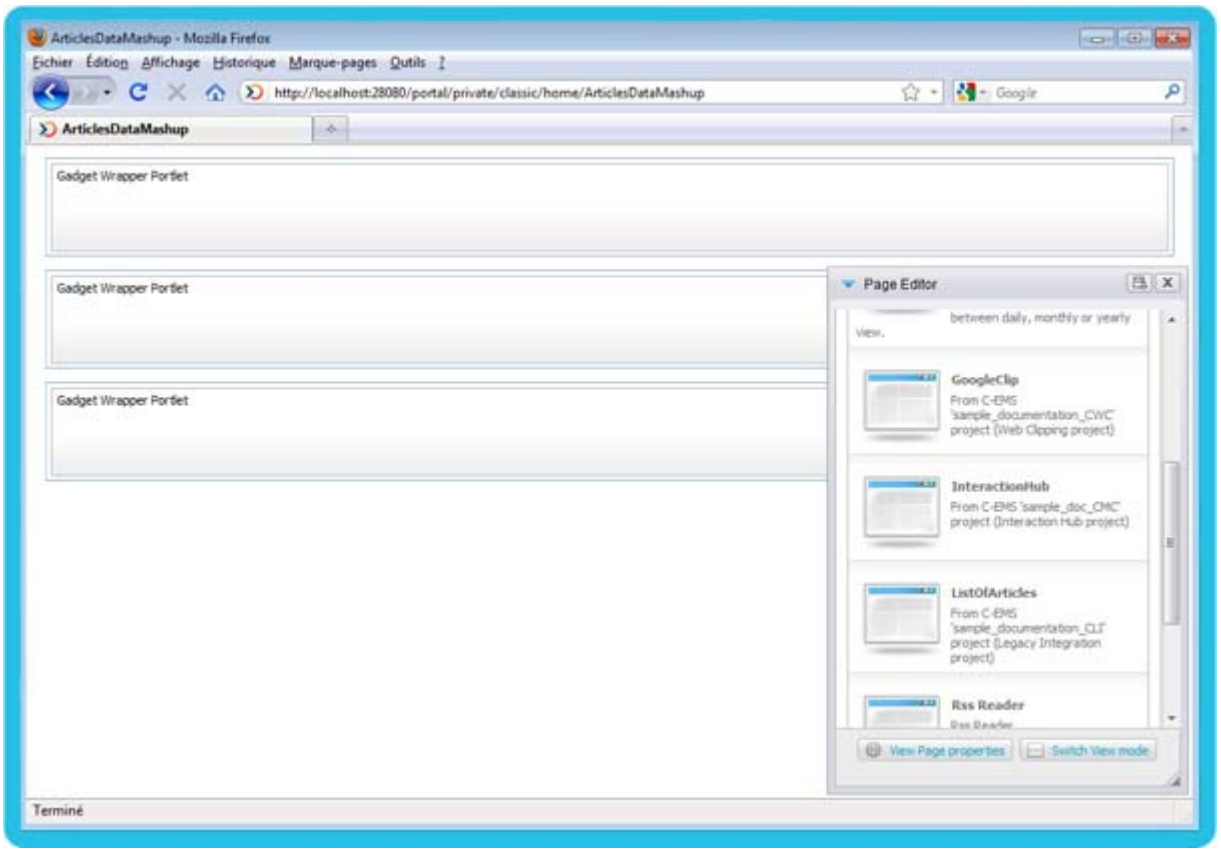

Figure 2 - 70: All widgets added to the page

The three widgets now appear on the page on top of each other. We would like to have the interaction hub widget on top of the page and the two other widgets displayed side by side, so we will now arrange their layout.

To arrange widget layout on the page

1 In the ***Page Editor*** toolbar, select **Containers** tab.

This tab offers several available containers to use to lay out the page, classified by categories (*Rows Layout*, *Columns Layout*, *Tabs Layout*, etc.):

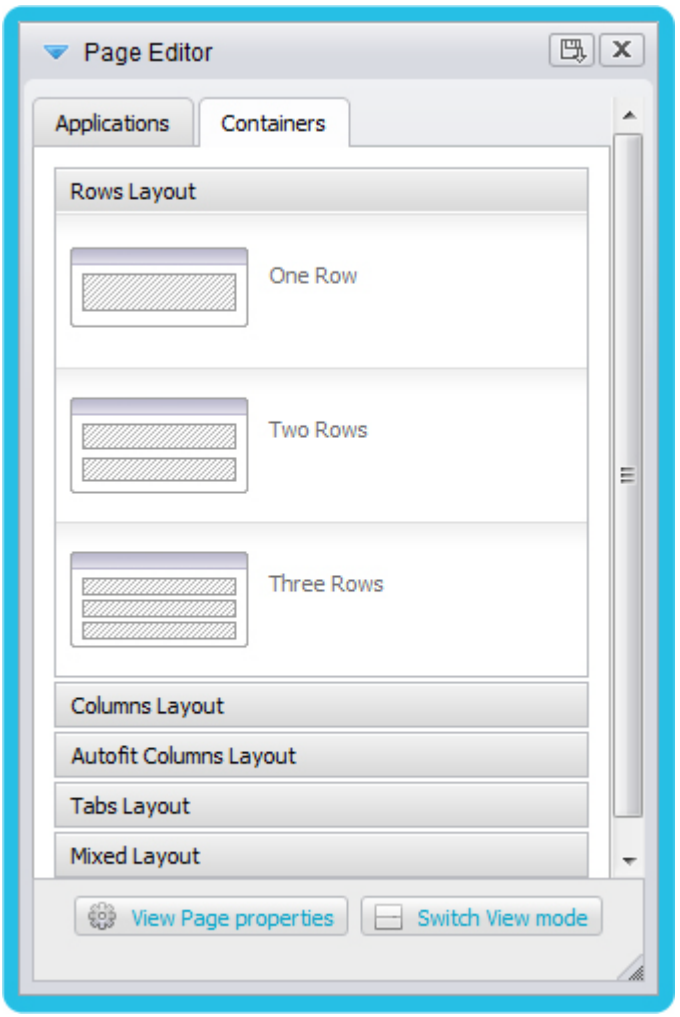

Figure 2 - 71: Selecting Containers tab in Page Editor toolbar

2 Expand **Mixed Layout** container category and find the appropriate layout for the page (as previously described, we would like to have the interaction hub widget on top of the page and the two other widgets displayed side by side):

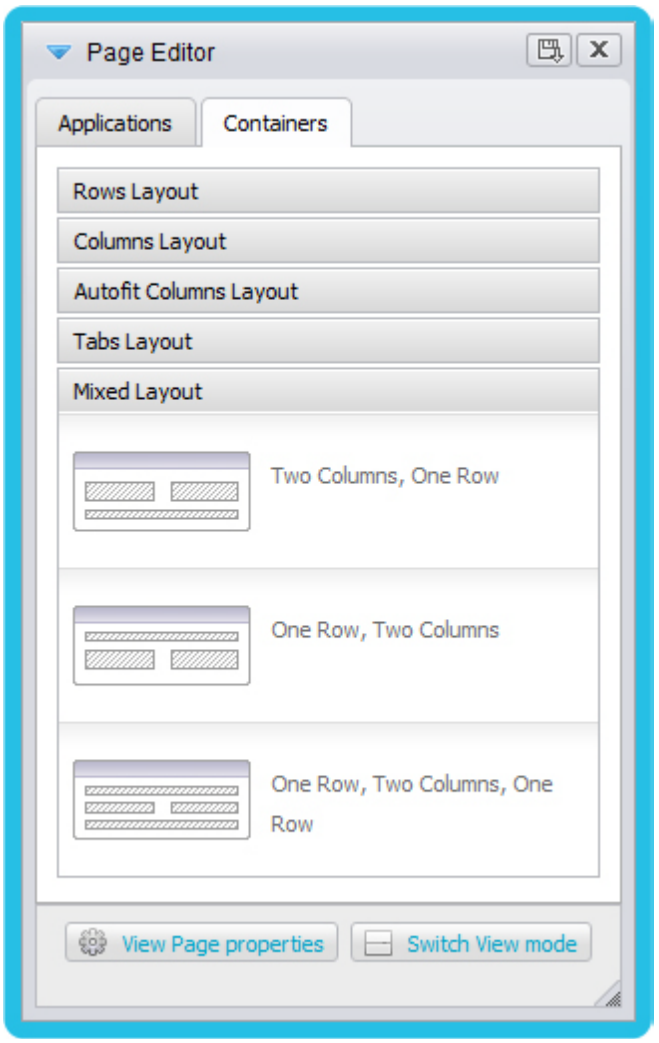

Figure 2 - 72: Available containers for Mixed Layout category

3 Select *One Row, Two Columns* container and drag-and-drop it to the page to add it:

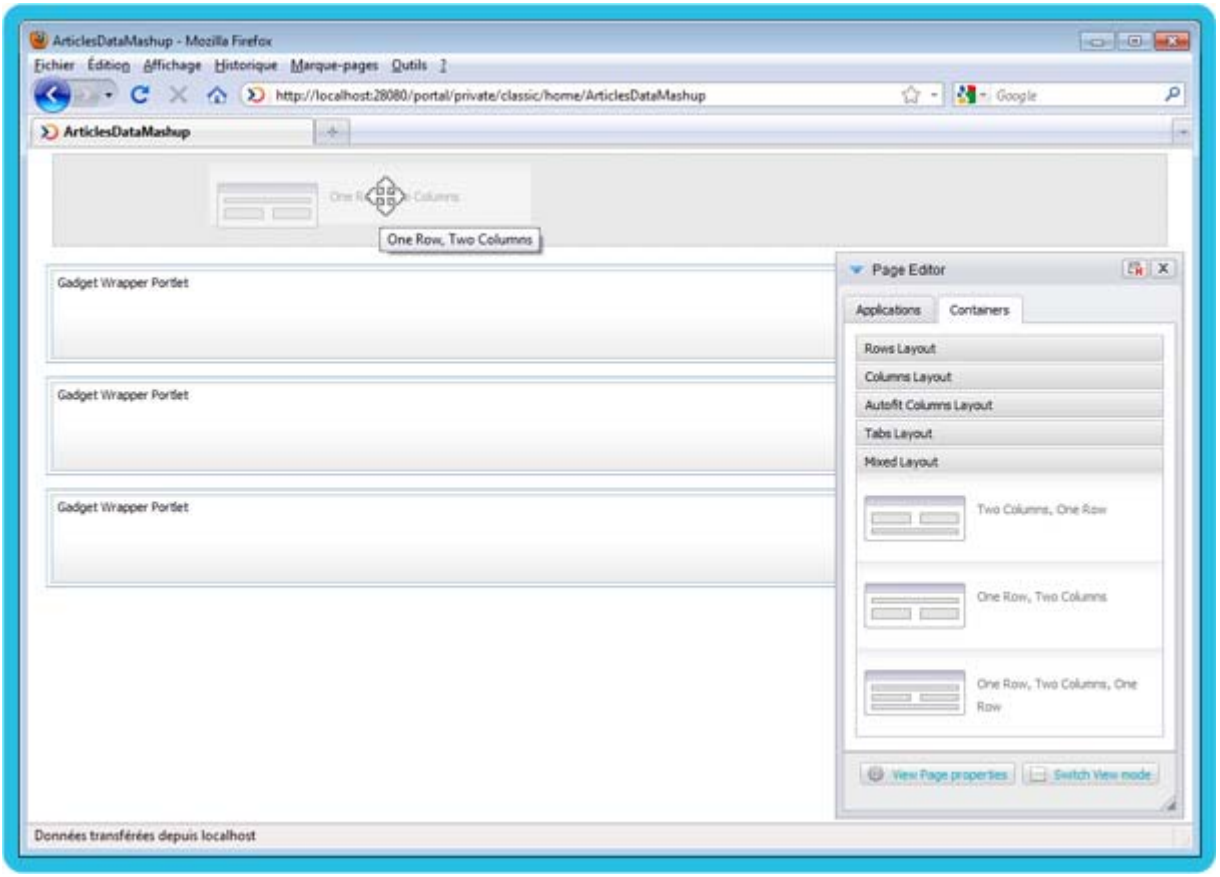

Figure 2 - 73: Adding One Row, Two Columns cfontainer to the page

The container is added in the page, next to the widgets.

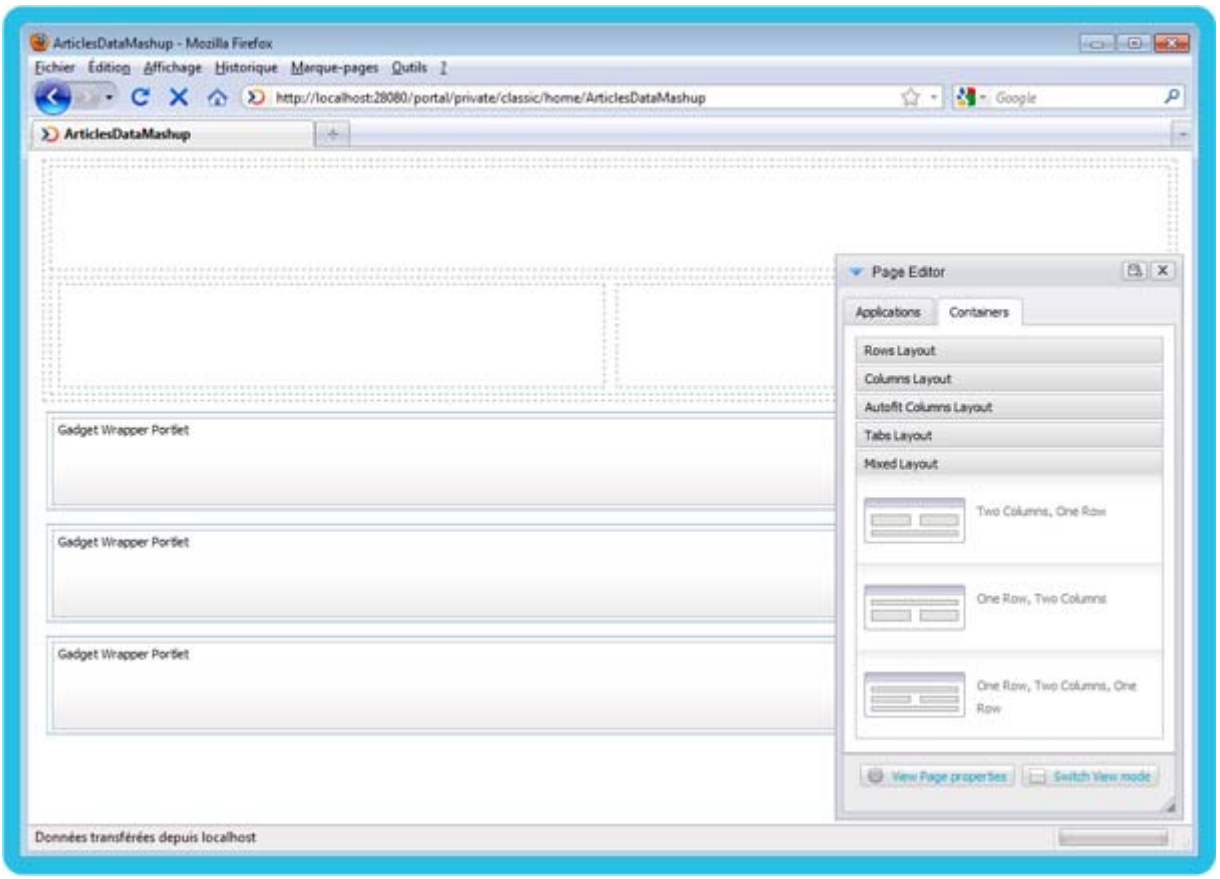

Figure 2 - 74: Container added in the page

The next step consists in moving the widgets to the container to finalize the page's layout. To be able to move the widgets, the Page Editor must be in ***Applications*** mode. In ***Containers*** mode, you can only move containers.

To know what kind of elements you can move and edit, move the mouse over elements on the page. You will see a small toolbar appearing above some elements: they are editable. The others are not and you should change Page Editor's mode to be able to edi t them.

4 In the ***Page Editor*** toolbar, select **Applications** tab.

5 Then, select the `InteractionHub` widget and, using its toolbar, drag-and-drop it to the container's first row:

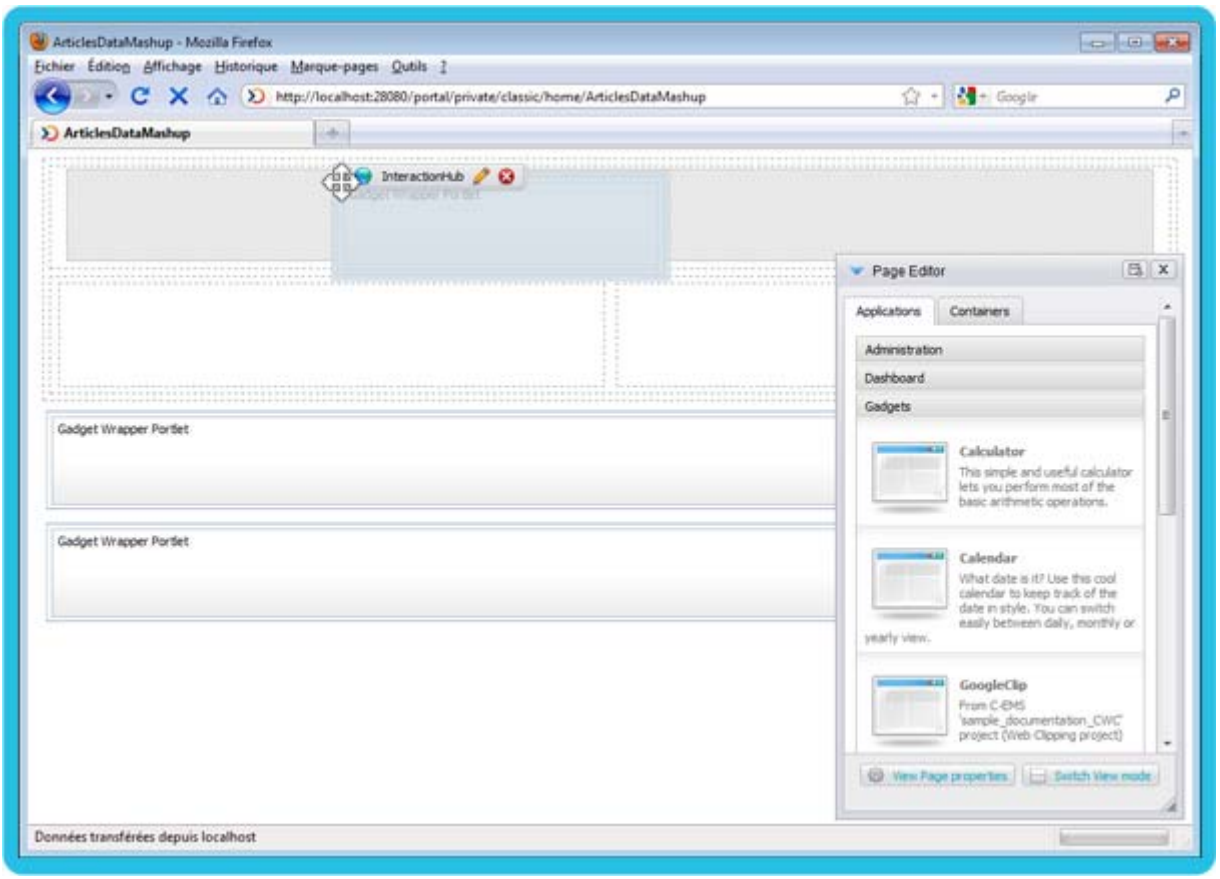

Figure 2 - 75: Moving InteractionHub widget in the layout

6 Repeat step 5 of this procedure for:

- the `ListOfArticles` widget, to be moved to the container's first column,
- the `GoogleClip` widget, to be moved to the container's second column.

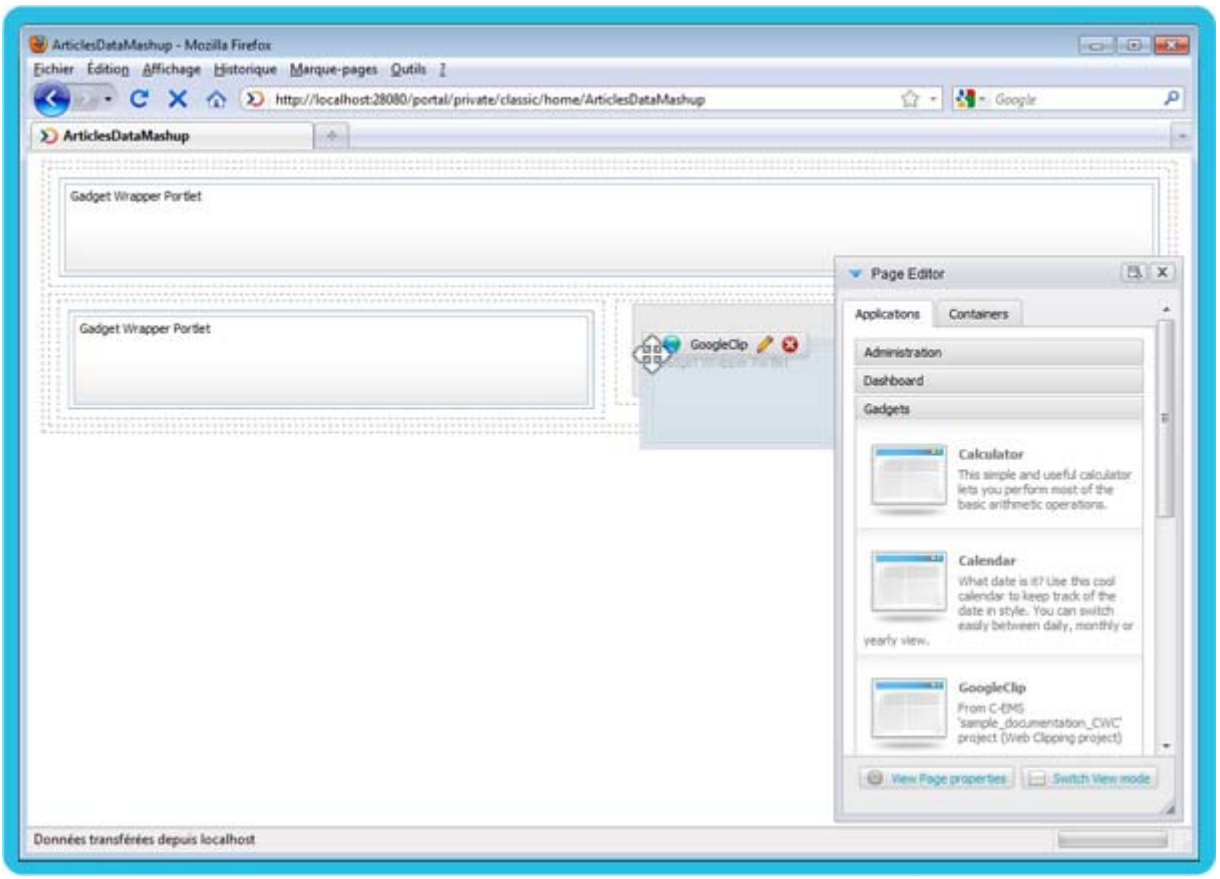

Figure 2 - 76: Moving GoogleClip widget in the layout

The three widgets are correctly laid out in the page.

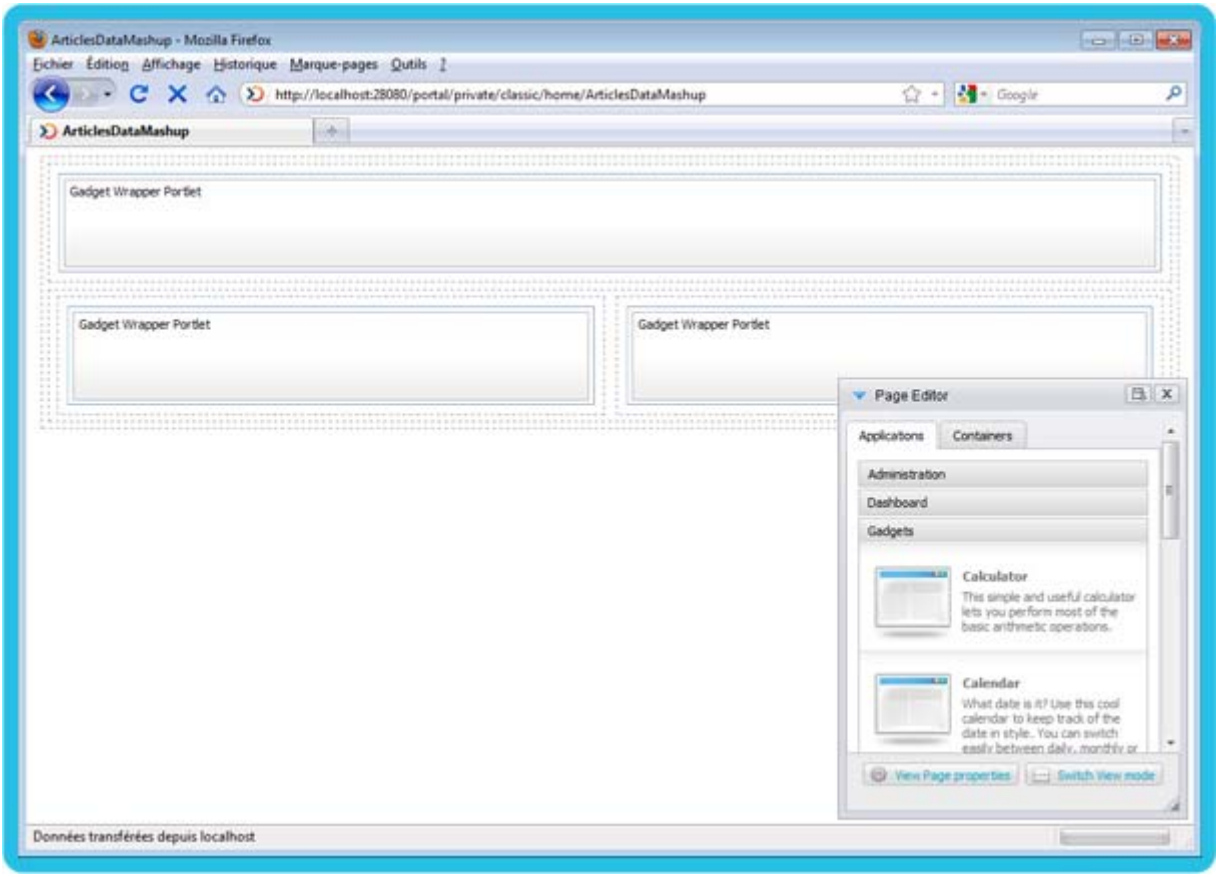

Figure 2 - 77: All widgets laid out in the page

7 Click on the **Finish** button of the ***Page Editor*** toolbar to save the edited page:

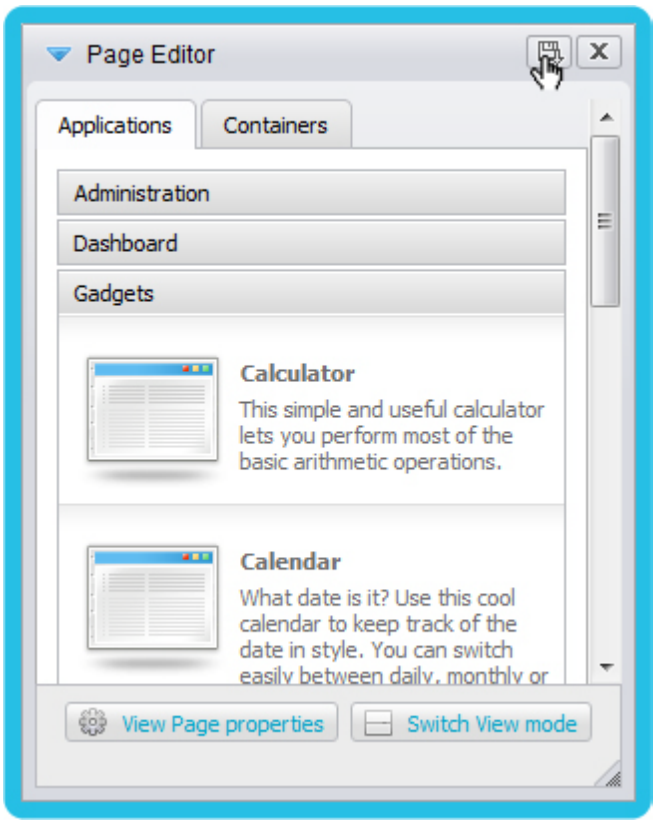

Figure 2 - 78: Clicking on finish button

The updated page is displayed, with the three widgets arranged in a user-friendly way.

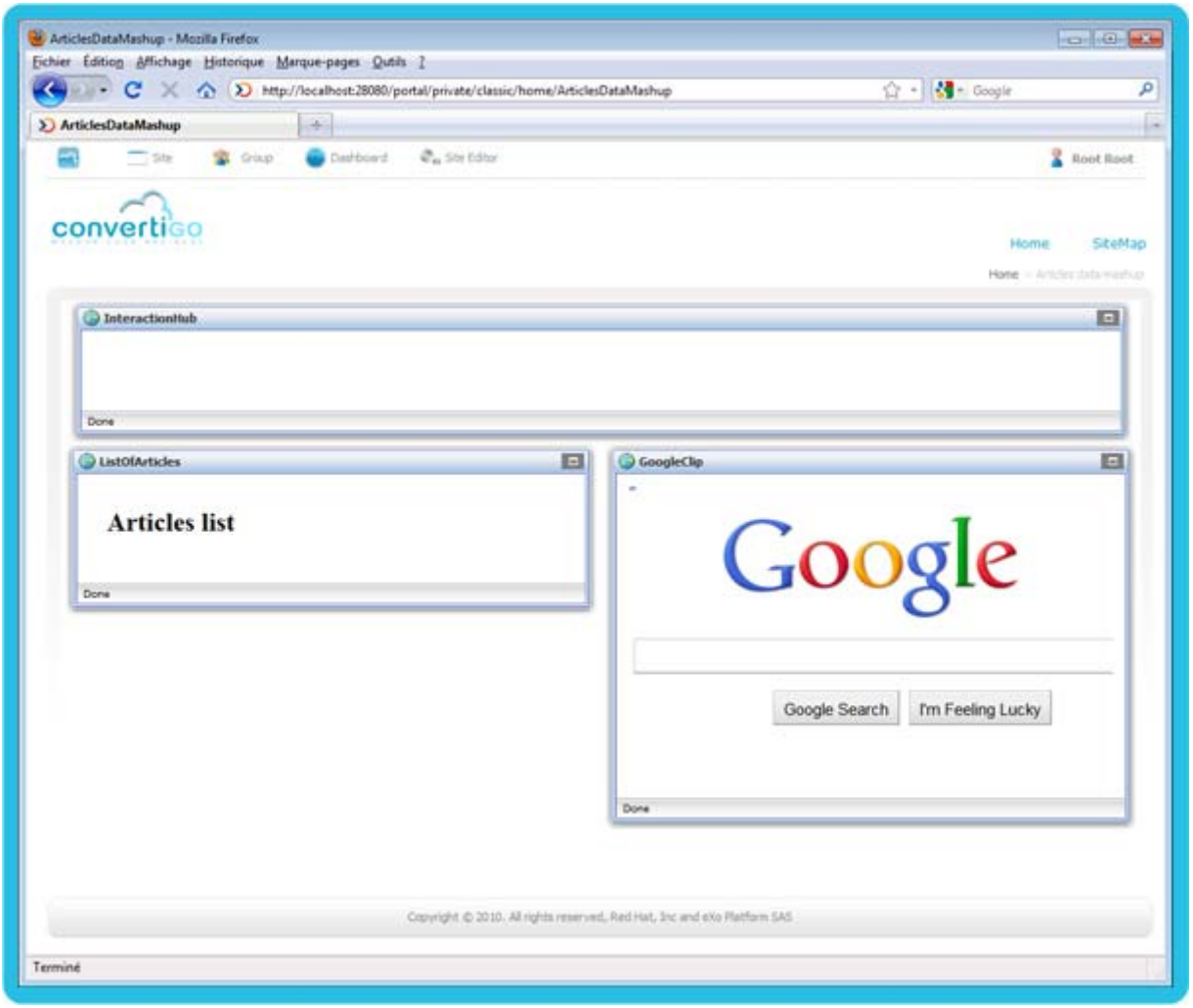

Figure 2 - 79: Article data mashup page

The three widgets are now arranged in a user-friendly way in the page. They are automatically starting when the page is displayed. We can notice that the `InteractionHub` widget is empty and the `ListOfArticles` widget displays a title but no content.

WHAT COMES NEXT?

The next section explains how to finalize the application so that the widgets display a correct content. After the composite application is finalized, we can test it by checking that the interaction is effective between widgets. This is also done in the next section.

2.5 Finalizing and Testing the Composite Application

The following sections explain step by step how to finalize and test your first Composite Application in Convertigo Mashup Composer:

- Finalizing widgets
- Testing the Composite Application

2.5.1 Finalizing widgets

Widgets automatically start as soon as the page including them is opened. They should display:

- what results from executing the transactions upon which they are based:
  - The `ListOfArticles` widget displays a list of articles together with Google search icons next to each article. This is the XML output resulting from the execution of the CLI `GetArticleData` transaction, transformed using the edited `articlesList.xsl` file.

For results to appear in the `ListOfArticles` widget, the CLI emulator must connect to a legacy server or a running trace, waiting on the login screen, so that the CLI transaction can start normally. For more information, refer to the "Starting with Convertigo Legacy Integrator" Quick Guide.

  - The `GoogleClip` widget displays the Google search engine with an empty field. This is the cliplet resulting from the execution of the CWC `GoogleClip` transaction.
- what is parametered in an HTML template file:
  - The `InteractionHub` widget is performing interactions between widgets, so it does'nt need to be effectively displayed. The widget can be hidden or an HTML template file can be parametered to display something.

The following procedures show how to finalize the above mentioned widgets.

Finalize ListOfArticles widget

Finalize InteractionHub widget

FINALIZE LISTOFARTICLES WIDGET

As previously said, at first, the `ListOfArticles` widget displays nothing but its title. This widget's malfunction is due to the legacy connection that is not established.

In the Convertigo Studio, this project runs a trace on which the connector can connect. Since the project is deployed on the Convertigo Server, the project can't connect as no trace is running.

The following procedures explain how to:

- access the Convertigo Server *Administration Console*,
- run a trace on the Convertigo Server,
- update legacy integrator project parameters to connect to the trace.

To acess the Convertigo Server Administration Console

1. Launch a standard Web browser.
2. In the URL address field, type in the Convertigo Server ***Administration Console*** URL in the following format:

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/admin
```

For example, in our project:

- *HostName* = `localhost` (default installed server);
- *ConvertigoPortNumber* = `28080` (default port for server).

For example, in a browser URL address bar:

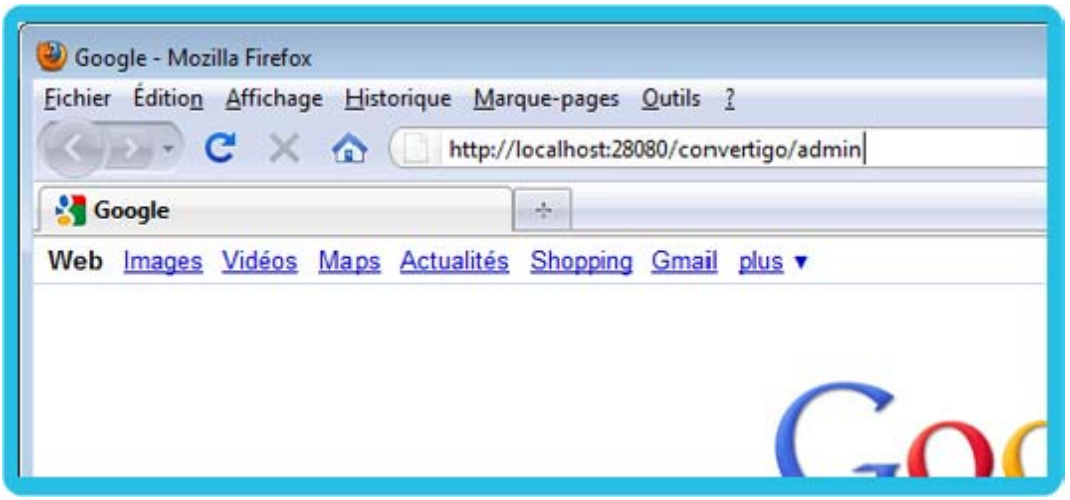

Figure 2 - 80: Convertigo Server Administration Console URL

The login page of the Server ***Administration Console*** opens:

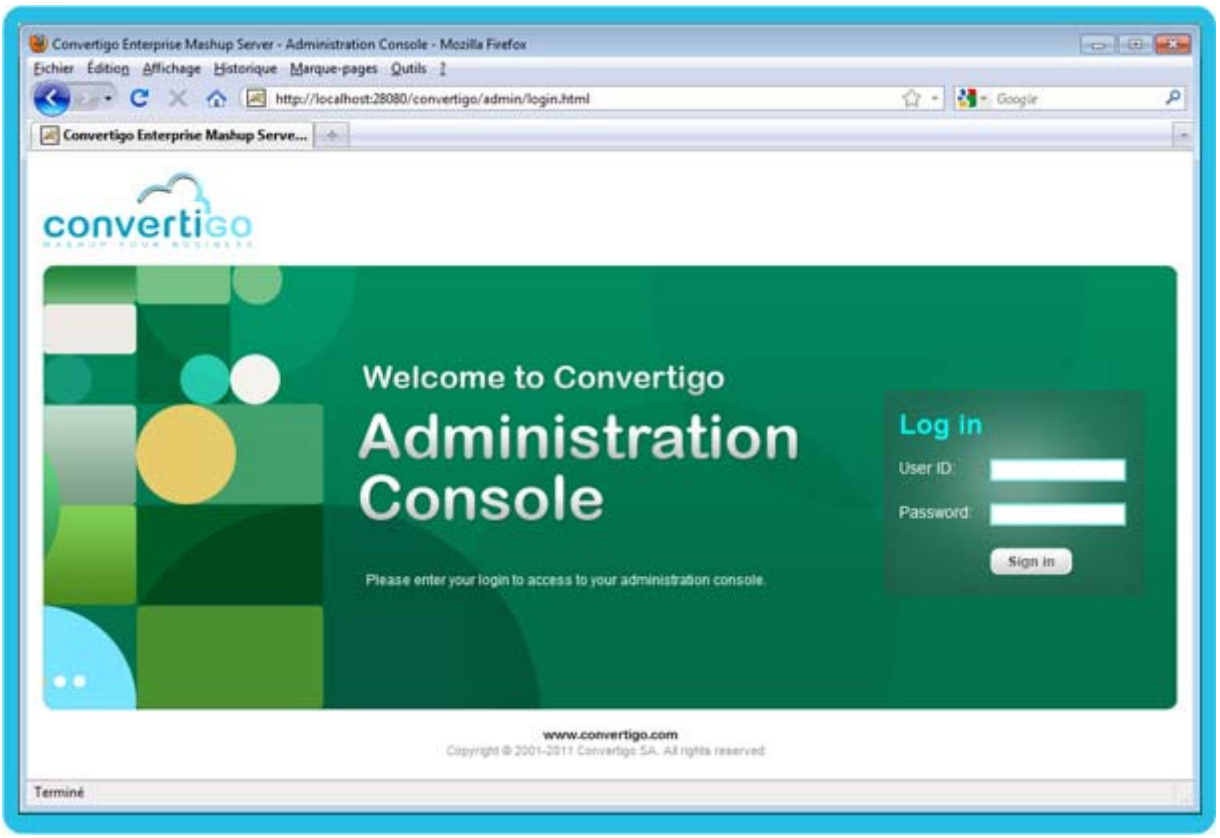

Figure 2 - 81: Login page of Server Administration Console

3 In the **User ID** field, type in the Server Administration user ID (for example `admin`).

4 In the **Password** field, type in the user's password (for example `admin`).

Figure 2 - 82: Typing in user ID and password

5 Click on the **Sign in** button to validate and authentify to the Server ***Administration Console***.

6 The Server Administration Console *Home* page opens.

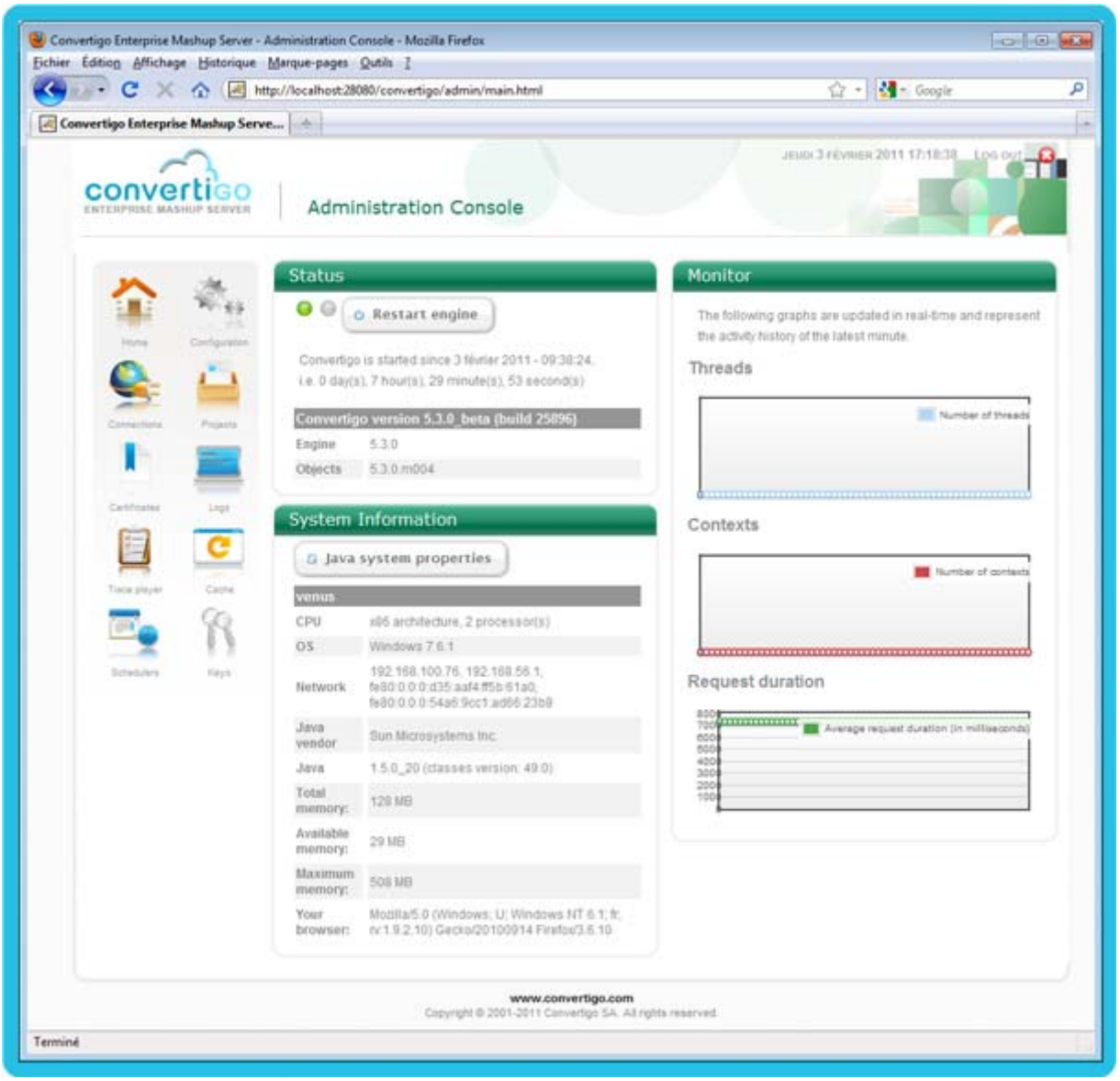

Figure 2 - 83: Administration Console Home page

It displays:

- a left menu leading to Convertigo Server Administration pages:

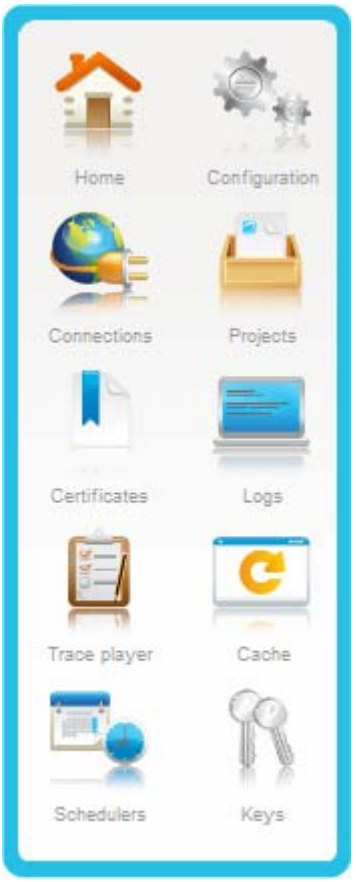

Figure 2 - 84: Administration Console left menu

- a right part containing three widgets:
  - ***Status*** widget;

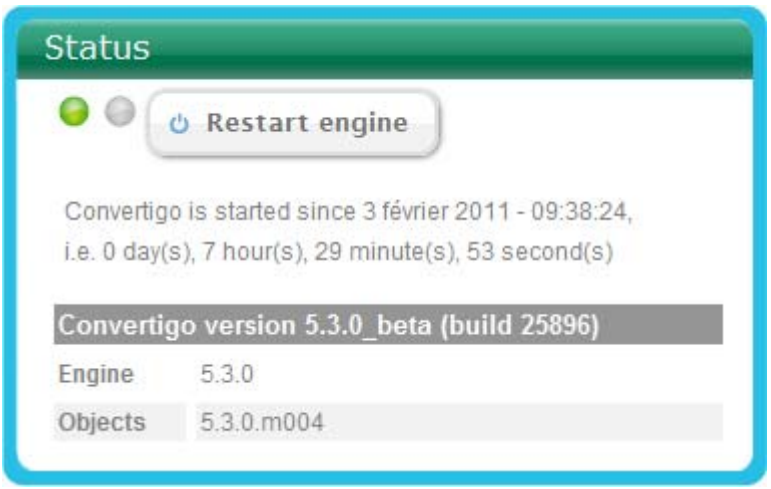

Figure 2 - 85: Status widget in Administration Console

  - ***System Information*** widget;

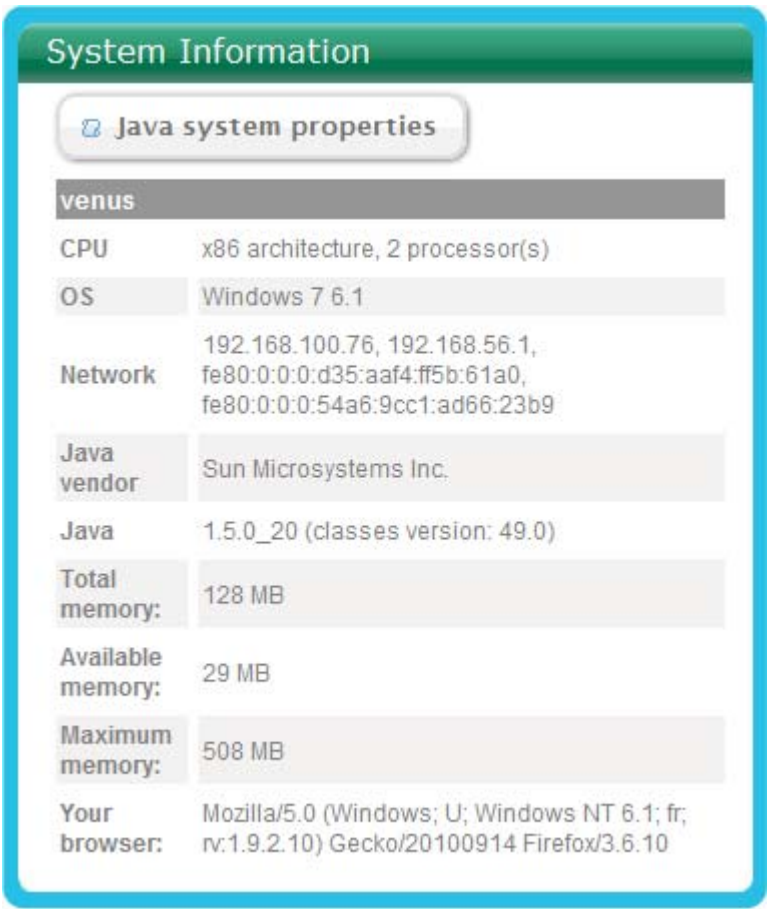

Figure 2 - 86: System Information widget in Administration Console

- ***Monitor*** widget.

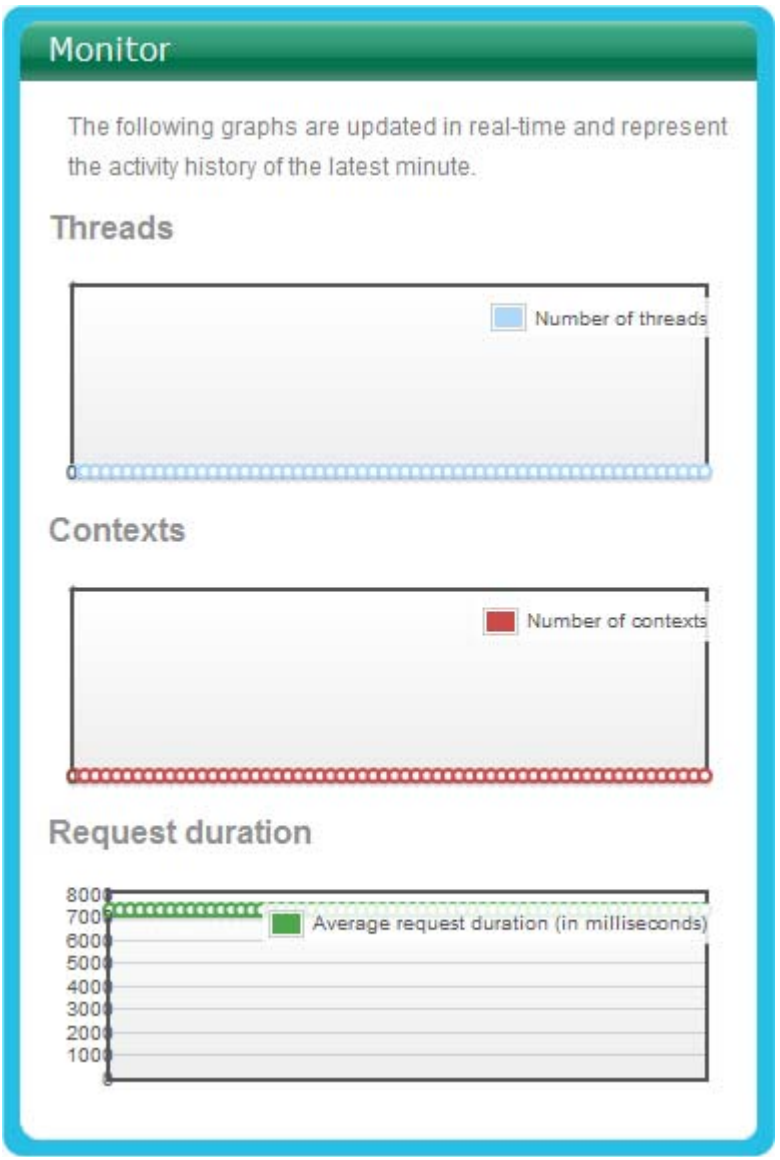

Figure 2 - 87: Monitor widget in Administration Console

The next step is to set up a trace running on the Convertigo Server thanks to the Administration Console.

To run a trace on the Convertigo Server

1 In the Server Administration Console menu, click on the **Trace player** button.

Figure 2 - 88: Clicking on Trace player button

The ***Trace player*** widget is displayed on the right part of the page:

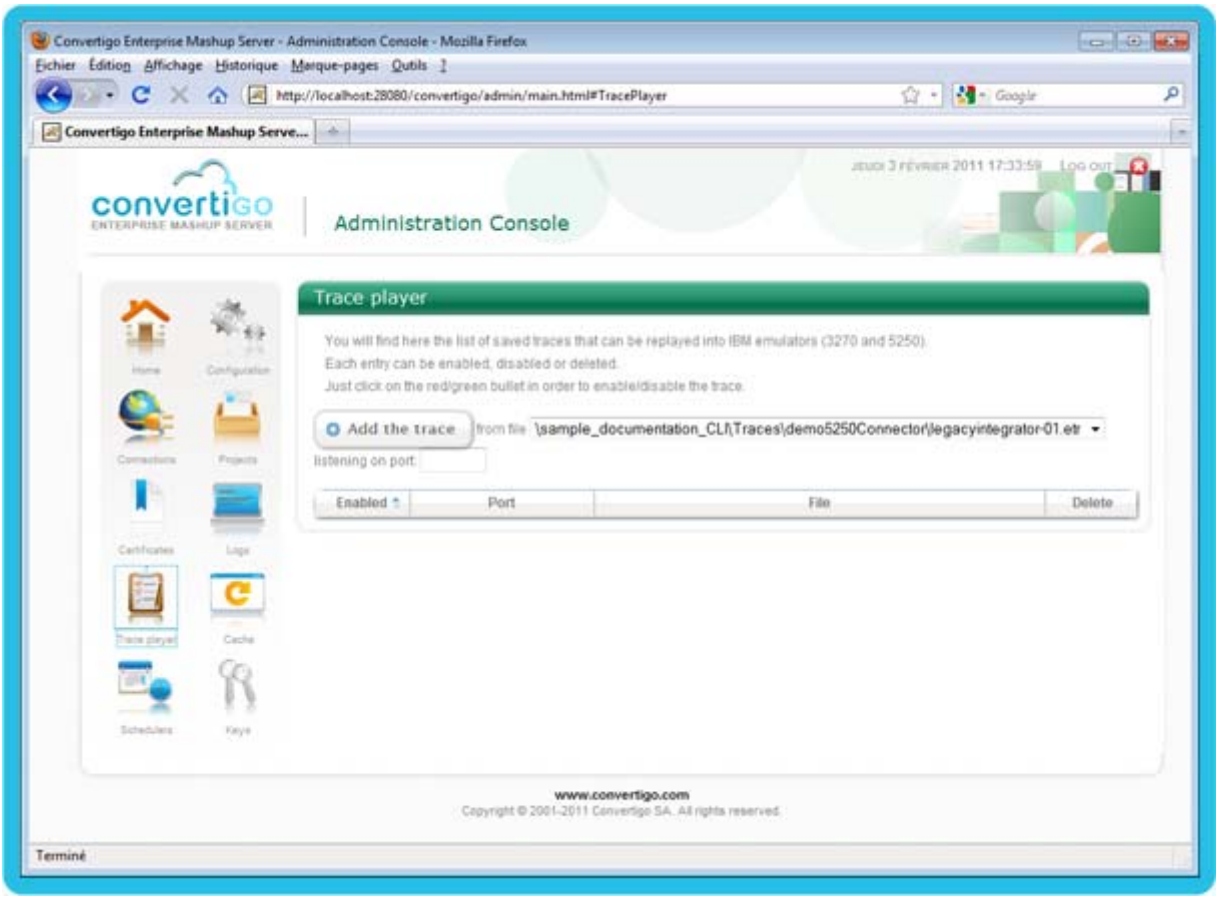

Figure 2 - 89: Trace player widget in Administration Console

2 A **from file** combo box lists all available trace files from projects deployed on the

server:

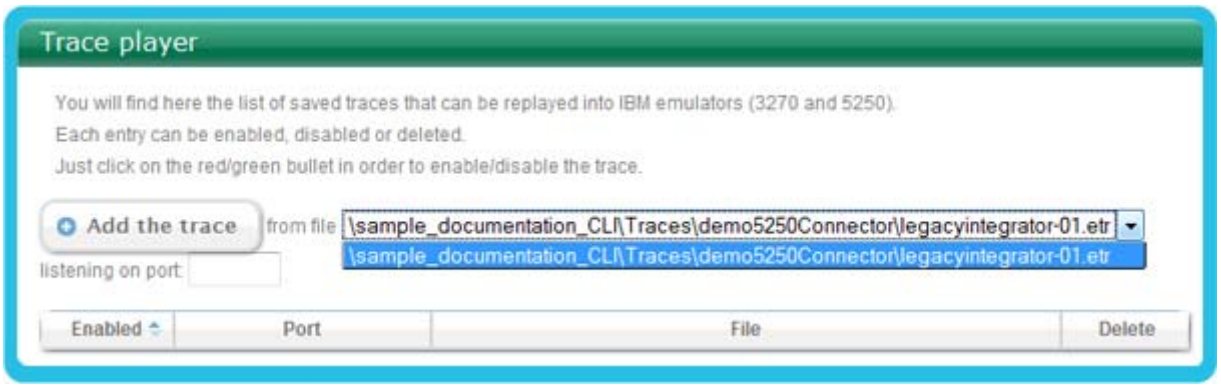

Figure 2 - 90: Available traces in Trace player widget

In the case of our sample project, one trace file is available: `legacyintegrator-01.etr` from `sample_documentation_CLI` project.

Select this trace in the combo box.

3 Then, input a port number value in the **listening on port** field (for example input `23000` port number).

The chosen port number must be between 1025 and 64999.

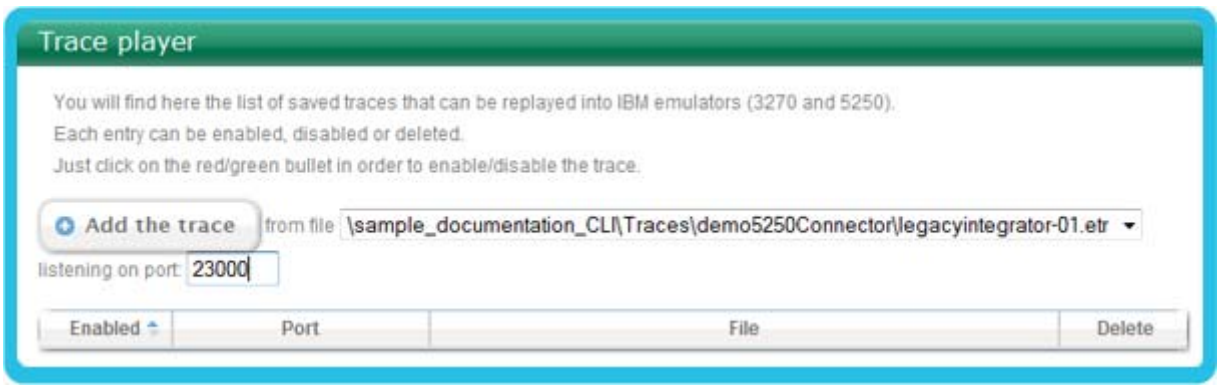

Figure 2 - 91: Setting trace running in Trace player widget

4 Click on the **Add the trace** button to add the trace to running traces.

5 A pop-up is displayed indicating that the trace has been correctly added on the chosen port.

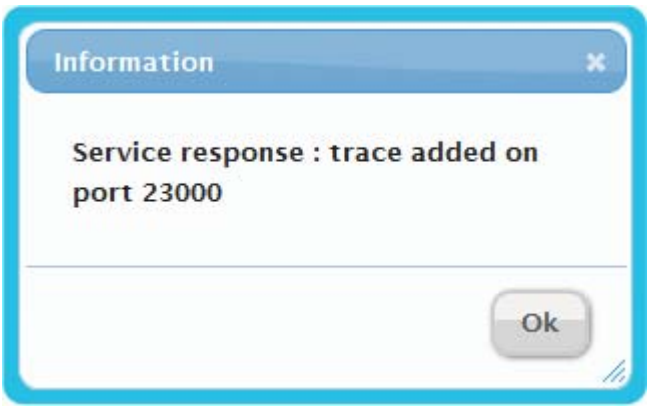

Figure 2 - 92: Trace added confirmation message

6 Click on the **Ok** button to close the message.

The trace is appearing in the table of the ***Trace player*** widget.

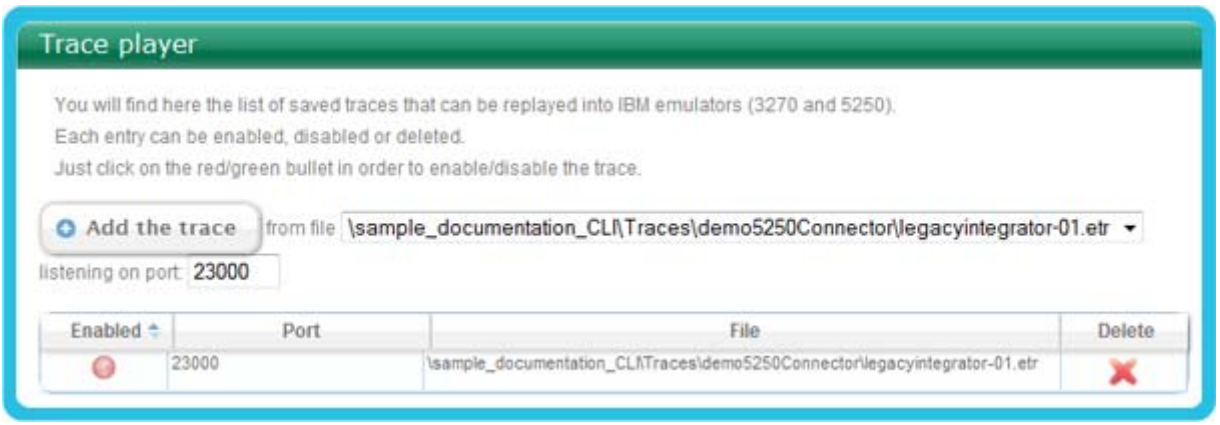

Figure 2 - 93: Trace appearing in Trace player widget

7 The **Enabled** columns icon is red, meaning that the trae is not running. To start running the trace, click on the red **Enabled** column icon.

The **Enabled** column icon turns to green, the trace is running.

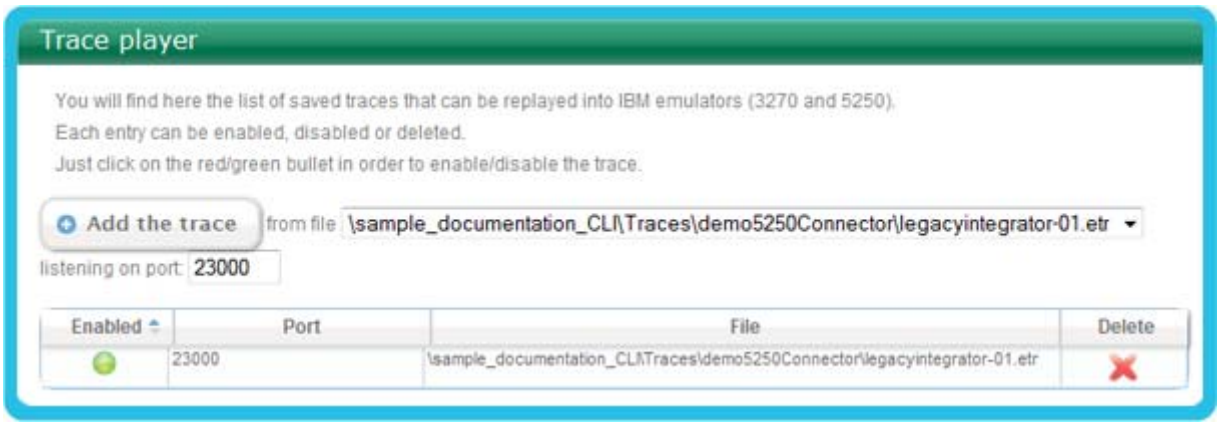

Figure 2 - 94: Trace running in Trace player widget

The next step is to update the sample_documentation_CLI project for its connector to conntect to the running trace on the Convertigo Server.

To update a project's properties on the Convertigo Server

1 In the Server Administration Console menu, click on the **Projects** button.

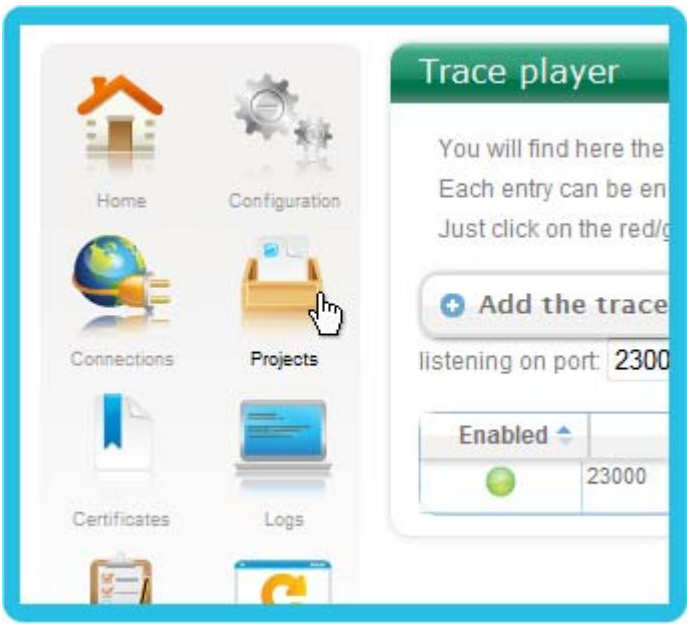

Figure 2 - 95: Clicking on Projects button

The ***Project List*** widget, listing the three projects deployed on the Server, is displayed on the right part of the page:

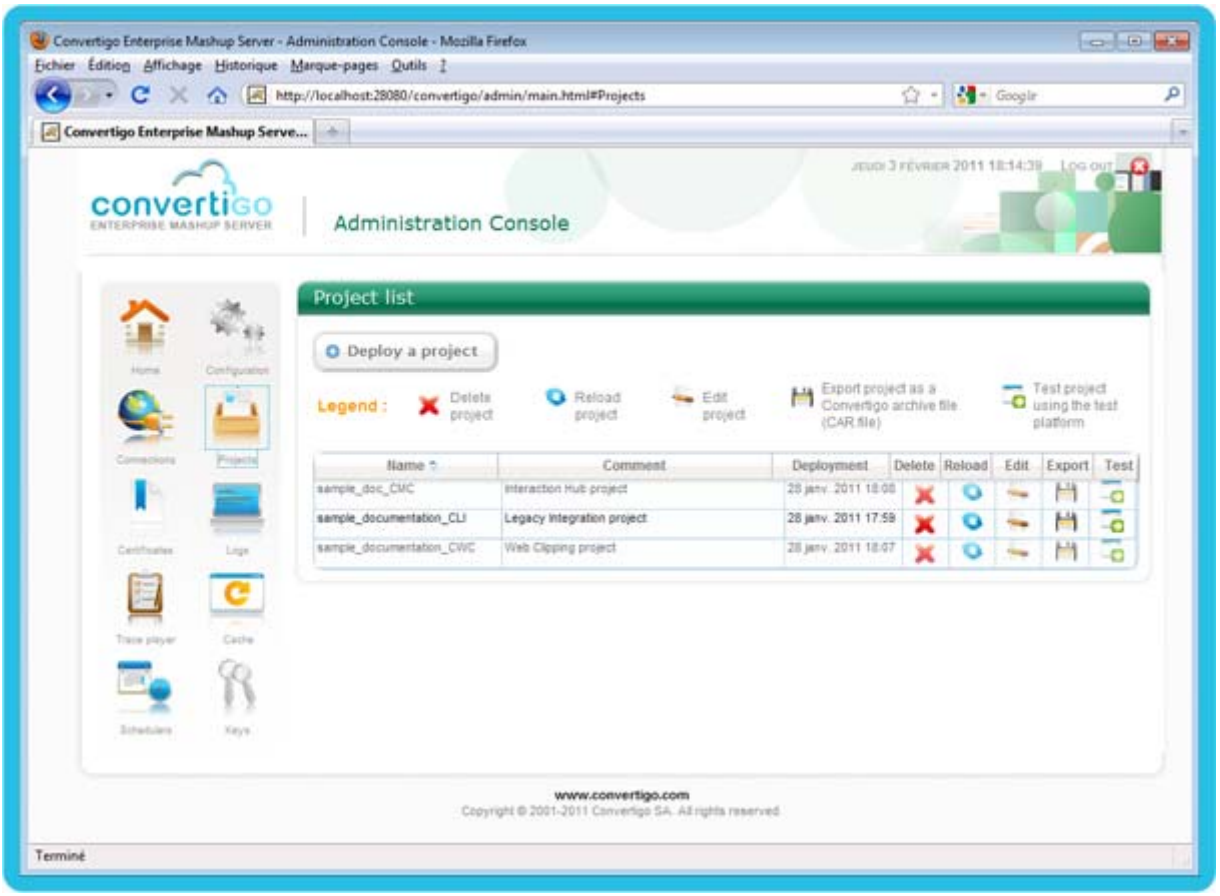

Figure 2 - 96: Project list widget in Administration Console

2 Click on the **Edit** button of the `sample_documentation_CLI` project's row.

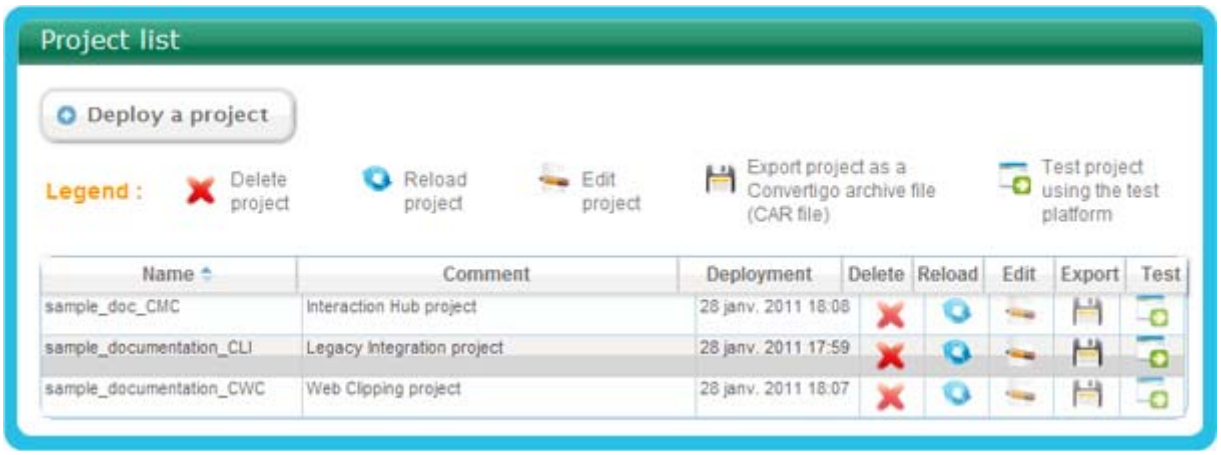

Figure 2 - 97: Accessing sample_documentation_CLI project's edition

The **Project sample_documentation_CLI** edition widget appears below the **Projects list** widget.

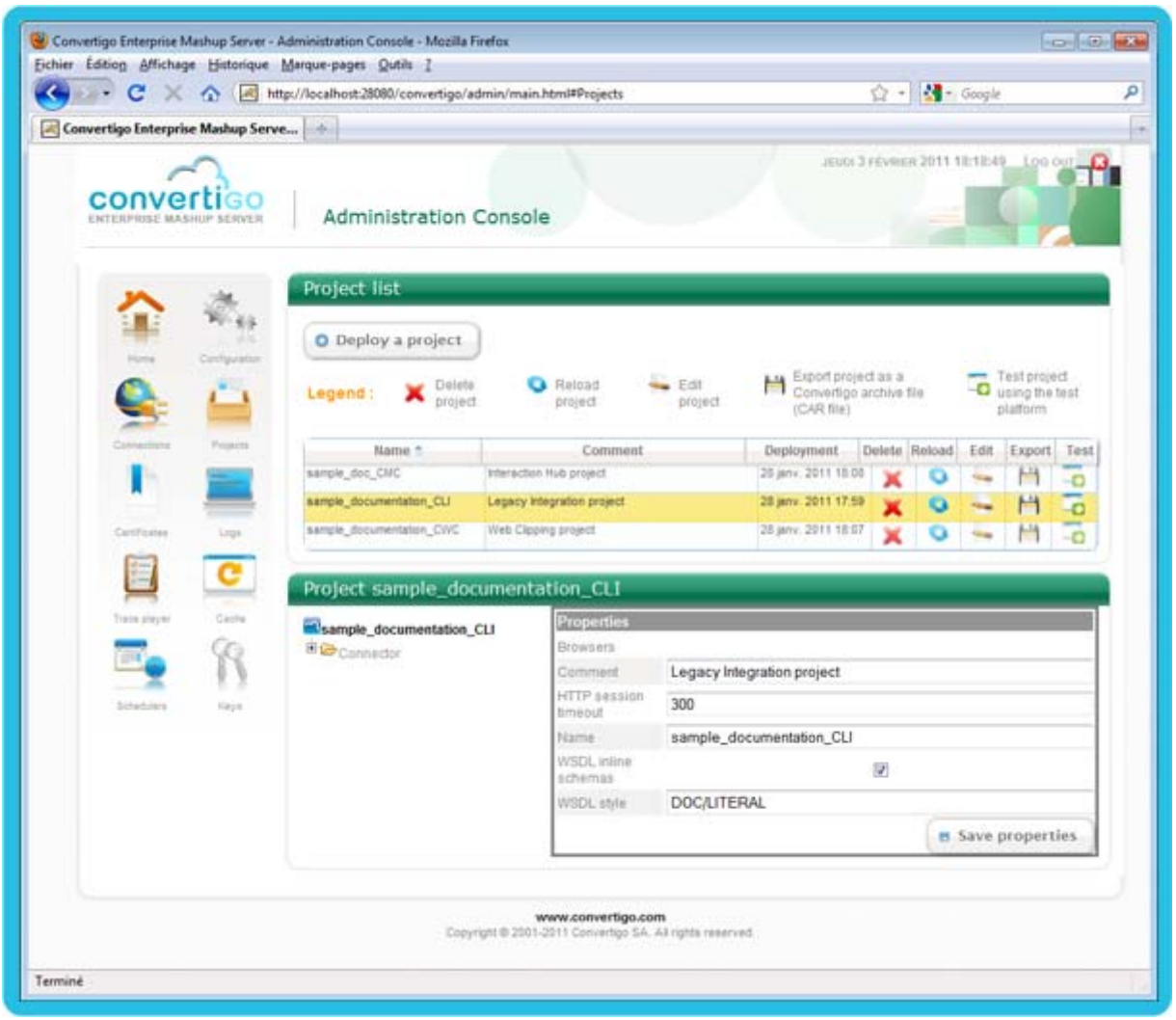

Figure 2 - 98: Project sample_documentation_CLI edition widget

It displays:

- on the left side: a *tree view* with the project's objects,

- on the right side: a *Properties* view, showing properties of the selected object in the tree view.

3 Expand the tree view until you can see the connector object and select the connector (for example `demo5250connector`) to display its properties.

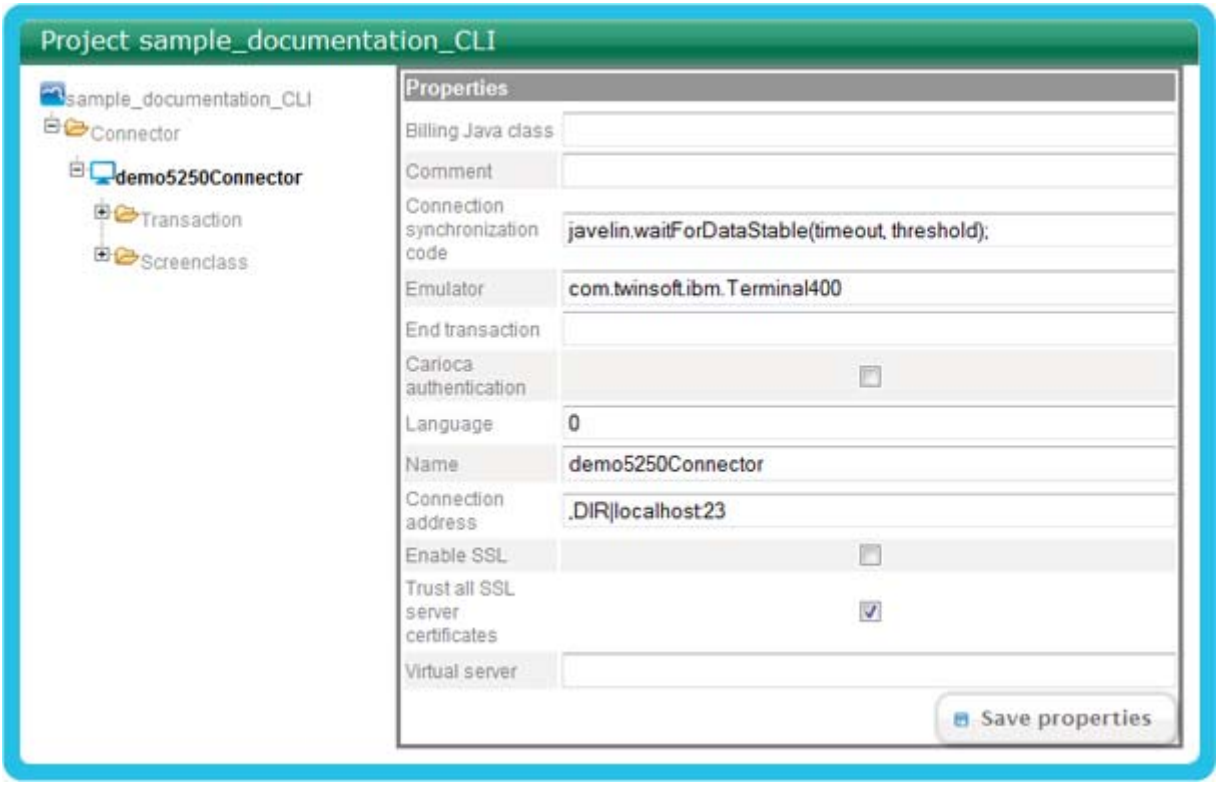

Figure 2 - 99: Displaying sample_documentation_CLI project's connector properties

The connector properties are displayed and editable in the *Properties* view.

4 Update the connector's **Connection address** property: replace the 23 port number by the port number parametered in the Trace player (for example set port number `23000`).

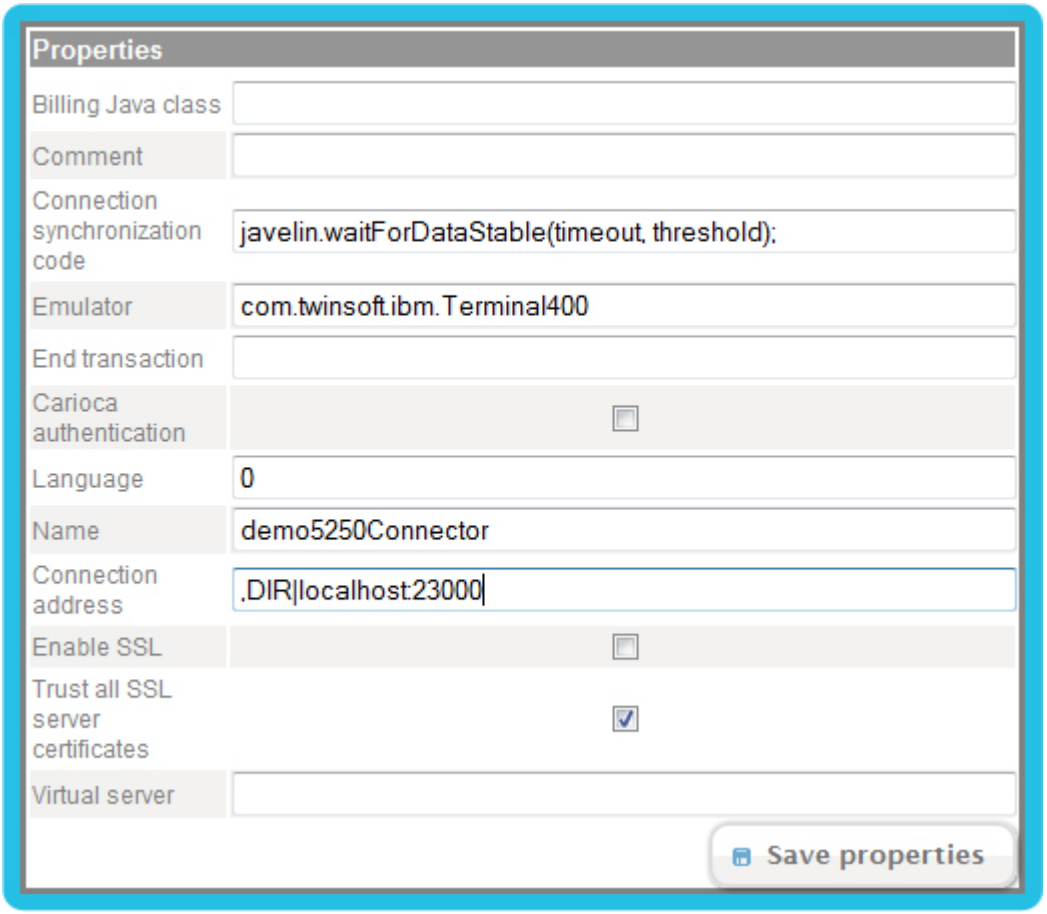

Figure 2 - 100: Updating connector's Connection address property

5 Click on the **Save properties** button to update the connector.

6 A pop-up is displayed indicating that the project has been updated.

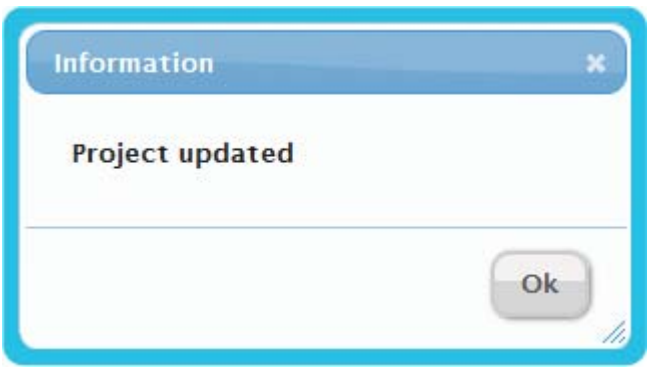

Figure 2 - 101: Project updated confirmation message

7 Click on the **Ok** button to close the message.

The project now correclty connects to the trace and the widget will display the list of articles.

8 Switch to the browser where is opened the Mashup Composer and refresh the *Articles data madshup* page. The `ListOfArticles` widget displays the expected list of articles:

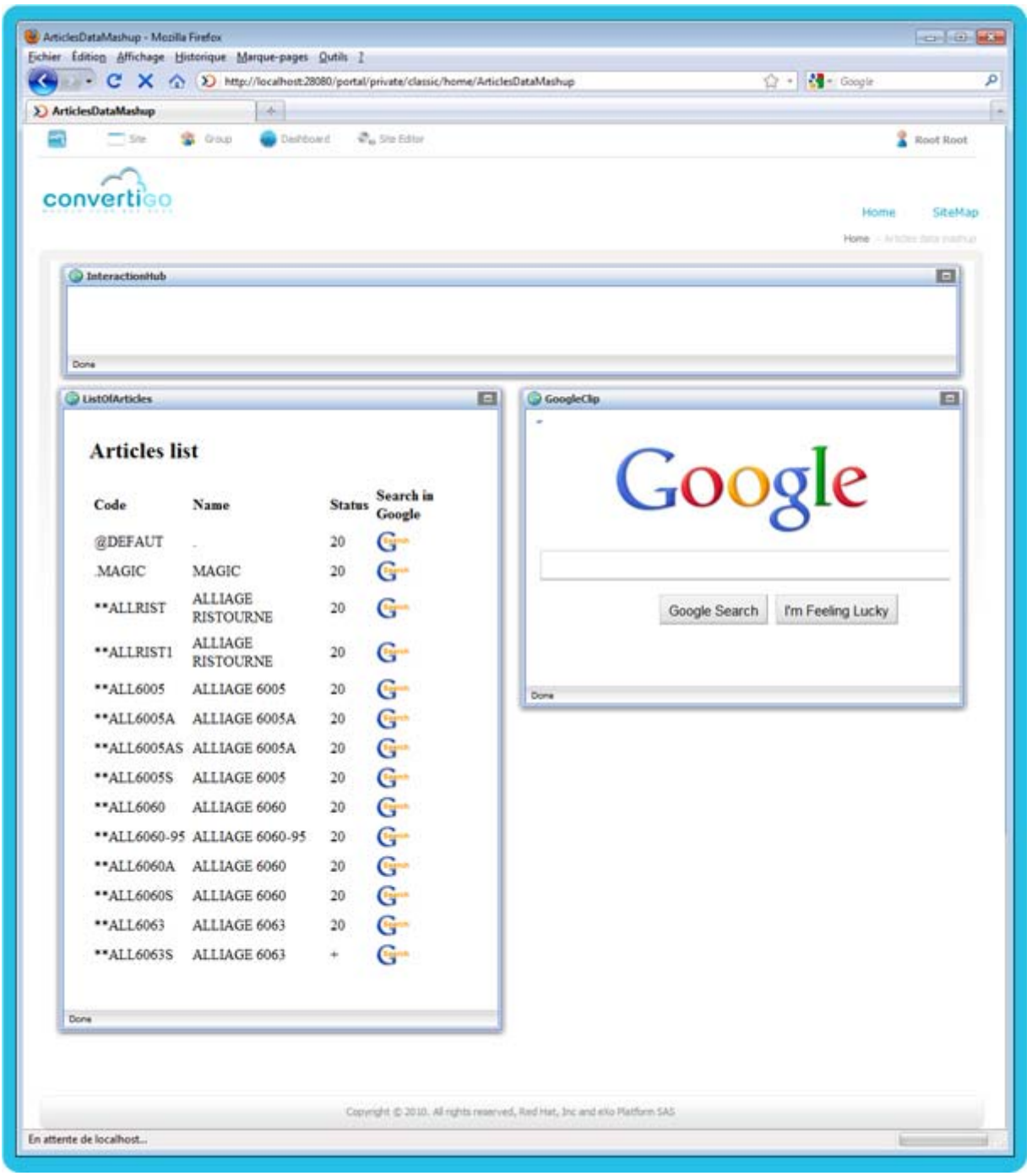

Figure 2 - 102: ListOfArticles widget displaying the list of articles from the legacy application

FINALIZE INTERACTIONHUB WIDGET

The `InteractionHub` widget's display is empty. We have to choose between two possible improvements:

- Hide the `InteractionHub` widget. In Mashup Composer, the widget's properties can be edtied for the widget not to be visible.
- Add a logo or a banner into this widget to improve the page render. In interaction hub projects, an HTML template file can be parametered to display something in the widget.

The following procedures (choose between both) explain how to:

- update widget's properties in Mashup Composer to hide the widget,
- edit the HTML template file of interaction hub project in the Studio, and deploy again the project to the Server.

To update widget's properties in Mashup Composer to hide the widget

1 Switch to the Mashup Composer window (already opened).

To open the Mashup Composer (if not already opened), see "Switching to the Mashup Composer" on page 2-27.

2 Open the ***Page Editor*** on the *Articles data mashup* page by following the procedure *"To open the Page Editor to edit a page"* on page 2 - 48.

The ***Page Editor*** opens with the following layout:

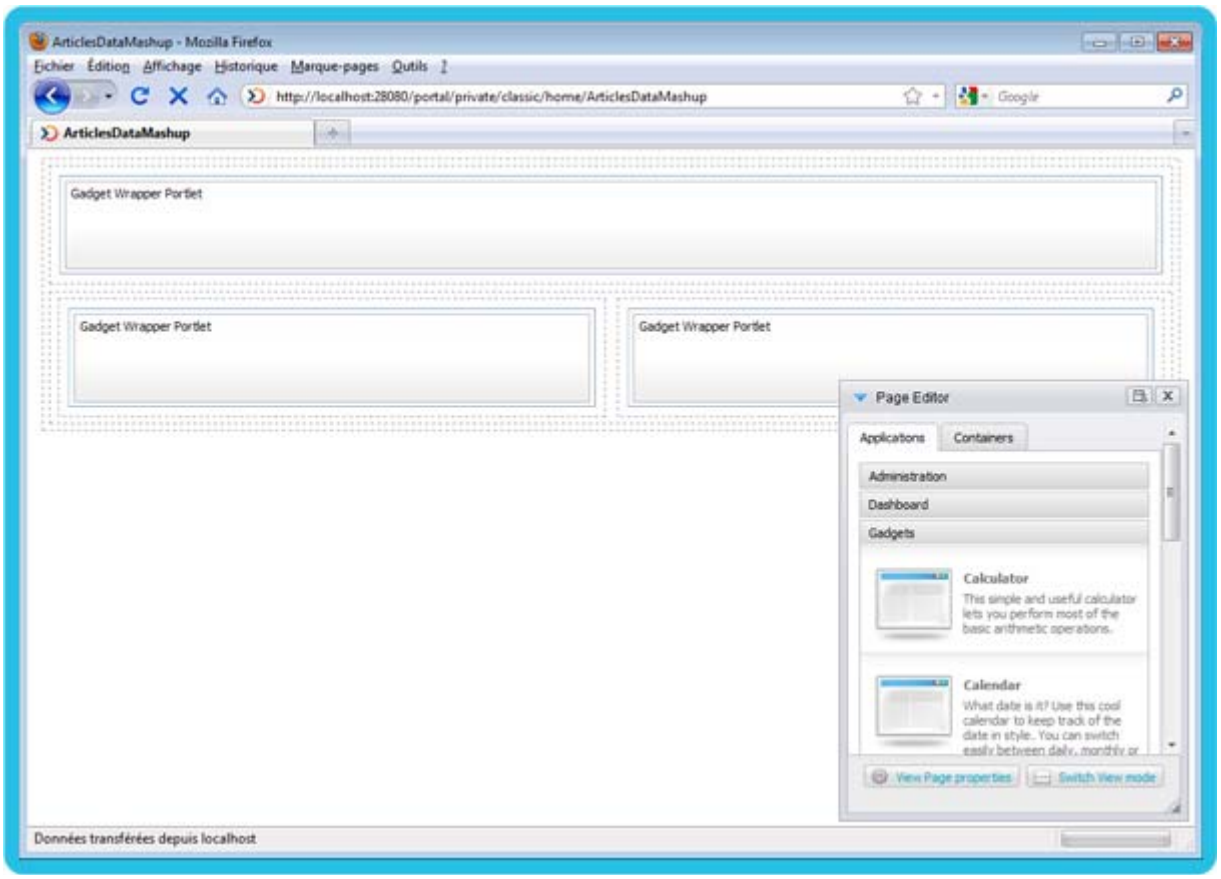

Figure 2 - 103: Page Editor opened on Article data mashup page

3 Move the mouse over the widgets (displayed as *Gadget Wrapper Porlet* in the ***Page Editor***) to display their toolbar and find the `InteractionHub` widget.

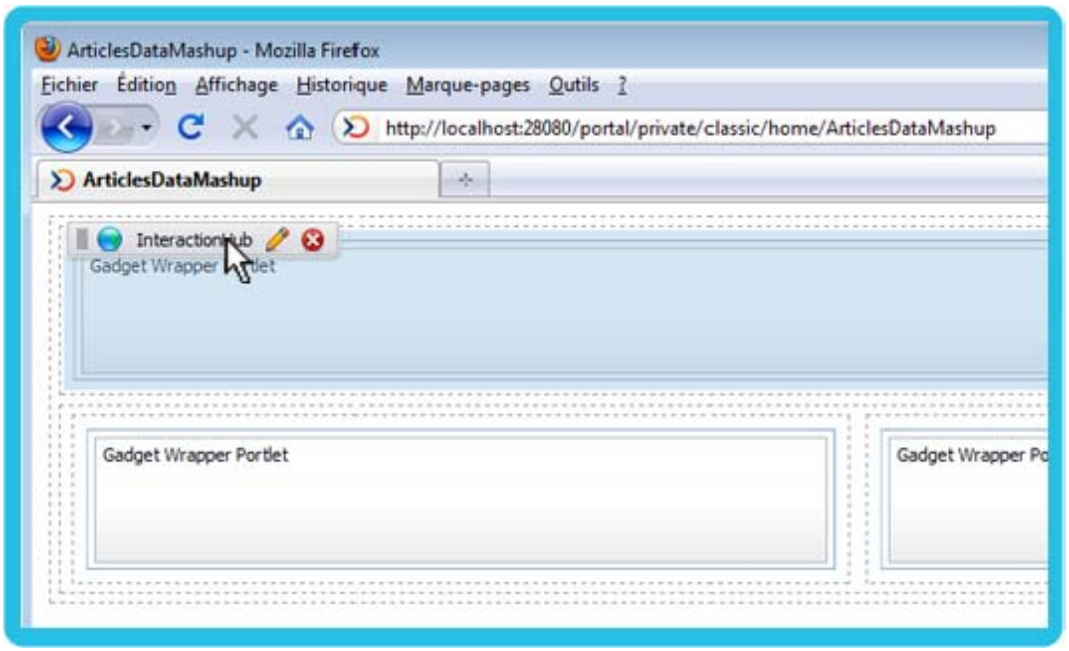

Figure 2 - 104: InteractionHub widget's toolbar

4 Click on the **Edit Porlet** button in the widget toolbar.

A form pops up to edit the widget's settings:

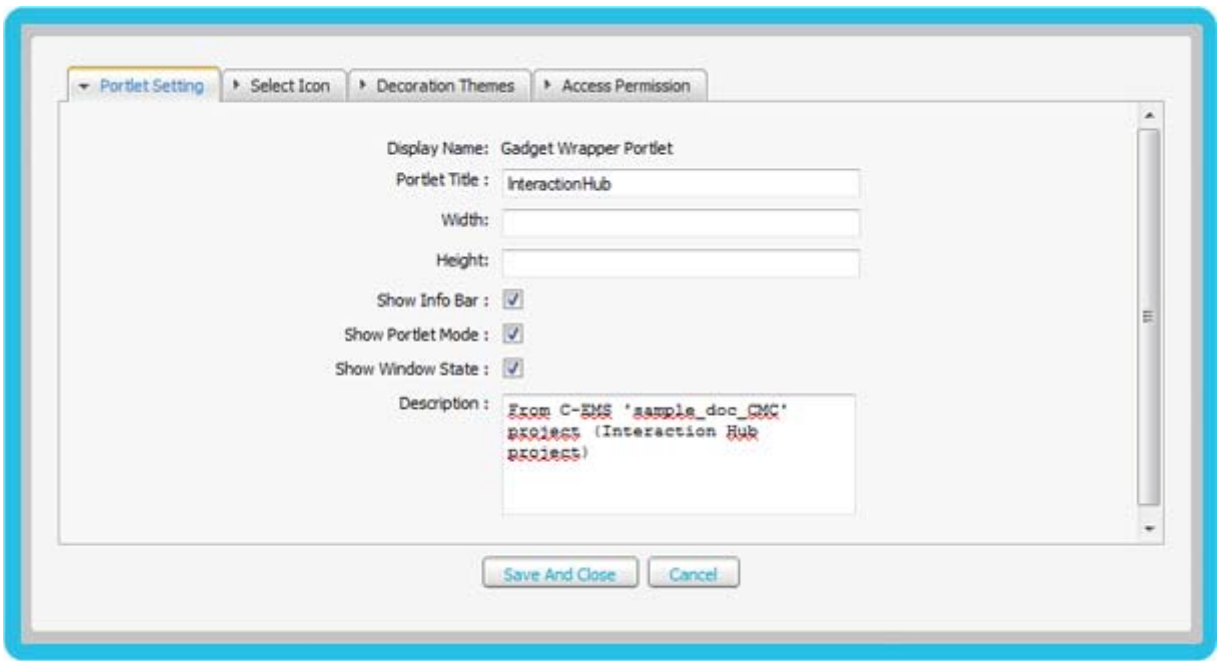

Figure 2 - 105: InteractionHub widget's settings

5 Set a very small size value in **Height** field (for example set `1px`).

6 Uncheck the three checkboxes **Show Info Bar**, **Show Portlet Mode**, **Show Window State**.

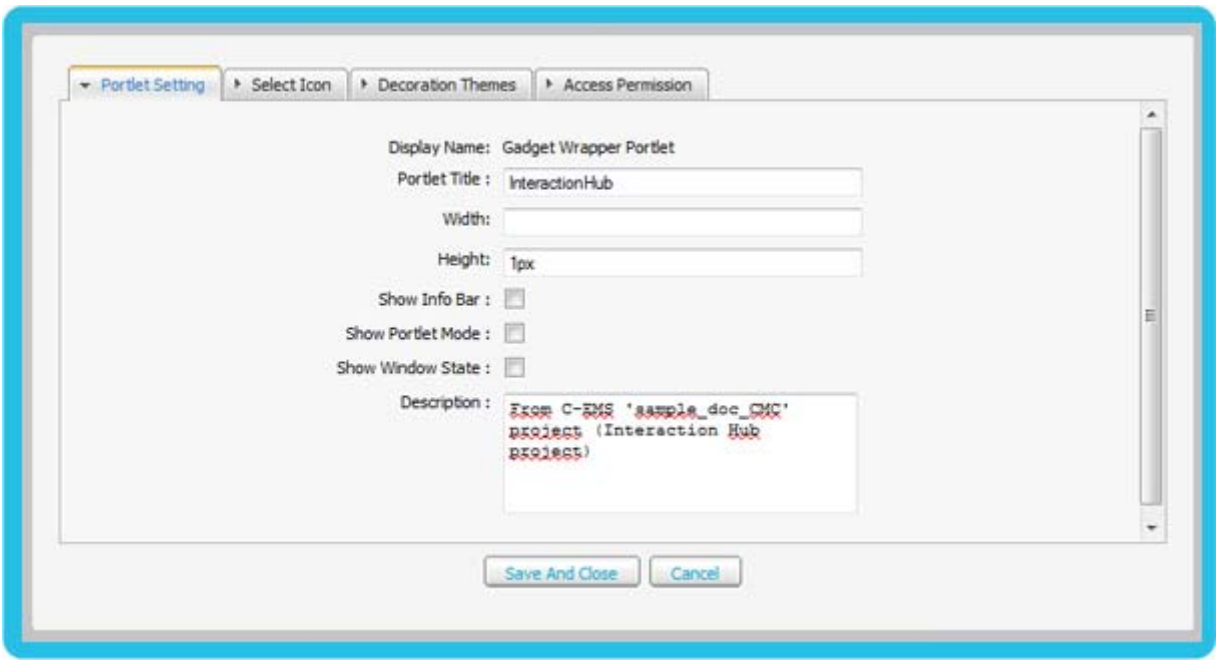

Figure 2 - 106: Edited InteractionHub widget's settings

7 Click on the **Save and Close** button to validate the modifications and close the settings form.

8 Click on the **Finish** button of the ***Page Editor*** toolbar to save the edited page:

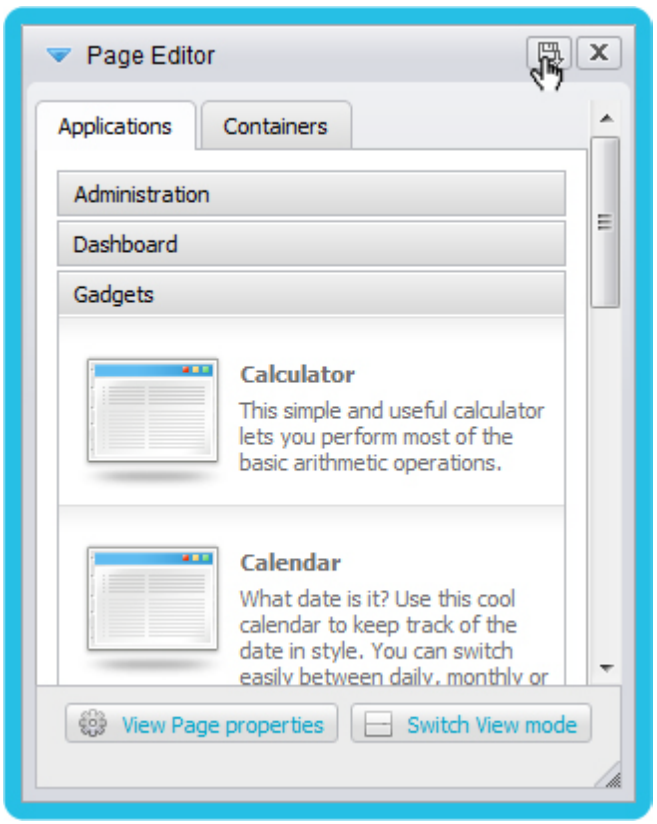

Figure 2 - 107: Clicking on finish button

The updated page is displayed, with the two left visible widgets.

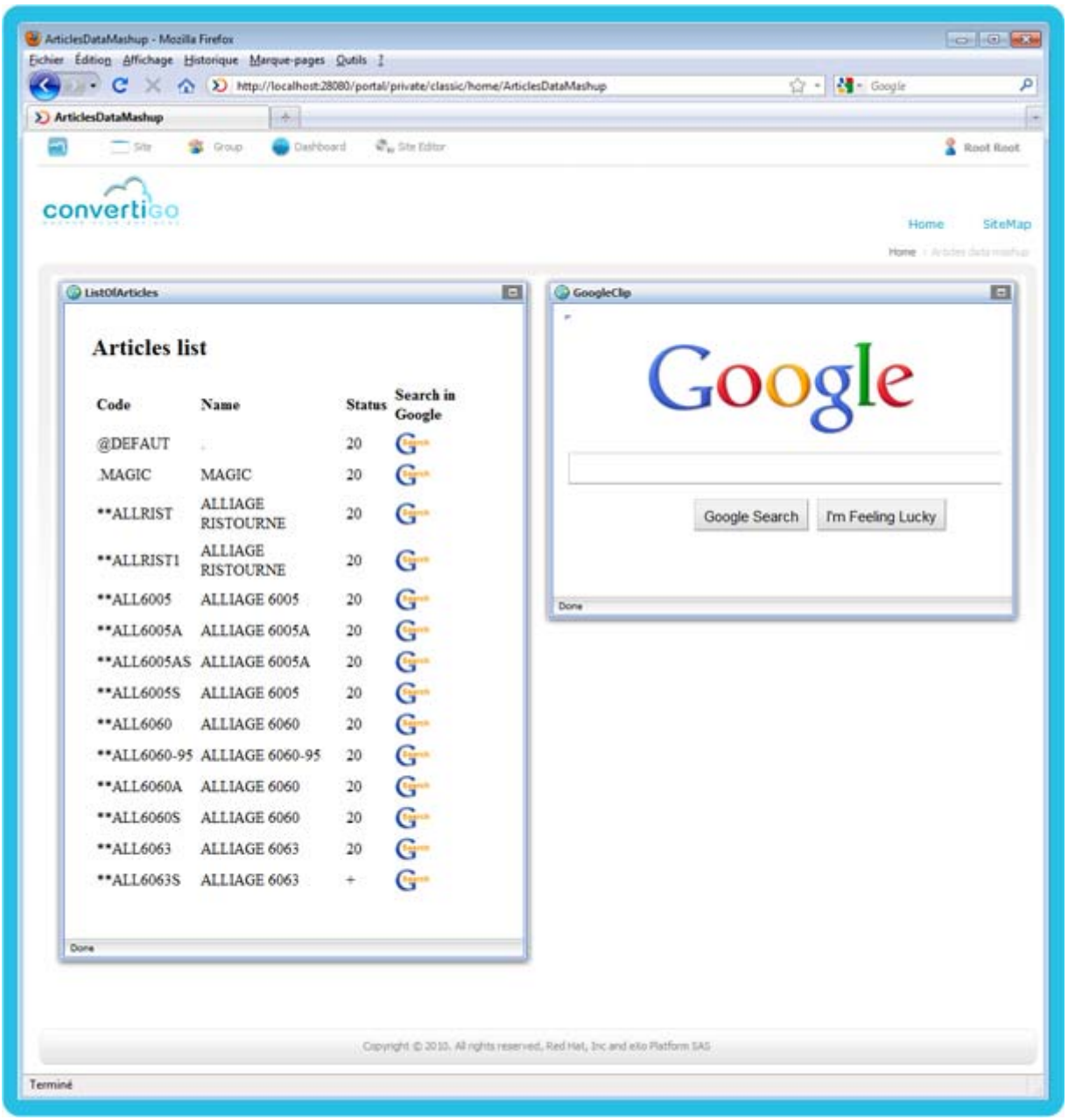

Figure 2 - 108: Hidden InteractionHub widget

This procedure explained how to hide the `InteractionHub` widget from the *Article data mashup* page to improve its display.

The following procedures explain another way of improving the page's display: to display a logo or banner in the `InteractionHub` widget.

To edit the HTML template file of interaction hub project on the Convertigo Studio

1. In the Convertigo Studio, open the interaction hub project (for example `sample_doc_CMC`).
2. In the **Projects** view, click on the **Project Explorer** tab.
3. Expand the `sample_doc_CMC` project folder, then `hub` folder.
4. Find the HTML template file named `default_hub.html`.

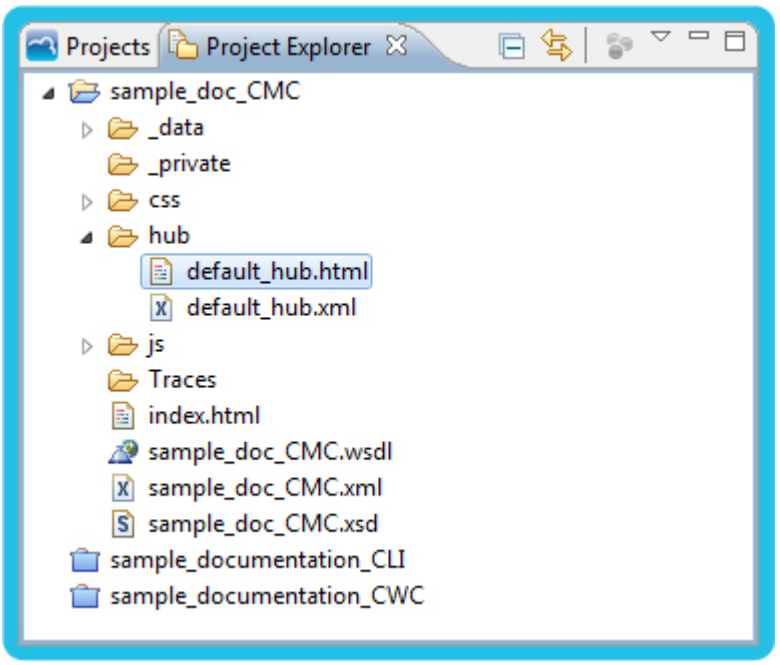

Figure 2 - 109: CMC project file tree

5 In the **Project Explorer**, double-click on the `default_hub.html` HTML file to open it in an editor.

Figure 2 - 110: default_hub.html file in editor

We will now edit this HTML file so that it displays a logo. To this end, an `IMG` element is needed, with the image file to display.

6 In the HTML page, add an `IMG` element with the following attributes:

- *required* attributes - `src` and `alt`;
- *standard* attributes - `title`.

Set these attributes according to the following table:

Table 2 - 4: `IMG` element attributes

Attribute	Description	Value
`src`	Source attribute. Relative path to image.	(for example) `"images/convertigo-small.gif"`

Table 2 - 4: `IMG` element attributes (...)

Attribute	Description	Value
`title`	Image title. Text displayed as the mouse pointer moves over the object.	(for example) `"Convertigo"`
`alt`	Alternative text. Text displayed when the `IMG` element cannot be displayed.	(for example) `"Convertigo"`

For more information on the required, optional, standard and event attributes of the HTML <img> tag, see the W3Schools dedicated page at: http://www.w3schools.com/tags/tag_IMG.asp

*To help you edit the HTML file, a ready-to-use `default-hub.html` file is present in the sample project. You can access it via the **Project Explorer** tab, in the `sample_documentation_CMC` project folder.*
You can copy its content and paste it in the `default-hub.html` file (see Figure 2 - 111).

Figure 2 - 111: Edited default-hub.html

An asterisk appears to the left of the tab file name, indicating that the file is being edited:

7 Save it by clicking on the **Save** icon of the CMC application or by pressing `Ctrl+S`.

8 The folder with the image file referenced by the `IMG` tag have to be created in the `sample_doc_CMC` project.

*To help you, a ready-to-use `images` folder with a convertigo-small.gif file is present in the sample project. You can access it via the **Project Explorer** tab, in the `sample_documentation_CMC` project folder.*
You can:
- copy the complete `images` folder and paste it in the `sample_doc_CMC` project folder (see Figure 2 - 112).
- create an `images` folder and paste any image into it (don't forget to update the `IMG` element `src` attribute in the `default-hub.html` file with the correct image file name).

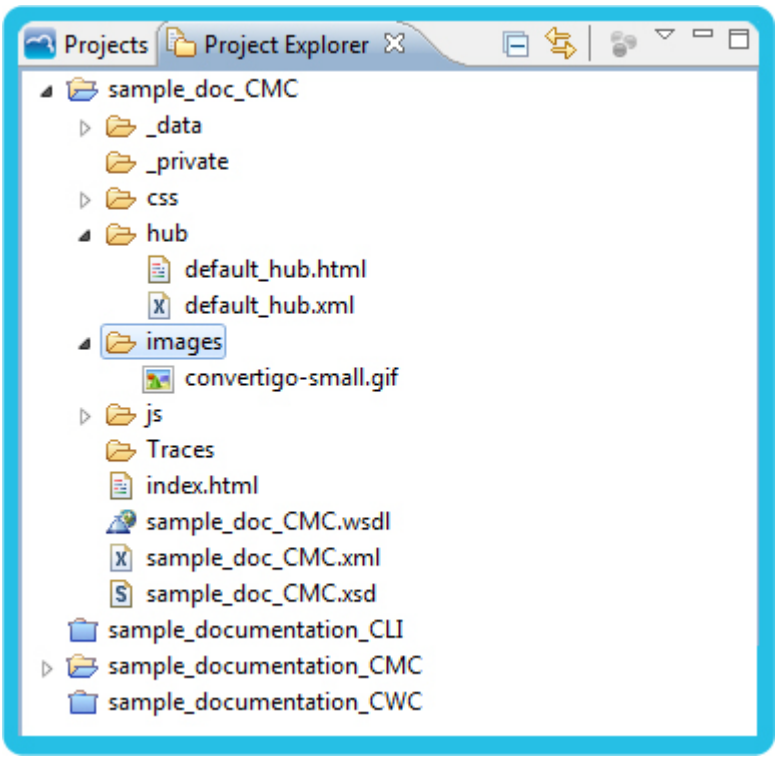

Figure 2 - 112: Images folder in sample_doc_CMC project

The interaction hub project is now updated. The next step is to deploy this `sample_doc_CMC` project to the Convertigo Server.

9 Even though a project already exists in the Convertigo Server, it can be deployed again. To do so, follow the steps of the procedure see *"To deploy a project from Convertigo Studio to Convertigo Server"* on page 2-23.

10 Switch to the browser where is opened the Mashup Composer and refresh the *Articles data madshup* page. The `InteractionHub` widget displays the expected image:

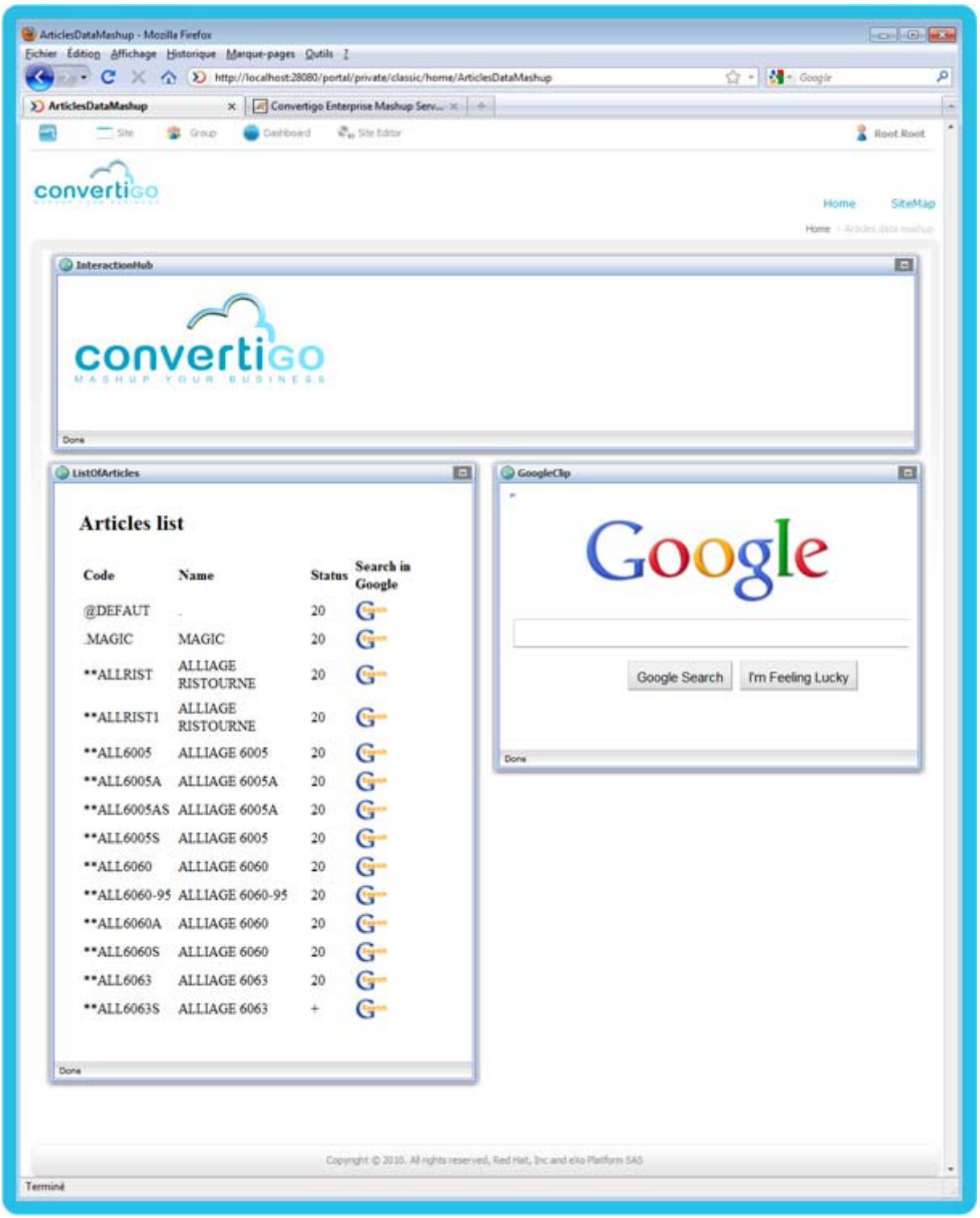

Figure 2 - 113: InteractionHub widget displaying image

2.5.2 Testing the Composite Application

Now that both widgets have been finalized in the server, we can test the Composite Application. Switching to the browser where is opened the Mashup Composer, the *Articles data madshup* page displays the widgets in "started" state, all containing expected content.

Depending on the chosen improvement of `InteractionHub` widget:

- two widgets are displayed:

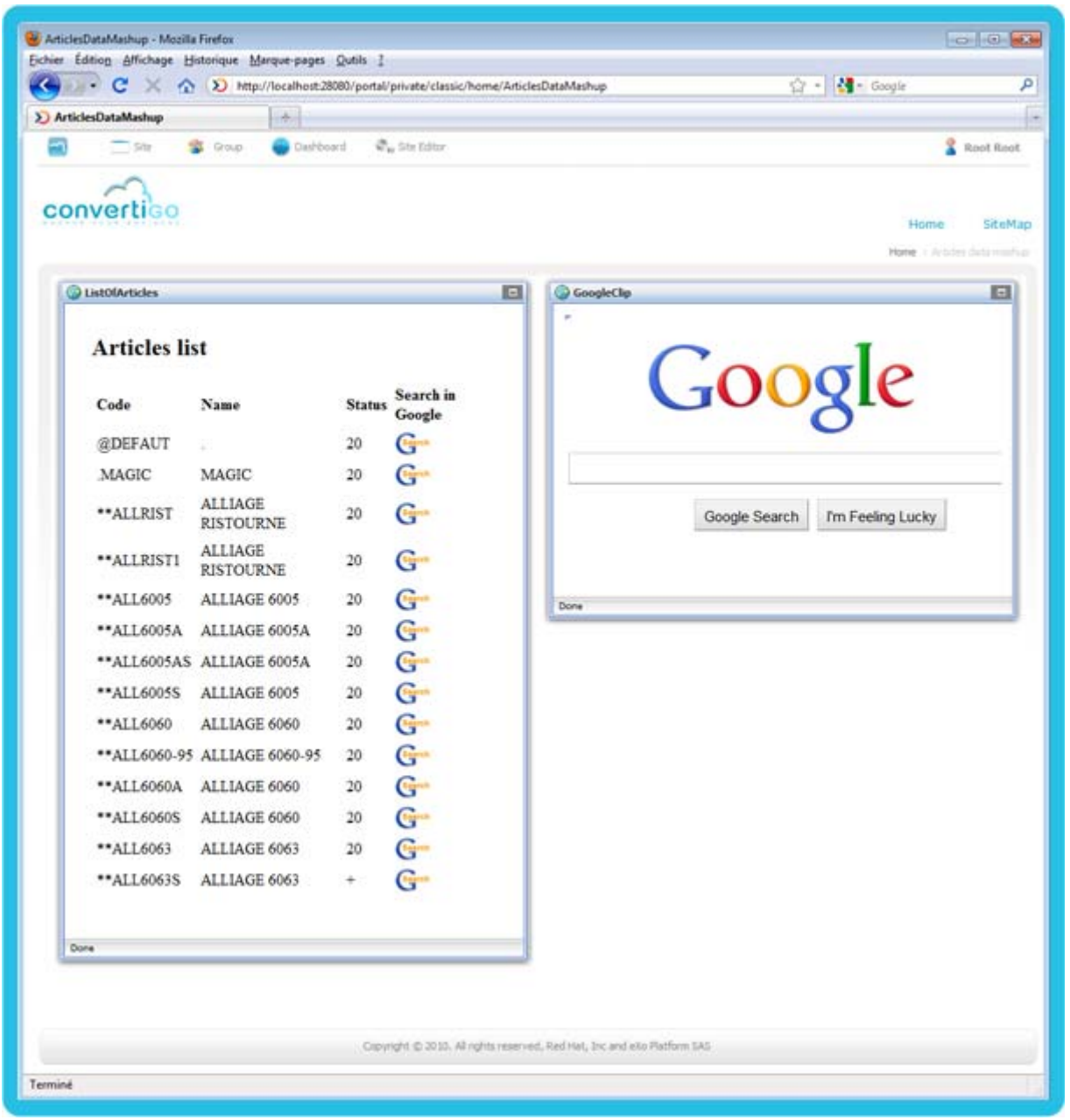

Figure 2 - 114: Articles data mashup page displaying two widgets in "start" state

- or three widgets are displayed:

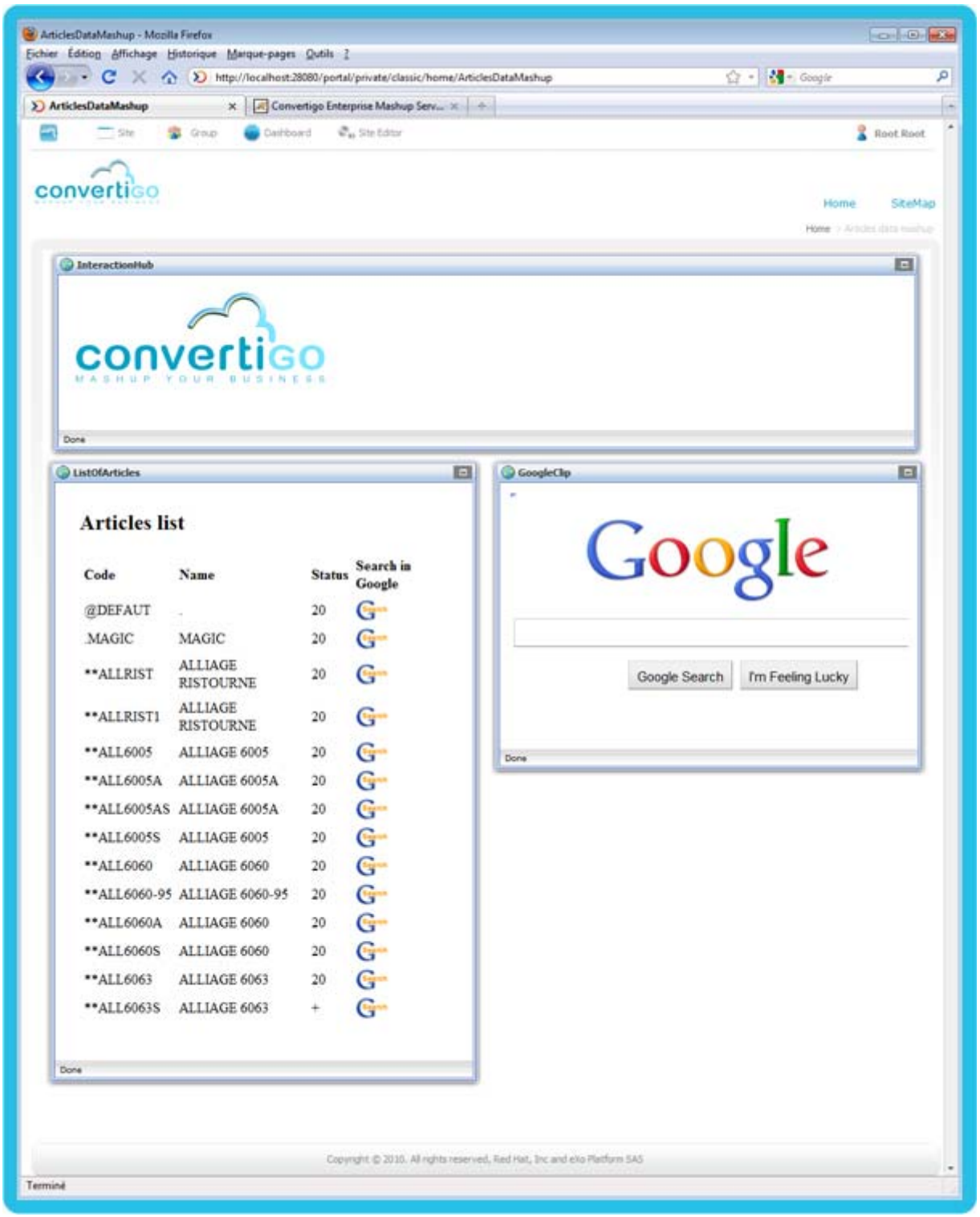

Figure 2 - 115: Articles data mashup page displaying three widgets in "start" state

You can now try clicking on one of the Google icons in the **Search in Google** column of the `ListOfArticles` widget.

A Google search based on the value contained in the corresponding **Name** column of the *Articles list* table is automatically launched in the `GoogleClip` widget. Google results are displayed in the same widget:

Figure 2 - 116: Test result