convertigo
MASHUP YOUR BUSINESS

Starting with Convertigo Web Clipper

Quick Guide

CEMS 5.1.0

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Getting Familiar with Convertigo Web Clipper

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

- CWC Purpose
- CWC Concepts and Objects
- CWC Studio

1.1 CWC Purpose

The purpose of Convertigo Web Clipper (CWC) is to "clip" parts of a target application (a Web page for example) to generate "cliplets" that can be integrated into any business portal:

Figure 1 - 1: CWC "clips" parts of target applications to be integrated into business portals

This technology is known as "Web clipping".

Convertigo Web Clipper supports *application* clipping and not just *page* clipping. Application clipping enables users to get clipped responses from clipped pages instead of having the response displayed unclipped in a new window or a new frame. The functionnalities of clipped applications are thus preserved.

For example, if the target application supports column sorting, then the resulting cliplet also supports column sorting:

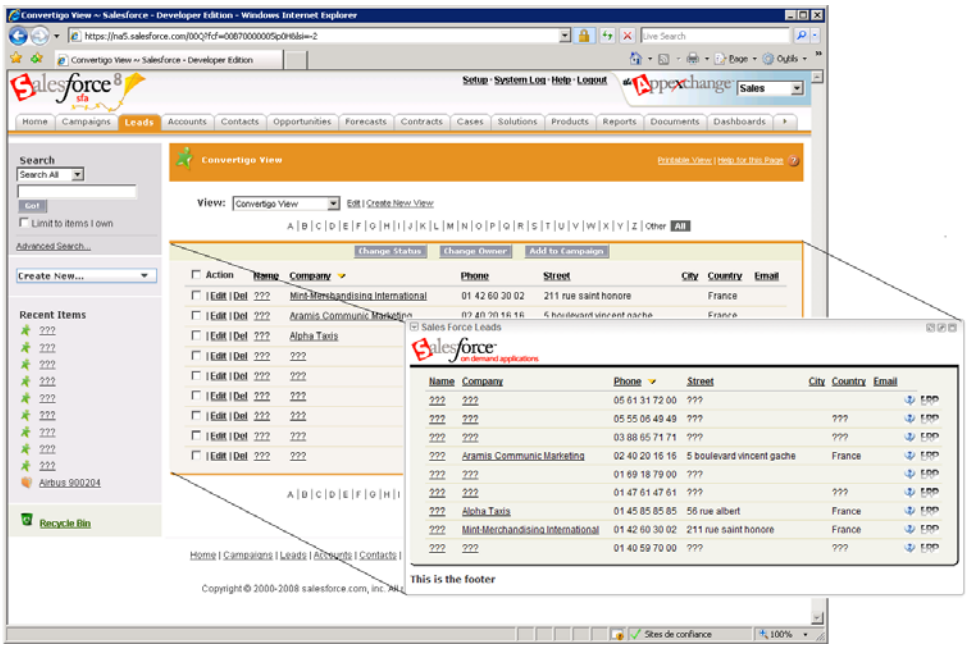


Figure 1 - 2: Web clipping preserves functionalities

CWC encompasses a large number of applications:

- *Creating* Widgets by clipping existing Web applications.
- *Saving* substantial amount of time by re-using business logic and presentation from existing Web applications.
- *Integrating* partner's Web applications into your corporate applications.
- *Combining* in mashups new and existing business logic and presentation.
- Non intrusively *refacing* existing Web applications to fit the corporate “look & feel”.
- Non intrusively *resizing* or *changing* layouts of existing Web applications to fit new presentation media such as PDAs or smartphones.

1.2 CWC Concepts and Objects

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

1.2.1 General Concept

CWC accesses a target application via a *connector*. It then navigates through application *screens* by detecting to which *screen class* screens belong (based on detection *criteria*) and executes associated *extraction rules*. In CWC, extracting data means "clipping" user-selected

elements of an HTML page on the fly.

The clipping process results in the generation of a "cliplet". A cliplet is a widget made of clipped HTML elements. Its structure follows the order in which HTML parts of Web pages have been clipped - for example, if a logo is clipped before a search field, the logo will appear in the cliplet above the search field, and vice-versa.

Navigation and clipping depend on events, actions and rules corresponding to CWC objects, that are set as part of a Web clipping project. Navigation is automated via the definition of a *transaction*, which includes *screen class handlers* triggering *statements* (i.e. actions) on detection of *screen classes.*

1.2.2 CWC Objects

The following table describes CWC objects:

Table 1 - 1: CWC objects

Object[a]	Description
Connector	Defines the type of application (HTTP, HTML, etc.) CWC connects to.
Screen class	Defines a family of HTML pages with common characteristics.
Criterion	Is a characteristic element defining a specific screen class.
Extraction rule	Is a rule defining how information is extracted from a Web screen.
Transaction	Defines the actions Convertigo must perform on HTML pages.
Screen class handler	Is a set of statements triggered on detection of a specific screen class.
Statement	Is an action or set of actions performed on HTML pages.

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

All of these objects are set when developing a Web clipping project. Once the project has been created, these objects appear in the **Projects View** of the CWC Studio (*“CWC Studio”* on page 1 - 9):

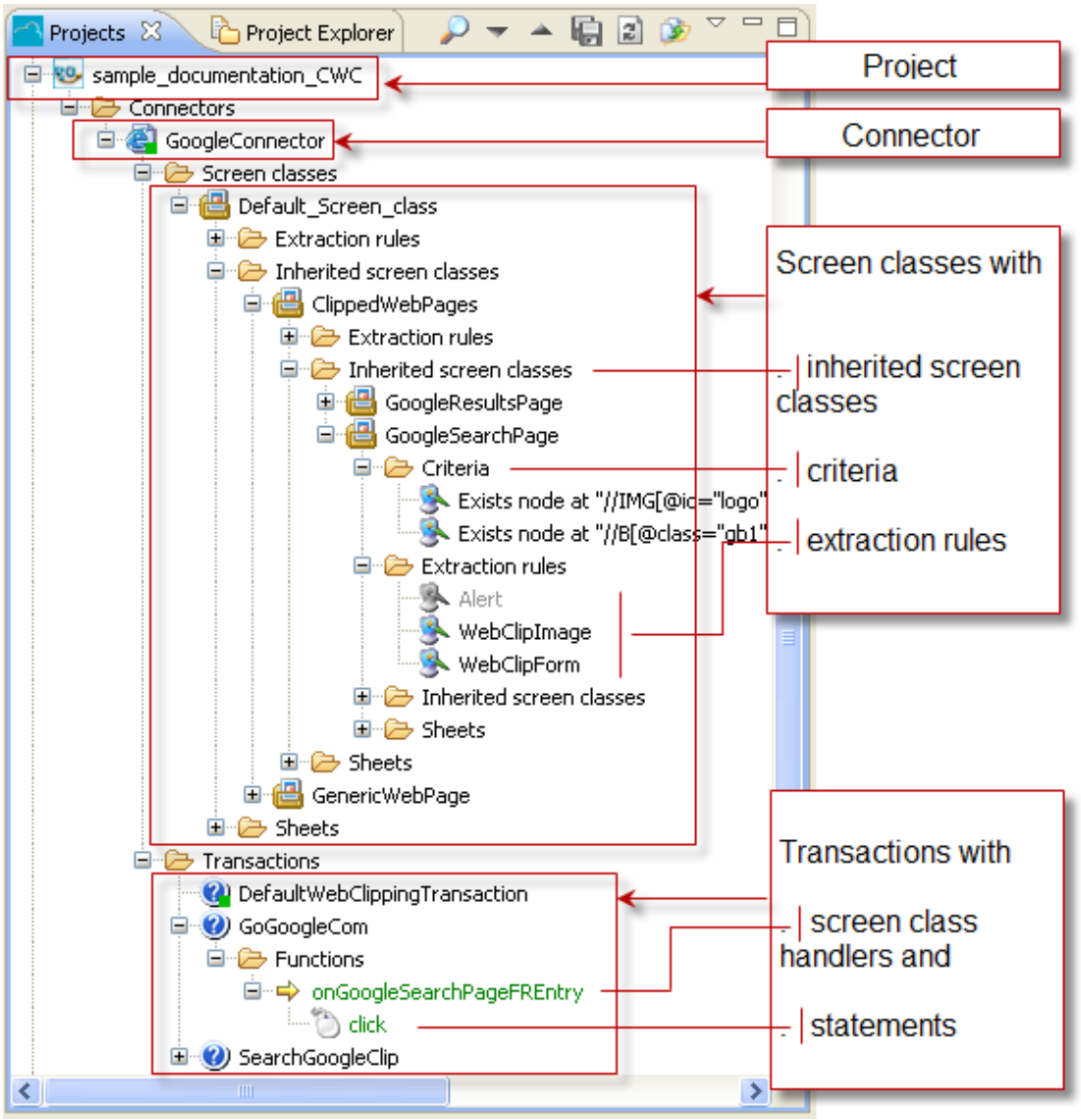


Figure 1 - 3: CWC object

To get started with your first Web clipping project, see *"My First Convertigo Web Clipper Project" on page 2-1*

1.2.3 Definitions

CONNECTOR

A connector defines the type of application or data source CWC is able to connect to.

One type of connector is available in CWC: the *HTML Connector*, which enables Convertigo to connect to any HTML-based application.

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

Once a new project has been created, the corresponding connector appears in the *Connectors* folder of the project, in the **Projects View** of the CWC Studio (*"CWC Studio"* on page 1 - 9):

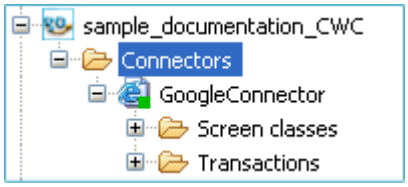


Figure 1 - 4: Connectors folder

Screen Class

Screen classes define families of Web screens (Web page, application form, etc.) with common characteristics (called *criteria*). Screen classes are:

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

A single root screen class, called ***Default screen class***, is created by default when creating a project. The default screen class is not - but can be - associated with any criterion or extraction rule. It is parent to all other screen classes.

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

In CWC projects, the `Default_Screen_Class` has two default child screen classes:

Table 1 - 2: CWC default inherited screen classes

Screen class	Description
`ClippedWebPages`	Any screen class defined by the user must be child to the `ClippedWebPages` screen class.
`GenericWebPage`	Any screen not detected as a user-defined screen class is detected as a `GenericWebPage` screen class.

The two default child screen classes (`ClippedWebPages` and `GenericWebPage`) appear in the **Projects View** in the **Inherited screen classes** folder of `Default_Screen_Class`:

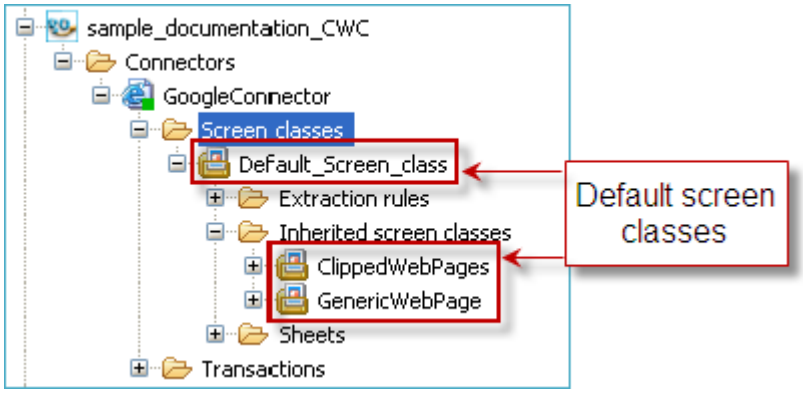


Figure 1 - 5: Default CWC Screen classes

The `ClippedWebPages` and `GenericWebPage` screen classes inherit their parent screen class (`Default_Screen_Class`) *extraction rules* and *sheets* (style sheet associated with a screen class). As *inherited objects*, these objects appear greyed (see Figure 1 - 15) in the **Projects View**:

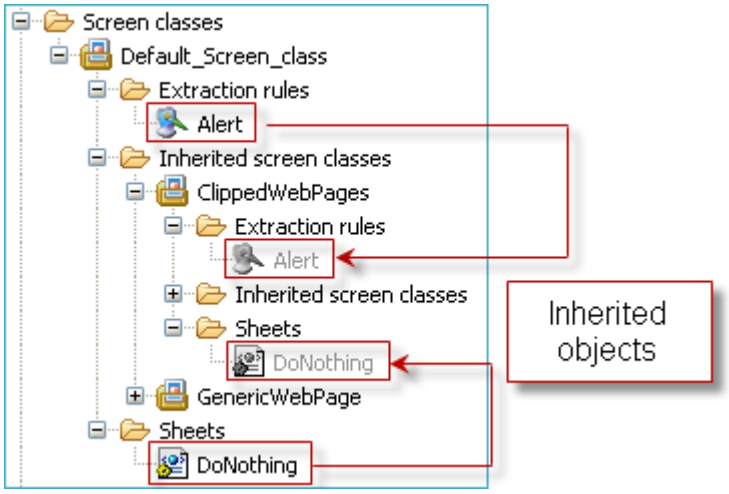


Figure 1 - 6: Objects inherited from the Default screen class

For more information on the DoNothing style sheet, see "The `DoNothing` Style Sheet" on page 3-1

Inherited screen classes appear in subfolders named *Inherited screen classes* in the **Projects View** of the CWC Studio (*"CWC Studio"* on page 1 - 9):

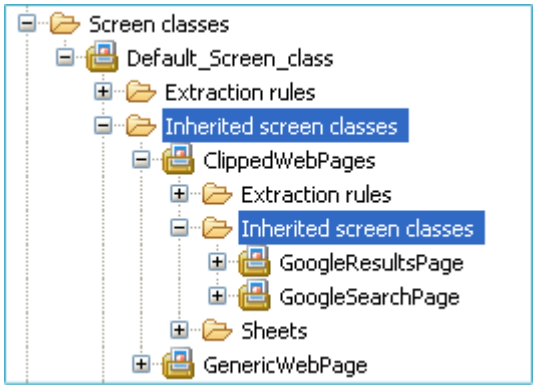


Figure 1 - 7: Screen classes and Inherited screen classes folders

Criterion

A Web screen is considered belonging to a screen class when it matches all criteria defined by the user for this screen class. If no criterion matches, the Web screen is considered belonging to the default screen class (provided that the default screen class is not associated to any detection criterion). However, every page should at least match the `GenericWebPage` screen class (provided that the `GenericWebPage` screen class is associated to a criterion that does match on every HTML page).

Criteria are set using the ***XPath language***, which addresses parts of an XML document; an XPath can be, for example, the path to a logo, a field, a word or any other identifying element on a Web screen. If at least one element is adressed by the XPath on the HTML page, the criterion matches.

XPaths are managed in the CWC Studio using the **XPath Evaluator**, which is part of the **Connector View** of the CWC Studio (*"CWC Studio"* on page 1 - 9).

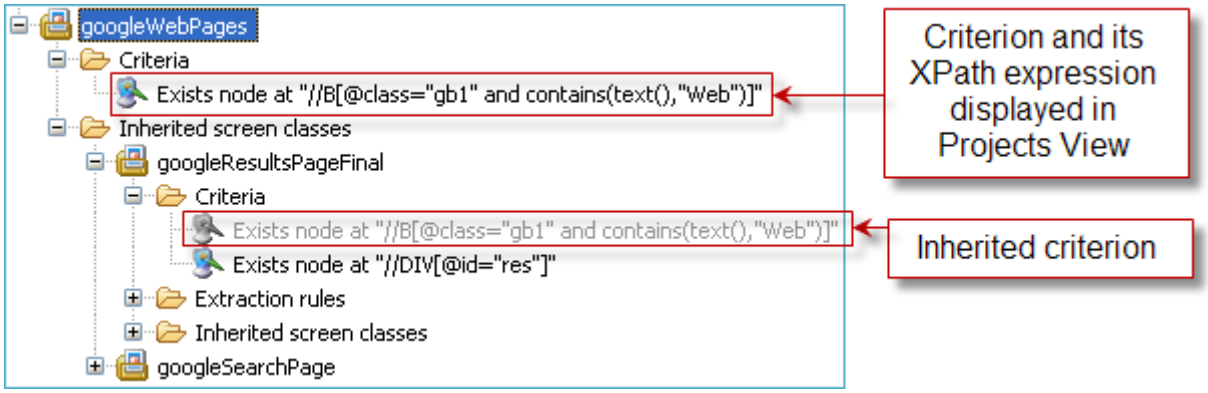


Figure 1 - 8: Criterion and inherited criterion

For more information on XPath, see the XPath tutorial from W3 schools at http://www.w3schools.com/XPath/.

The `ClippedWebPages` screen class does not have any detection criteria, since it is never *detected* stricto sensu (it only contains screen classes that can be detected).

The `GenericWebPage` screen class has one detection criterion called `Exists node at "/HTML"`:

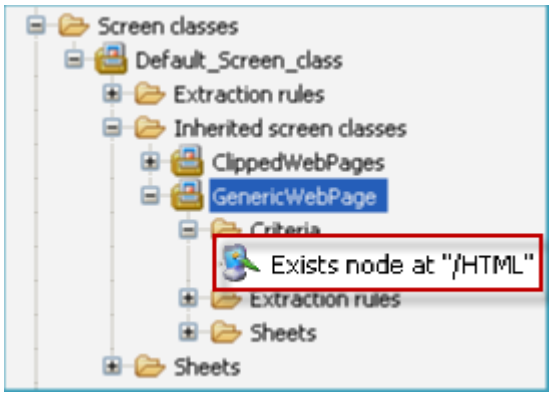


Figure 1 - 9: GenericWebPage screen class - XPath criterion

This XPath (defined as `/HTML`) should match on *any* HTML Web page. Any screen not matching criteria of screen classes inherited from `ClippedWebPages` is therefore detected as a `GenericWebPage` screen class.

Extraction rule

Extraction rules define which data must be extracted (or "clipped" in CWC) from the target application. They are used to clip parts of an HTML document.

During a *transaction* process, when a screen class is detected (i.e. all criteria defined for this screen class match), all screen class-specific extraction rules -including those of its parent screen class - are executed (and corresponding HTML parts clipped) according to their order in the **Extraction rules** folder of the screen class.

The `GenericWebPage` screen class contains a specific *extraction rule* called `WebClipper`:

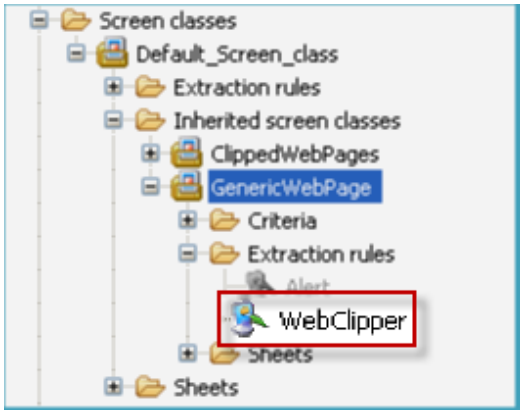


Figure 1 - 10: GenericWebPage screen class - WebClipper extraction rule

This extraction rule extracts the whole content of screens detected as belonging to the `GenericWebPage` screen class.

TRANSACTION

Web page navigation is performed using HTML transactions, that is to say a series of actions (called *statements*) performed from one Web screen to another. These actions are triggered by events associated with the kind of screen class accessed and detected, using *screen class handlers*.

Once a transaction has been defined, it appears in the *Transactions* folder of the project, in the **Projects View** of the CWC Studio (*"CWC Studio"* on page 1 - 9):

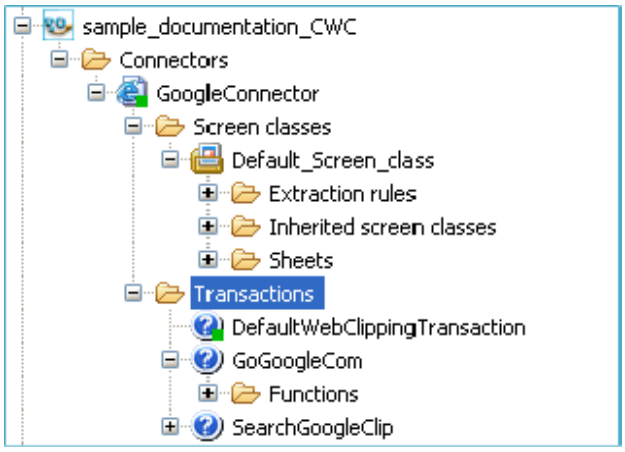


Figure 1 - 11: Transactions folder

SCREEN CLASS HANDLER

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

In other words, once a given screen class has been accessed and detected (depending on user-defined criteria), the corresponding screen class handler is invoked and triggers user-defined actions.

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

Table 1 - 3: Screen class handler types

Screen class handler	Description
on<*ScreenClass*>Entry	Invoked when CWC detects a given <***ScreenClass***>. Is identified by a  symbol on the left of the screen class handler name (see Figure 1 - 12).
on<*ScreenClass*>Exit	Invoked after extraction rules have been executed for a given <***ScreenClass***>. Is identified by a  symbol on the left of the screen class handler name (see Figure 1 - 12).

Other handlers exist, but they fall out of the scope of this guide.

Once a screen class handler has been created, it appears in the *Functions* folder of the transaction, in the **Projects View** of the CWC Studio (*"CWC Studio"* on page 1 - 9):

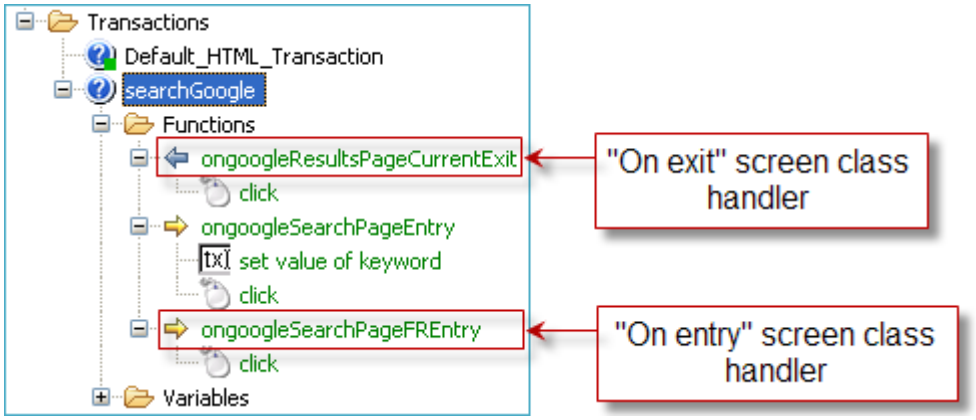


Figure 1 - 12: "On entry" and "on exit" screen class handlers

A screen class handler can contain one or several *statements*. For example: Type in a keyword (statement #1) then click on ***Next*** (statement #2):

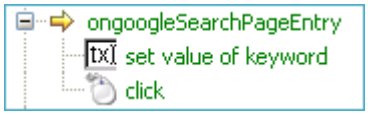


Figure 1 - 13: List of statements

STATEMENT

A statement is an action that is performed on **entry** to or on **exit** from a Web screen, provided that the screen class of this Web screen has been detected. Statements include mouse clicks, data inputs, etc.

1.3 CWC Studio

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

1.3.1 General Presentation

Based on an Eclipse platform, the CWC 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:

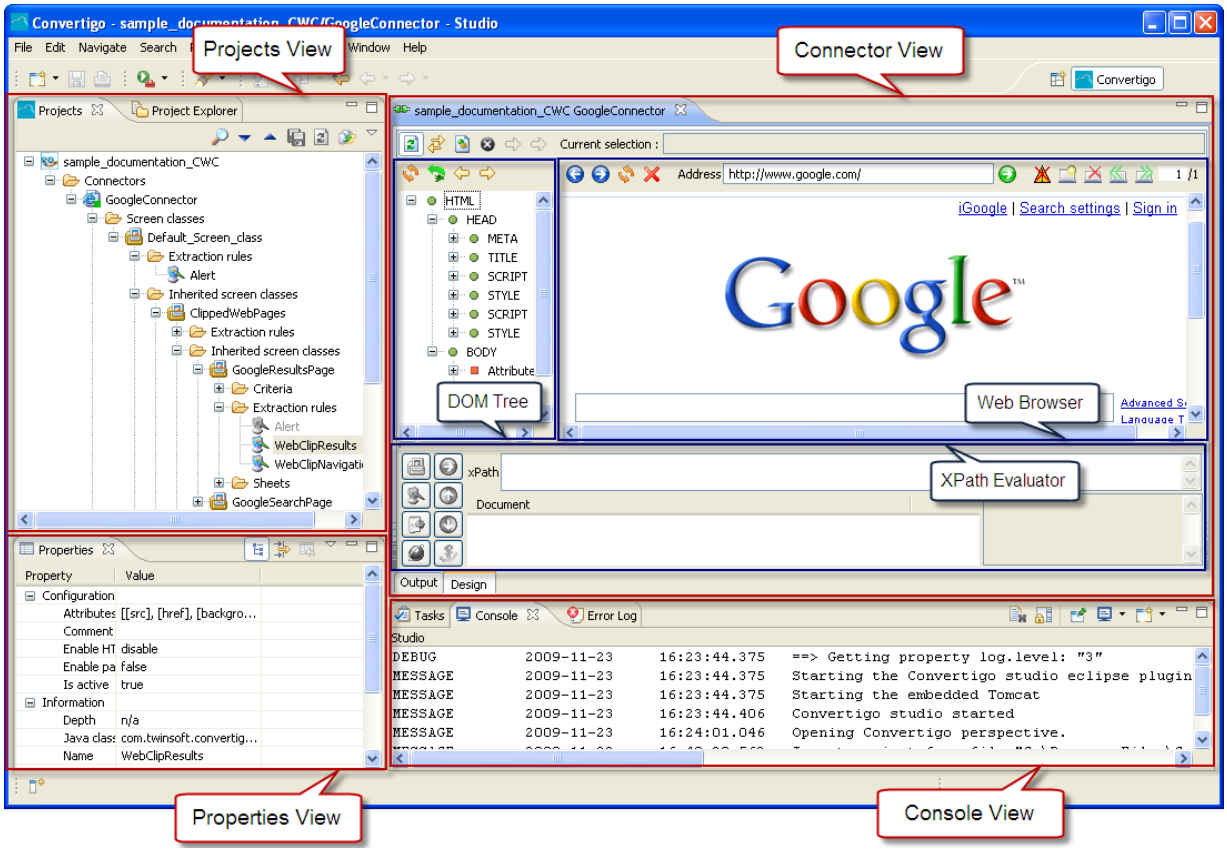


Figure 1 - 14: CWC Studio

The following table describes the views of the Convertigo Web Clipper studio:

Table 1 - 4: CWC views

View Name	Description
Projects View	Displays in a tree structure all CWC projects and their related objects: connectors, screen classes (including criteria, extraction rules and inherited screen classes) and transactions (including screen class handlers and statements).
Properties View	Displays the properties of the CWC object selected in the **Projects View**. Used to modify a given object's properties.

Table 1 - 4: CWC views (...)

View Name	Description
Connector View / Editor	The **Connector View** is divided into three parts: • the **DOM Tree** - displays the Document Object Model Tree (logical structure) of the Web page accessed by CWC; • the **Web Browser** - displays the Web page accessed by CWC, depending on the connector parameters set for the project; • the **XPath Evaluator** - displays XPath-specific information such as the XPath expression of a node. Contains shortcuts (icons) for creating new children screen classes, criteria, extraction rules, statements, etc. It is automatically updated as an **Editor** whenever 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 - 5: Standard Eclipse buttons

Button	Icon	Description
Menu		Opens a drop-down menu with a content 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 CWC 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 - 5 on page 1-11), the **Projects View** includes tabs and icons:

Table 1 - 6: Projects View tabs and icons

View Element		Description
Tabs	Project	Convertigo view. Contains all projects and related objects. Objects are presented in a tree structure and contained in folders.
	Project Explorer	Standard Eclipse view. Contains all projects and related files. Files are represented in a tree structure.
Icons		*Find* icon. Search project tree for a specific object
		Decrease priority icon. Available only if an extraction rule is selected. Decreases its priority, thus changing extraction rules execution order for this screen class and its children classes.

Table 1 - 6: Projects View tabs and icons (...)

View Element	Description
	Increase priority icon. Available only if an extraction rule is selected. Increases its priority, thus changing extraction rules execution order for this screen class and its children classes.
	*Save* icon. Available only when at least one object has been modified. Saves all the project. All objects will be saved, and project xml file will be updated. **Remember to save projects on a regular basis.**
	*Refresh* icon. Available only if a project is selected. Refreshes Projects View by reloading project content from the hard drive. Identical to closing and re-opening project.
	*Import* icon. Imports a new project (in CAR or XML format) in the Studio.

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

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

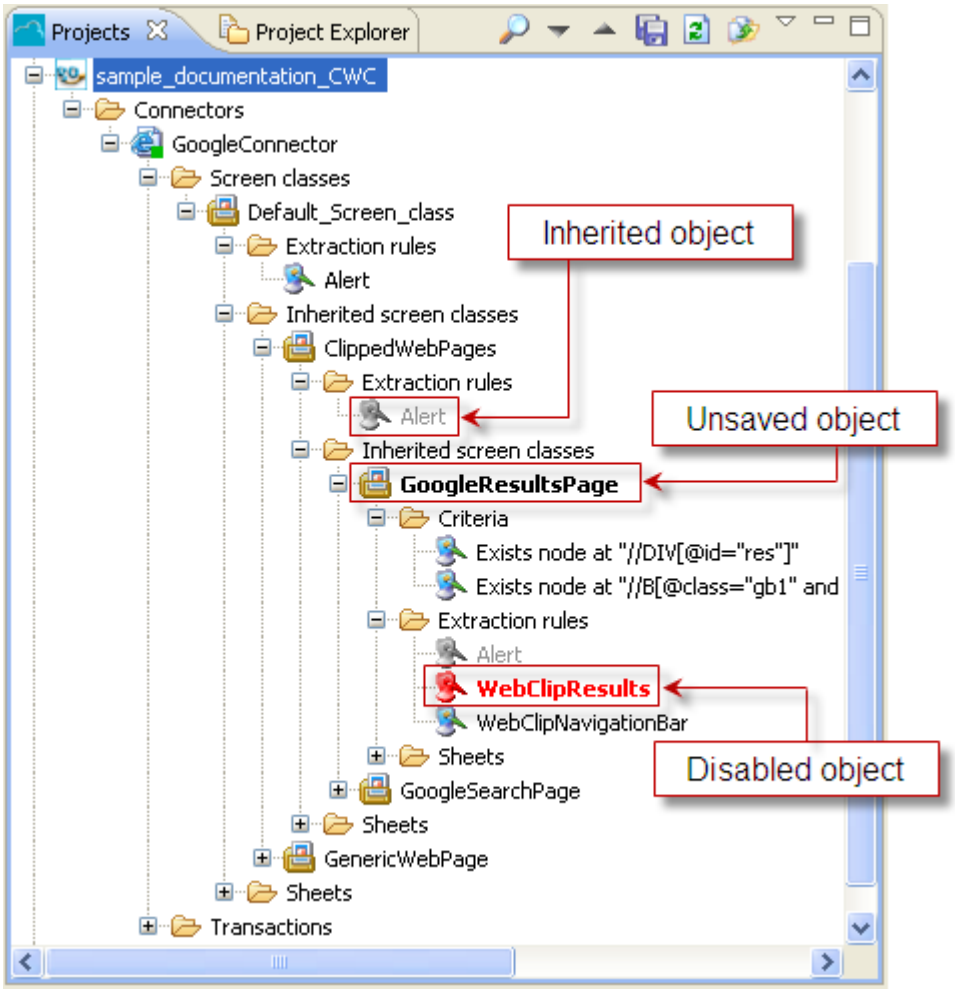


Figure 1 - 15: Formatting of unsaved, inherited and disabled objects in the Projects View

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

Properties View

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

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

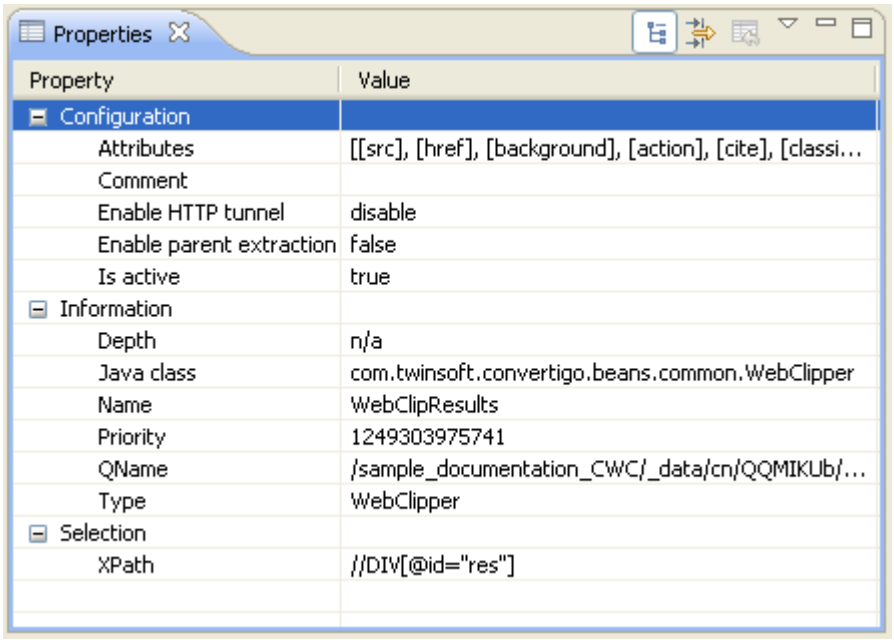


Figure 1 - 16: Properties View

In addition to object properties and standard Eclipse buttons (see Table 1 - 5 on page 1-11), the **Properties View** includes one tab (*Properties*) and icons:

Table 1 - 7: Properties View icons

Icon	Description
	*Show Categories* icon. Displays or hides 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:

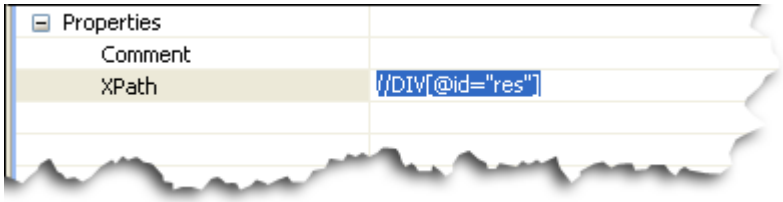


Figure 1 - 17: Text property field edition

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

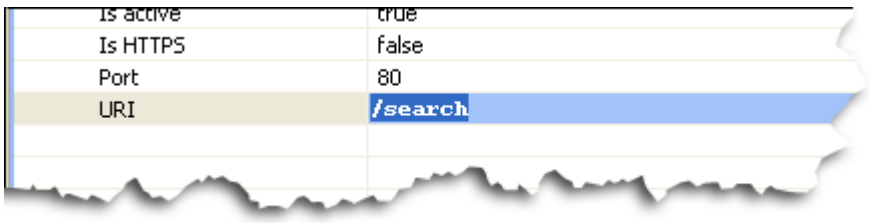


Figure 1 - 18: Javascript expression field edition

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

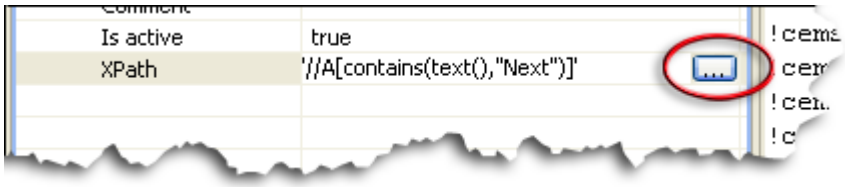


Figure 1 - 19: Text property edition via edition window

Click on  . Depending on the property type:

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

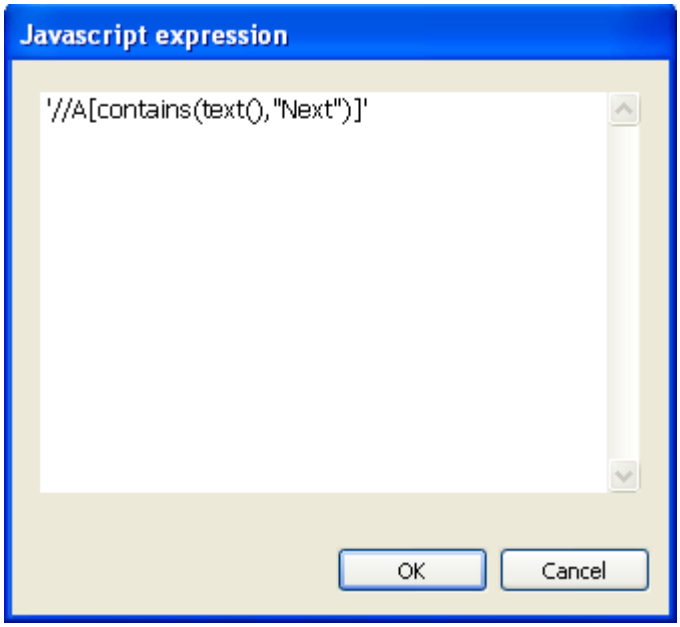


Figure 1 - 20: Javascript expression window

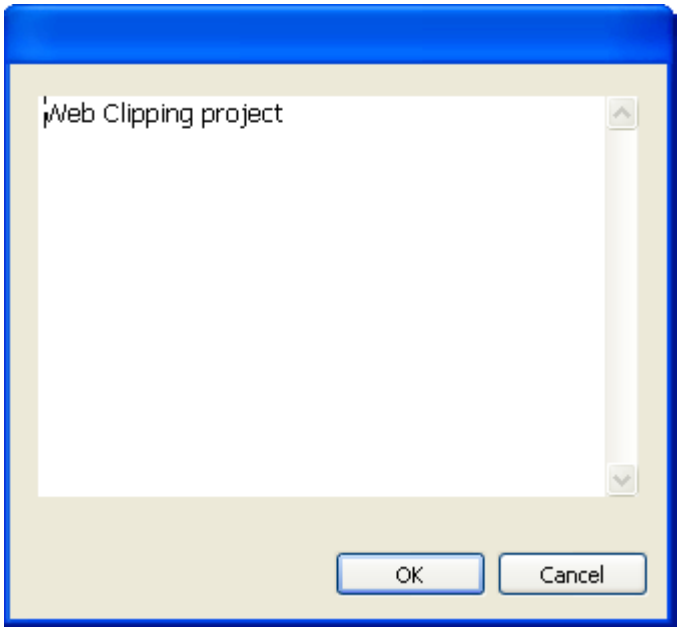


Figure 1 - 21: Edition window

1 Enter the new JavaScript expression or the text comment in the window.

2 Click on **OK**.

B. A configuration window opens (in this example, the **Trigger Editor** window for statement synchronization):

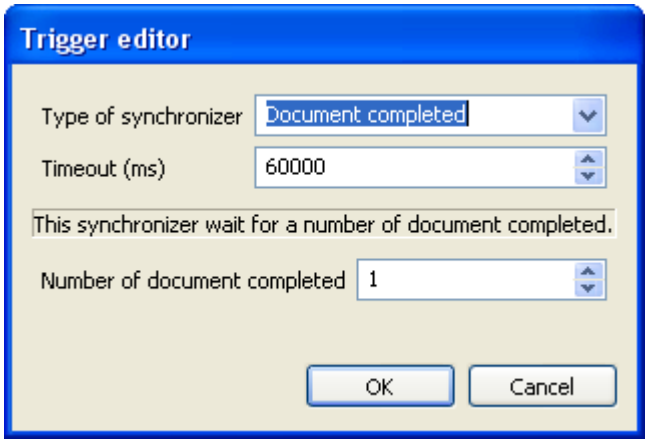


Figure 1 - 22: Configuration window

1 Set the parameters in the different fields and menus.

2 Click on **OK**.

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

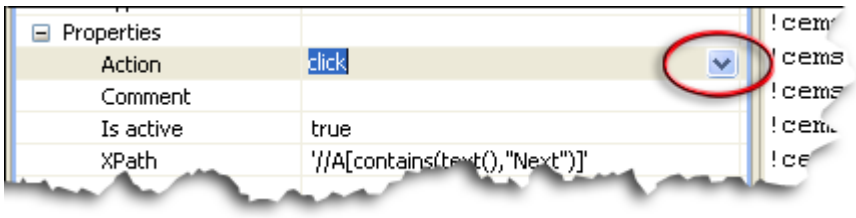


Figure 1 - 23: Text property edition via drop-down menu

1 Click on . A drop-down menu appears:

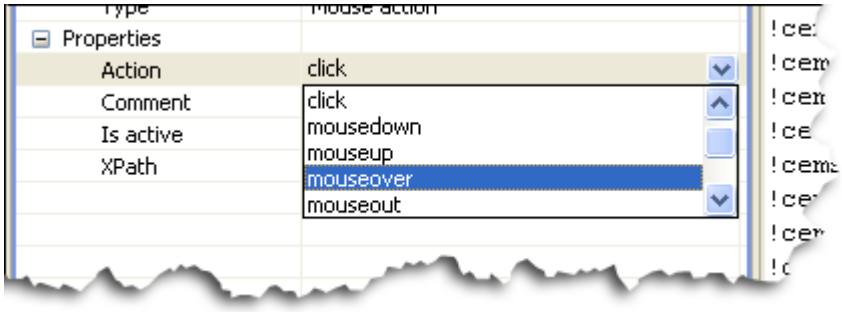


Figure 1 - 24: Selecting a value in the drop-down menu

2 Select a value in the drop-down menu.

3 Press **Enter**.

Connector View

The **Connector View** displays connector-specific information. This information depends on the type of connector set for the project. In the case of HTML connectors, the view is divided into three parts.

The table below describes these parts:

Table 1 - 8: Connector View parts

Name	Description
Dom Tree	Displays the logical structure of the Web page displayed in the **Web Browser**.
Web Browser	Displays the Web page accessed by CWC.
XPath Evaluator	Displays the XPath expression of HTML elements selected in the **Web Browser** and result nodes of the written XPath execution on the **DOM Tree**. Includes shortcuts to create new criteria, statements, screen class, etc.

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

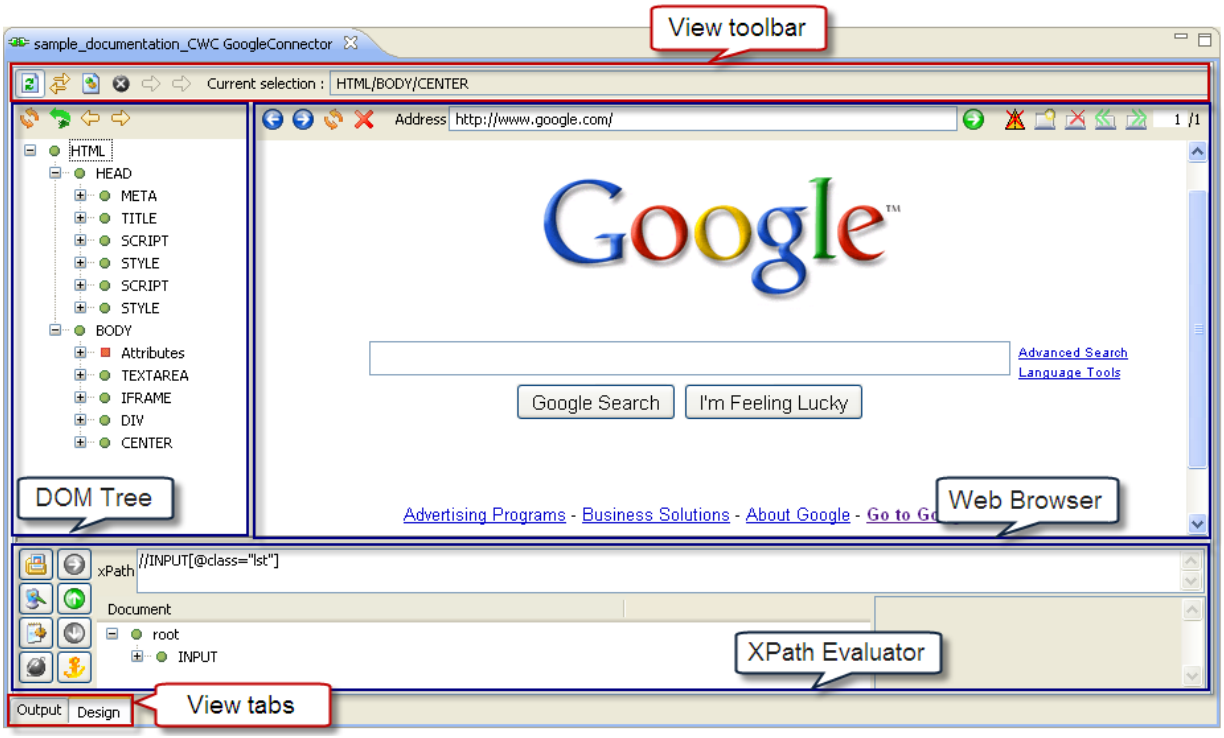


Figure 1 - 25: Connector View (Design tab selected)

The table below describes the Connector View icons that are available when the **Design** tab is selected (by default):

Table 1 - 9: Connector View tabs, icons and field

View Element		Description
Tab	Output	Displays the XML code generated following either a transaction (according to extraction rules) or a *Generate XML* action (see Figure 1 - 26).
	Design	Displays the **Dom Tree**, **Web Browser** and **XPath Evaluator**.
Icon		*Toggle auto refresh domTree* icon. Press this icon to activate the auto refresh domTree function, which automatically refreshes the **DOM Tree** when a new Web page is displayed in the Web Browser.
		*Show current screen class* icon. Highlights in light grey in the **Projects View** the screen class detected for the Web page currently displayed in the **Web Browser**.
		*Generate XML* icon. Generates the XML code of the Web page currently displayed in the **Web Browser** by executing the related extraction rules. The XML code is displayed under the **Output** tab of the **Connector View**.
		*Stop transaction* icon. Stops the current transaction (while a transaction is in process).
		*Learn* icon. Available only when a transaction is selected. Activates the *Learn* mode - any action made on the HTML connector view will be recorded and HTTP statements will automatically be generated in screen classes entry handlers. For advanced users only.

Table 1 - 9: Connector View tabs, icons and field (...)

View Element		Description
		*Accumulate learning mode* icon. Available only when *Learn* mode is on. Accumulates data on visited Web screens. Screen class exit handlers will automatically be created in the transaction. For advanced users only.
Field	Current selection	Displays the path to the node selected in the DOM Tree (left-click) or to the HTML element selected in the Web Browser (right-click).

When the **Output** tab is selected, the view displays XML code:

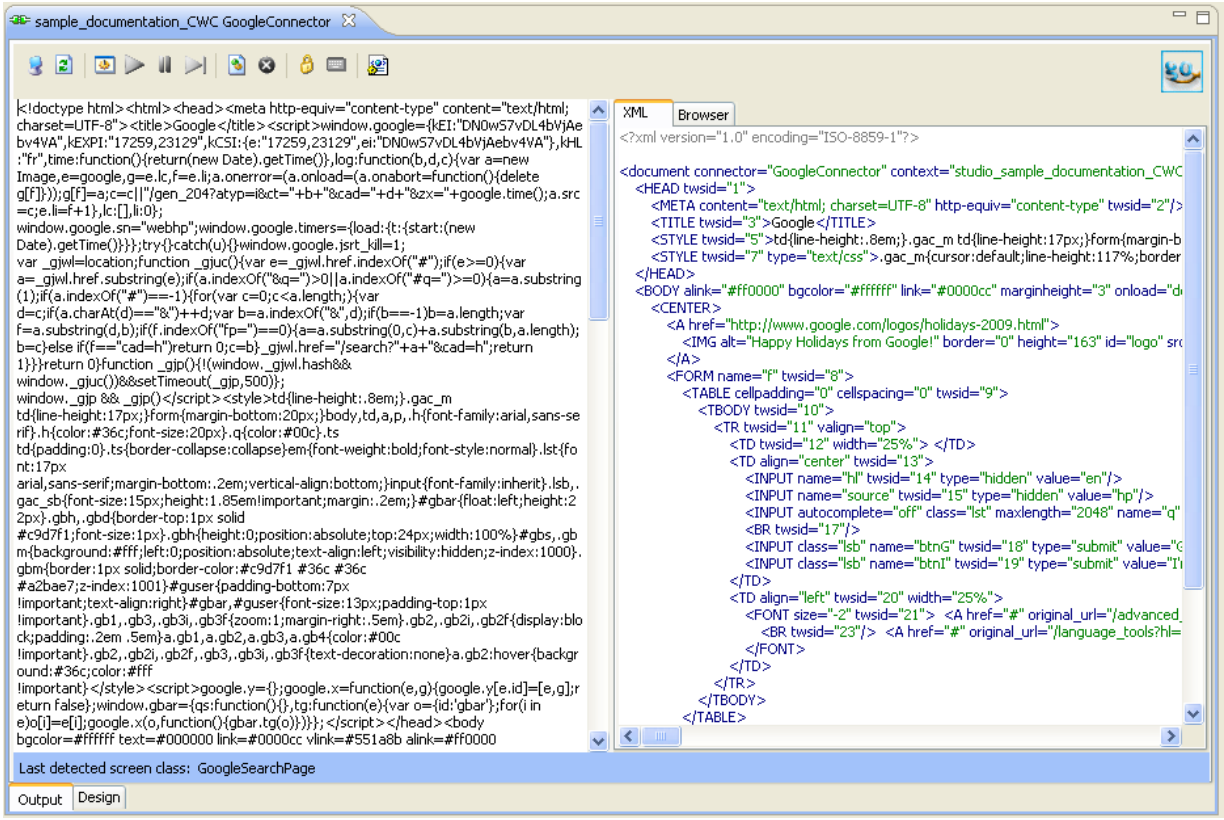


Figure 1 - 26: Connector View (Output tab selected)

DOM TREE

The **DOM Tree** models an HTML (or XML) document as a tree structure called *node-tree*. It contains all nodes (parents, children and siblings) of the HTML document it models (which is displayed in the **Web Browser** in the CWC Studio):

- root nodes;
- element nodes;
- text nodes;
- attribute nodes;
- namespace nodes;
- processing instruction nodes;

- comment nodes.

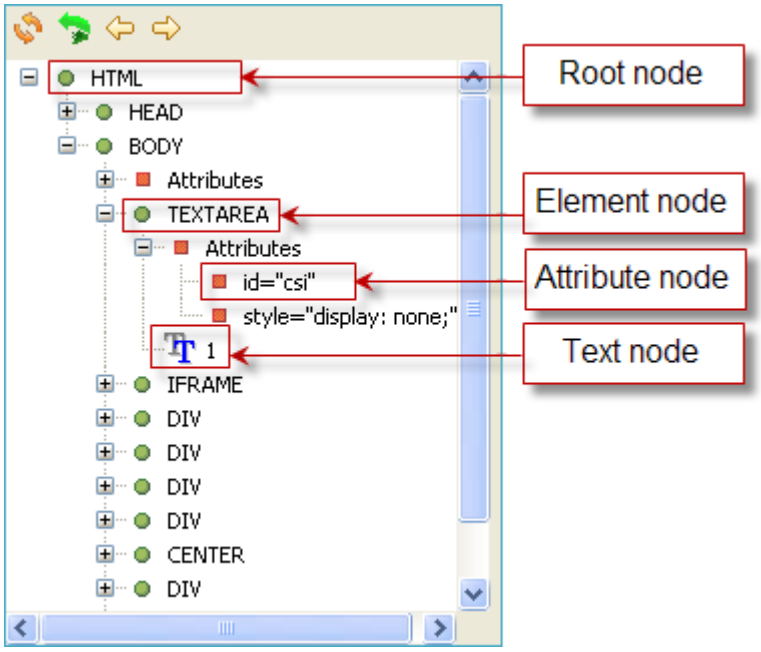


Figure 1 - 27: DOM Tree

The table below describes the **Dom Tree** icons:

Table 1 - 10: Dom Tree icons

Icon	Description
	*SyncTree* icon. Synchronizes the DOM Tree with the HTML page currently displayed in the **Web Browser**.
	*Parent node* icon. Selects and highlights in green the parent node of the node currently selected in the **DOM Tree**. Also outlines in green the corresponding HTML element in the **Web Browser**.
	*Previous node* icon. Selects and highlights in green the node appearing before the node currently selected in the **DOM Tree**. Also outlines in green the corresponding HTML element in the **Web Browser**.
	*Next node* icon. Selects and highlights in green the node appearing after the node currently selected in the **DOM Tree**. Also outlines in green the corresponding HTML element in the **Web Browser**.

Web Browser

The **Web Browser** displays the Web page accessed by CWC. Its logical structure is displayed in the DOM Tree on the left of the browser.

The **Web Browser** automatically displays the HTML page of the HTTP server defined as **Target Server** in the connector parameters (see *"Creating a Project and Setting Connector Parameters"* on page 2-6).

All HTML elements (image, text, field, etc.) of Web pages displayed in the **Web Browser** can be selected with a right-click. Once selected, they appear outlined in green in the **Web Browser**, and their corresponding element node is highlighted in green in the **DOM Tree**:

Figure 1 - 28: HTML element selected in the Web Browser

The XPath expression of selected elements can be generated either by right-clicking the required element node in the **DOM Tree** and selecting **Generate selection XPath (Enter)** or by selecting it with a left-click and pressing **Enter**. The XPath expression then appears in the **XPath Evaluator**.

The **Web Browser** includes a toolbar with:

- standard Browser icons - **Back**, **Forward**, **Refresh**, **Stop**;
- an address bar;
- icons to manage Web page tabs - **New**, **Close, Next, Previous**. The **Allow alert box** icon allows to display or block javascript pop-ups (blocked by default in CWC).

XPath Evaluator

The purpose of the **XPath Evaluator** is to display the XPath expression of HTML elements previously selected (with a right-click) in the **Web Browser**. The **Document** field displays the results from running the XPath expression specified in the **xPath** field on the **DOM Tree**.

The **XPath Evaluator** also contains icons for creating new CWC objects (statements, criteria, screen classes, etc.) from the XPath currently displayed in the **xPath** field.

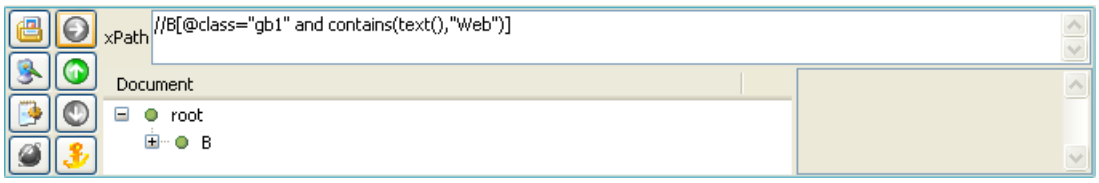


Figure 1 - 29: XPath Evaluator

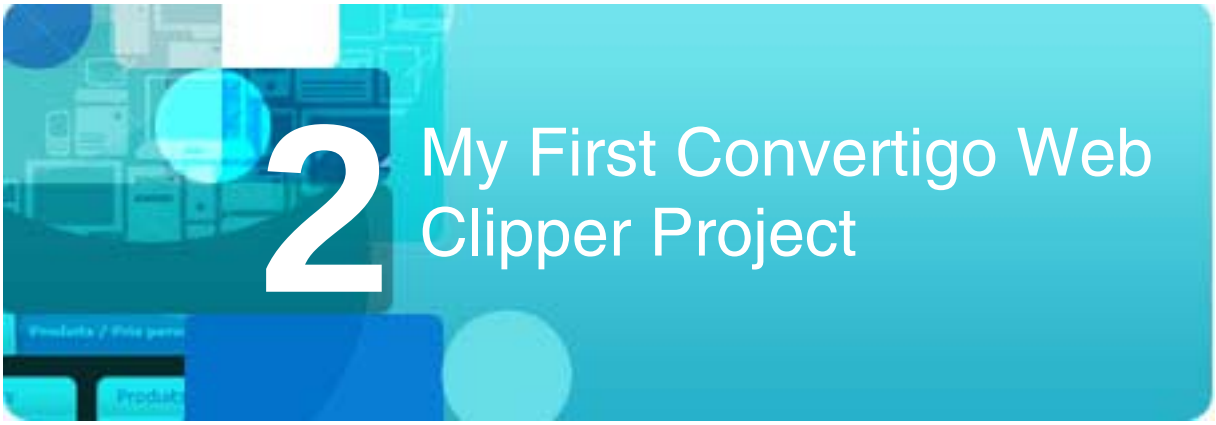
The table below describes the fields and icons of the **XPath Evaluator**:

Table 1 - 11: XPath Evaluator elements

XPath Evaluator Elements		Description
Field	xPath	Displays the XPath expression generated from the **Dom Tree**. XPath expressions can also be typed in directly and evaluated on the **DOM Tree**

Table 1 - 11: XPath Evaluator elements (...)

XPath Evaluator Elements		Description
	Document	Displays the results from running the XPath expression specified in the *xPath* field on the **DOM Tree**.
Icon		*Create new child screen class from current XPath* icon. Launches the **New Screen Class** wizard.
		*Create new criterion from current XPath* icon. Launches the **New Criterion** wizard.
		*Create new extraction rule from current XPath* icon. Launches the **New Extraction Rule** wizard.
		*Create new statement from current XPath* icon. Launches the **New Statement** wizard. Available when a suitable handler or container is selected in the **Projects View** only.
		*Evaluate XPath* icon. Evaluates the XPath expression specified in the *xPath* field on the **DOM Tree**. The result is displayed in the *Document* field.
		*Backward XPath History* icon. Displays in the *xPath* field the previous XPath stored in the XPath history.
		*Forward XPath History* icon. Displays in the *xPath* field the next XPath stored in the XPath history.
		*Set Anchor* icon. Applicable after an XPath has been generated for a given node. Fixes this XPath and highlights it in yellow. Used to develop complex XPath from an *anchored* XPath.



2 My First Convertigo Web Clipper Project

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

2.1 Web Clipping Project

This section describes the sample Web Clipping project and how to open the completed sample project from the Studio:

2.1.1 Project Description

The purpose of this Web clipping project (called `sample_doc_CWC`) is to:

- *Connect* to the Google *search* page (at `www.google.com`) and *clip* the Google ***logo*** and ***search form*** (search field, buttons and associated links) from it.
- *Launch* a search from the cliplet resulting from this first clipping.
- *Get* results from the Google servers and *clip*, on Google *result* pages, ***results*** as well as the Google ***navigation bar***.
- *Navigate* through results using the cliplet and open accessed result Web pages in new tabs in your browser.

The process can be basically summed up as follows:

1 Connect to the Google HTTP server at `www.google.com`:

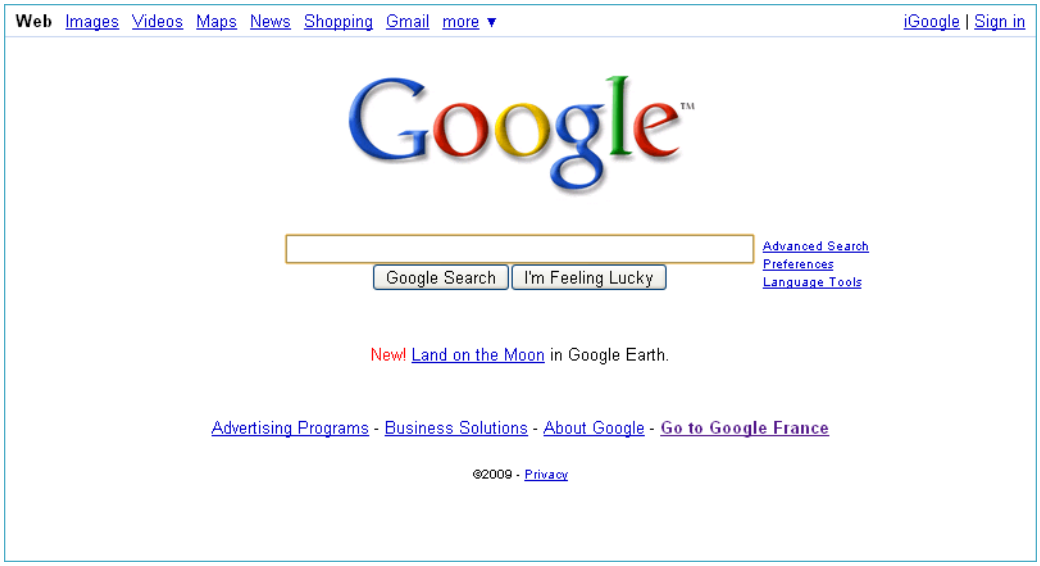


Figure 2 - 1: Connection to Google Web site

When accessing the `www.google.com` Website, Google servers provide Internet users all around the world with a "localized" version of the Google search engine. This adds an extra step to the project, since CWC must then be redirected to the standard Google home page (see "Case of a Country-Specific Google Search Page - Definition of the `GoogleSearchPageFR` Screen Class" on page 2-25).

2 Clip the Google logo:

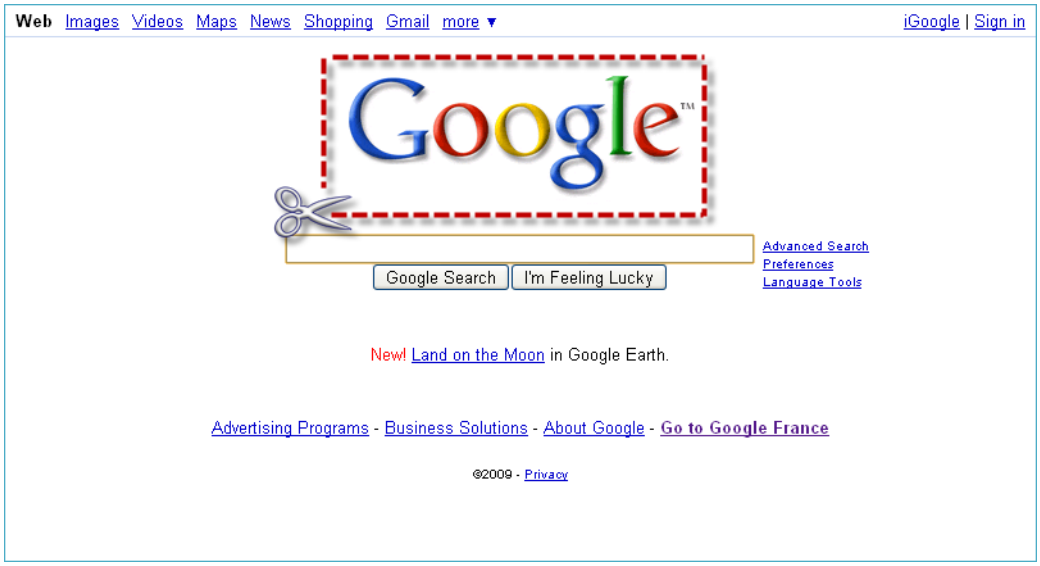


Figure 2 - 2: Clipping the Google logo

3 Clip the Google search form:

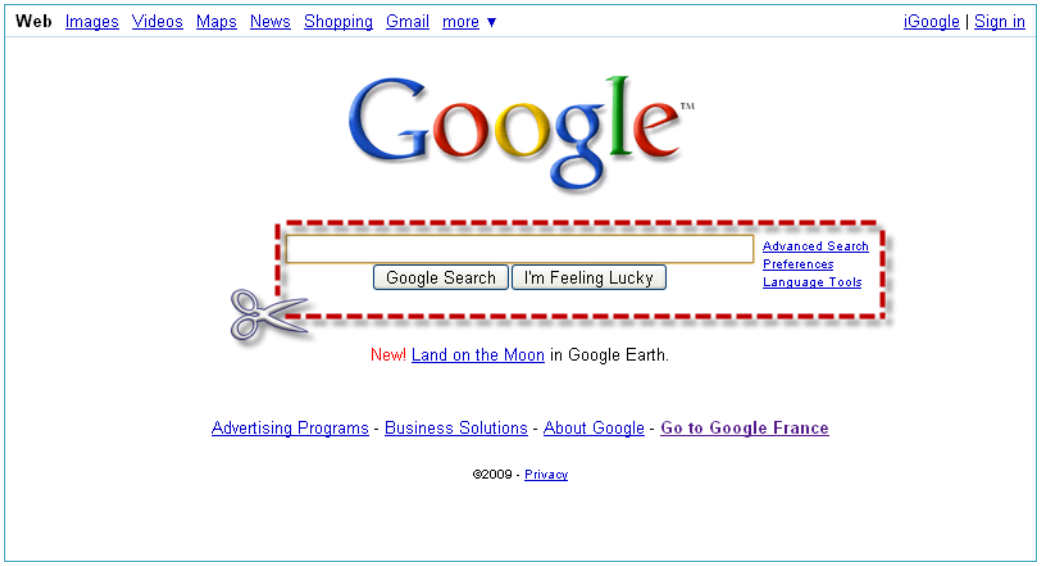


Figure 2 - 3: Clipping the Google search form (field and associated buttons and links)

4 Generate a cliplet based on these elements and have it displayed in the browser:

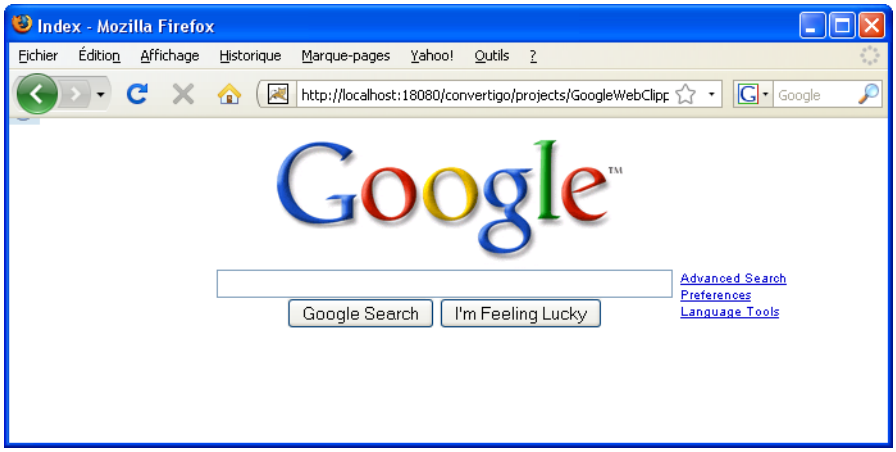


Figure 2 - 4: Cliplet generated based on Google logo and serach field clipping

5 Manually launch a search from the cliplet.

6 Clip results as well as the Google navigation bar:

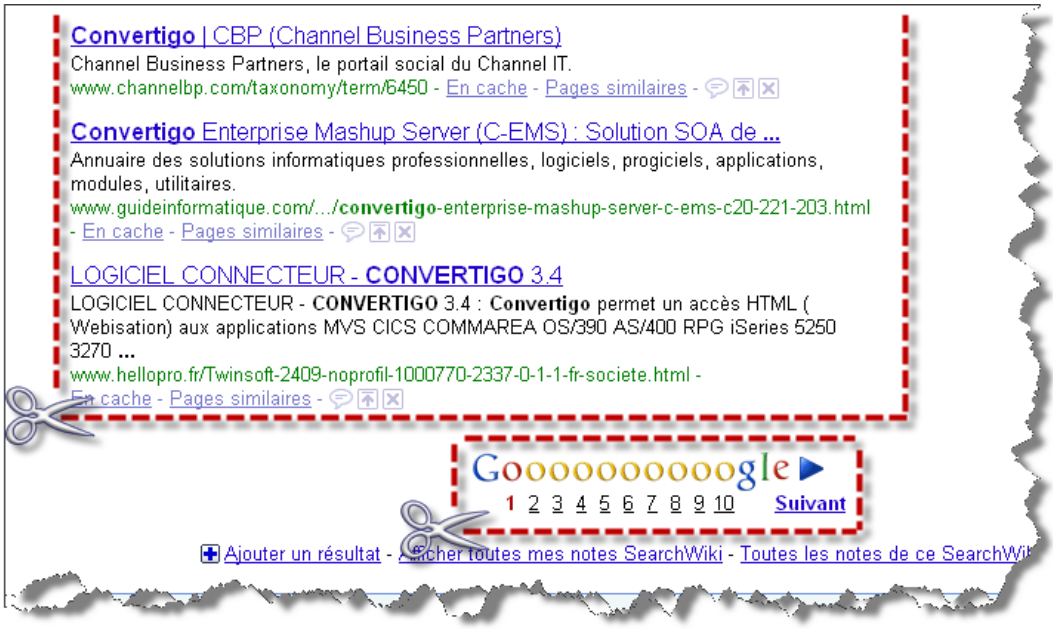


Figure 2 - 5: Clipping results and the Google navigation bar

7 Display clipped elements in browser:

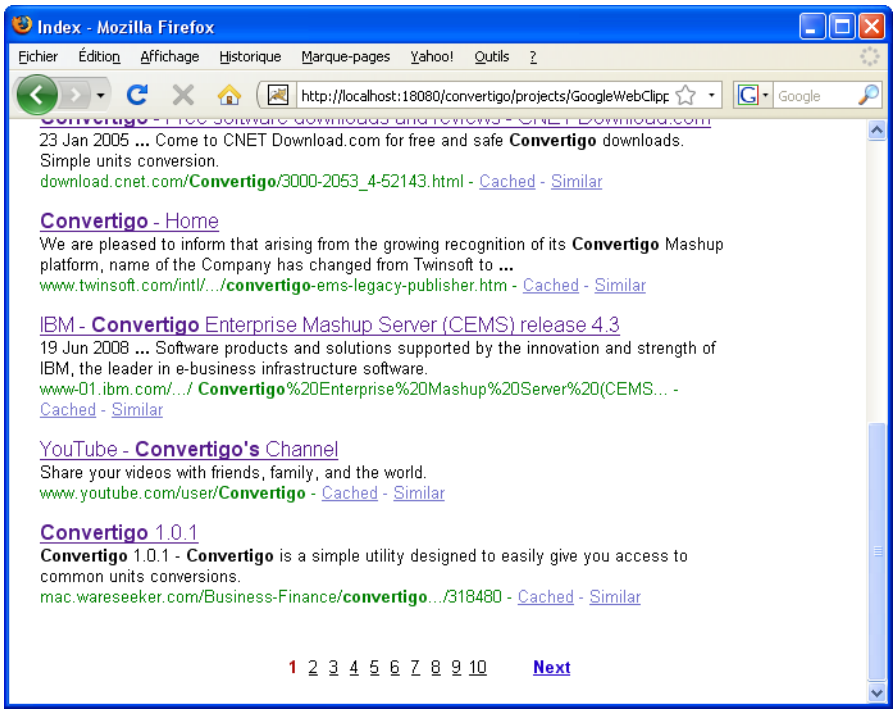


Figure 2 - 6: Clipped results and navigation bar

The cliplet creates a *light* version of the Google search engine, with only well-targeted information (results and navigation bar; no ads, promotional links, etc.). The user can navigate through results as if in a standard Google page and present data as required using specific style sheets (see *"Editing an XSL Style Sheet"* on page 3-12).

2.1.2 Opening the Sample Project from the Studio

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

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

To open the sample project from the Studio

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

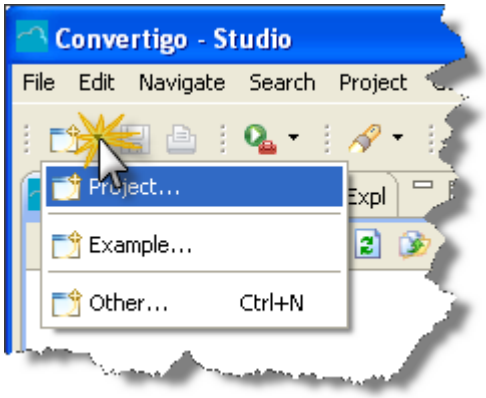


Figure 2 - 7: Launching the New Project wizard

A **New Project** wizard opens.

2 Expand **Convertigo Projects**, then **Documentation Samples**, and select **Web Clipping**:

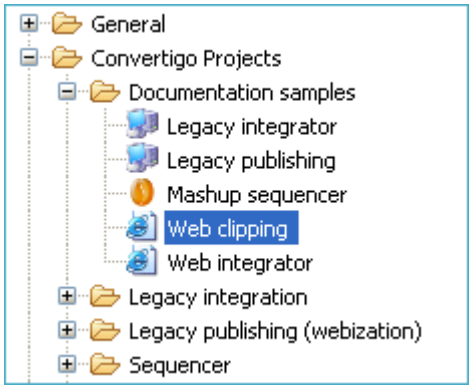


Figure 2 - 8: Selecting the CWC sample project

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

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

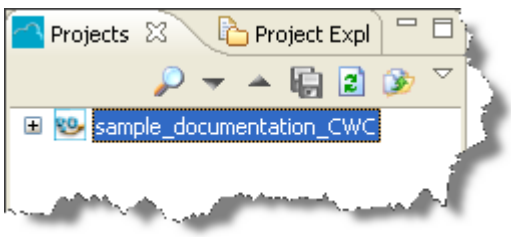


Figure 2 - 9: Sample project in Projects View

You can now start setting up your own CWC project, `sample_doc_CWC`.

2.2 Setting Up the Project

The following sections explain step by step how to set up your first Web clipping project:

- Creating a Project and Setting Connector Parameters
- Defining Screen Classes
- Defining a Transaction
- Executing a Transaction

2.2.1 Creating a Project and Setting Connector Parameters

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

In this example:

- the project is called `sample_doc_CWC`;
- the connector is an HTML connector called `GoogleConnector` accessing the `www.google.com` Web site.

To create a project and set a connector

1 Launch the **Convertigo Studio**.

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

A **New Project** wizard opens:

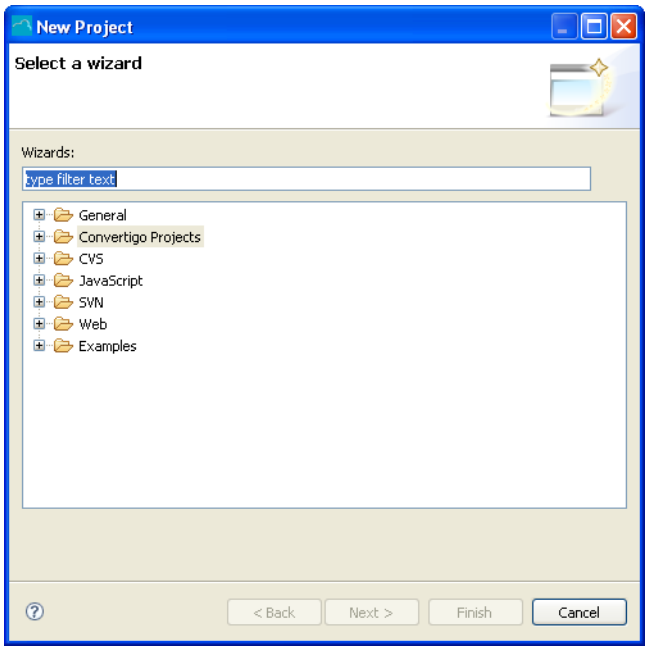


Figure 2 - 10: New Project Wizard

3 Expand **Convertigo Projects**, then **Web Clipping** and select **Web Clipping**:

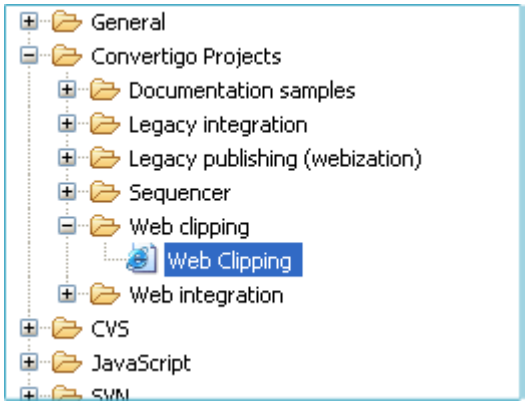


Figure 2 - 11: Selecting a project type

4 Click on **Next**:

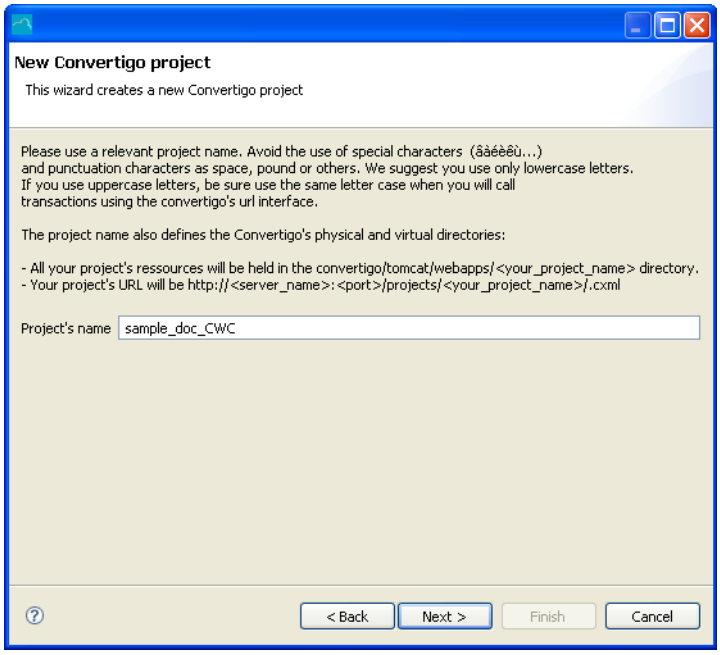


Figure 2 - 12: Entering a project name

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

6 Click on **Next**:

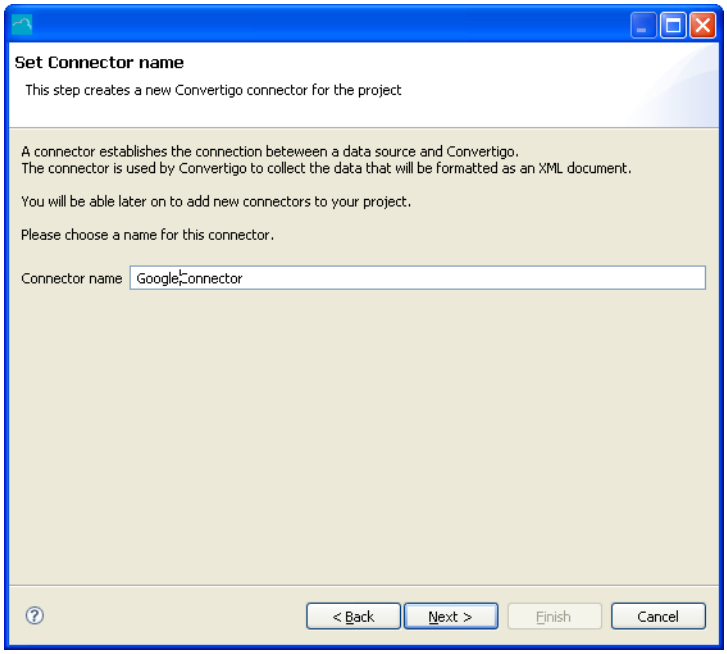


Figure 2 - 13: Entering a connector name

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

7 Click on **Next**:

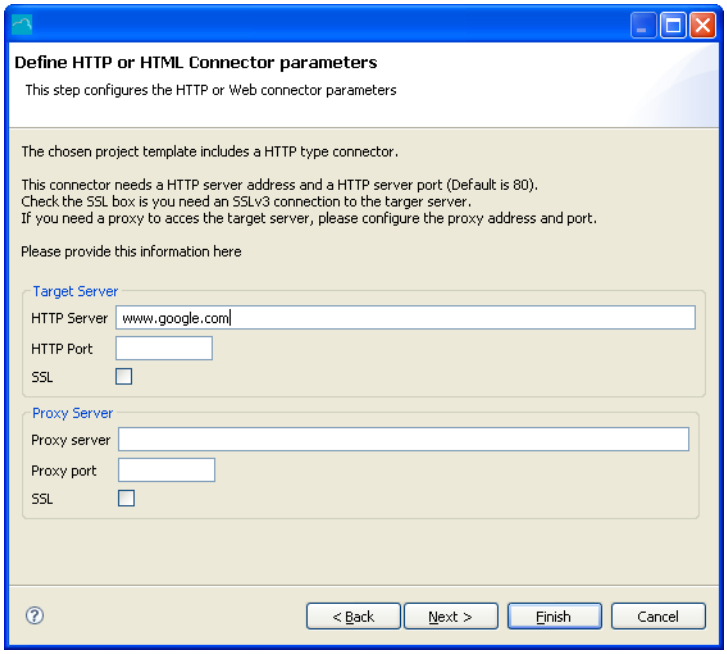


Figure 2 - 14: Setting HTTP or HTML connector parameters

8 Enter the required information in the **Target Server** section:

- in the **HTTP server** field, type in the target Web site URL address (for example `www.google.com`);
- in the **HTTP Port** field, type in the HTTP port if required (left blank in this sample project);

- check the **SSL** checkbox to connect to a secured HTTP server (HTTPS server).

9 If you're using a proxy server to access the Web, enter the required information in the **Proxy Server** section (**Proxy Server** address, **Proxy Port** and **SSL**).

10 Click on **Next**.

The **New project summary** window opens:

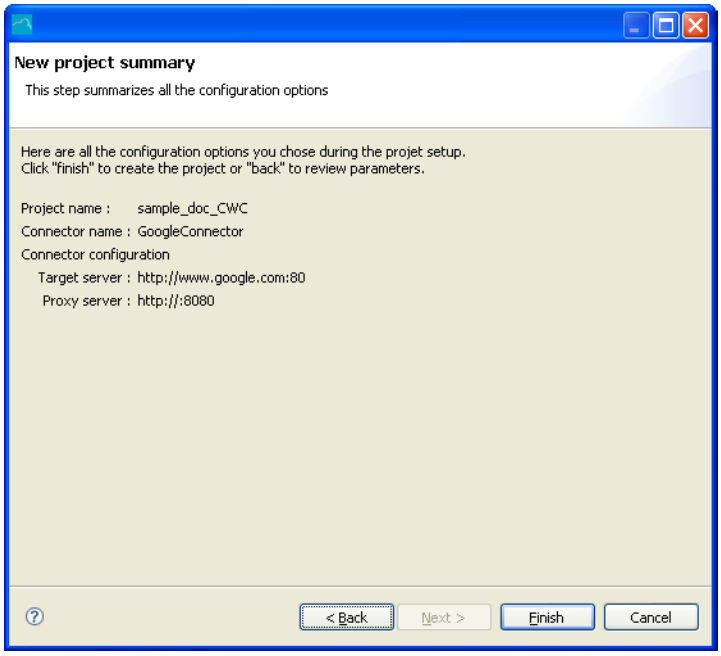


Figure 2 - 15: New project summary window

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

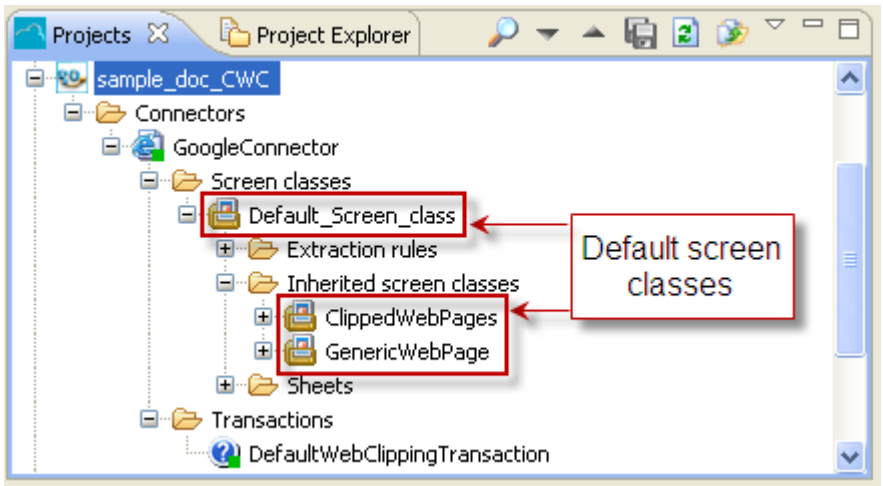


Figure 2 - 16: sample_doc_CWC project appearing in the Projects View

Three default screen classes and one default transaction have been created in their respective folders (*Screen Classes* and *Transactions* - see *"Screen Class"* on page 1-5 for more information on default screen classes).

The HTML page of the HTTP server specified as **Target Server** opens in the **Web Browser** of the **Connector View**, with its **DOM Tree** to the left:

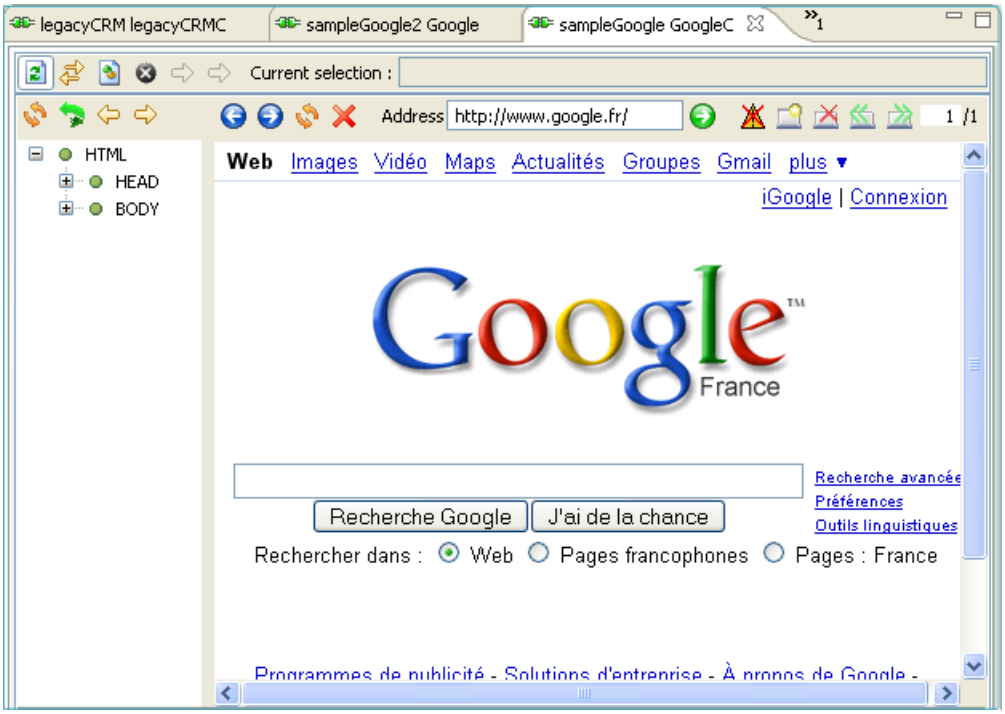


Figure 2 - 17: DOM Tree and Web Browser

This is the starting point of the project.

WHAT COMES NEXT?

The following steps consist in defining a set of screen classes and in setting their *detection criteria* (in other words, HTML elements - text, links, logos, etc. - defining a Web screen in a unique manner).

When CWC accesses a page, all criteria are checked; if all criteria defined for a screen class match page elements, then CWC associates the page to this screen class. The *extraction rules* as well as *statements* associated with the detected screen class are then executed and required parts of the HTML document are clipped.

2.2.2 Defining Screen Classes

This section shows how to create and define screen classes.

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

1 *Selecting* the element on the screen (right-click).

2 *Finding* its most accurate node(s) in the DOM Tree.

3 *Generating* its XPath.

4 *Creating* the corresponding screen class from the generated XPath expression.

DEFINED SCREEN CLASSES

In our sample project, two *main* screen classes are created:

Table 2 - 1: Defined screen classes

Name	Description	Detection criteria
`GoogleSearchPage`	Correspond to Google search pages.	Google logo + plain Web link
`GoogleResultsPage`	Corresponds to the Google results page.	Google result + plain Web link

These screen classes are children screen classes of the `ClippedWebPages` screen class (see *"Screen Class"* on page 1-5).

For projects set in countries for which a country-specific Google homepage exists, a third screen class must be defined (named `GoogleSearchPageFR` in our project, see "Case of a Country-Specific Google Search Page - Definition of the `GoogleSearchPageFR` Screen Class" on page 2-25). This screen class is detected using an hypertext link (either "Google.com in English" or "Go to Google.com" depending on the country) of country-specific Google homepages. This link is used to redirect CWC back to the standard Google.com search page.

The following figures show the HTML elements chosen as detection criteria for defining the `GoogleSearchPage` and `GoogleResultsPage` screen classes:

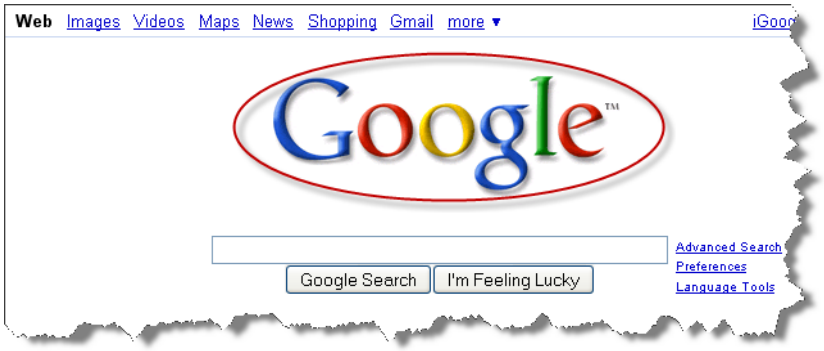


Figure 2 - 18: Google logo (GoogleSearchPage screen class)

Figure 2 - 19: Plain "Web" link (GoogleSearchPage and GoogleResultsPage screen classes)

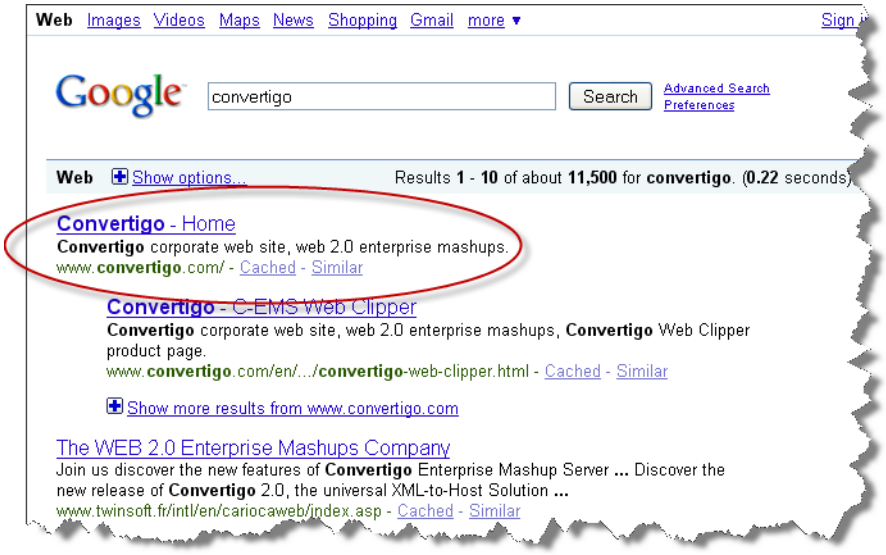


Figure 2 - 20: Google result (GoogleResultsPage screen class)

Definition of the `GoogleSearchPage` Screen Class

The first screen class to be created is `GoogleSearchPage`.

Why is the `GoogleSearchPage` screen class needed?

This screen class corresponds to the first page accessed by CWC (using the **HTTP server** parameter set for the connector when creating the project): the Google search page. It must be defined for CWC to be able to clip the Google logo and search form according to extraction rules created later on in the project see *"Creating Extraction Rules"* on page 2-28).

What are the `GoogleSearchPage` screen class detection criteria?

The `GoogleSearchPage` screen class is detected using two detection criteria:

- the Google homepage logo;
- the "Web" plain link in the upper left corner of the Google search page.

What are the basic steps for creating the `GoogleSearchPage` screen class?

The basic steps for creating this screen class are the following:

1 In the Google search page, *right-click* the Google logo.

2 *Select* an appropriate element node in the **DOM Tree**.

3 *Generate* the XPath corresponding to the previously selected element.

4 *Create* the new screen class from generated XPath.

5 *Right-click* the "Web" text item.

6 *Generate* the Xpath corresponding to the previously selected element.

7 *Add* a second criterion to the screen class previously generated.

To create the GoogleSearchPage screen class

1 Select `ClippedWebPages` with a left-click in the **Projects View** (`GoogleSearchPage` is a child screen class of `ClippedWebPages`):

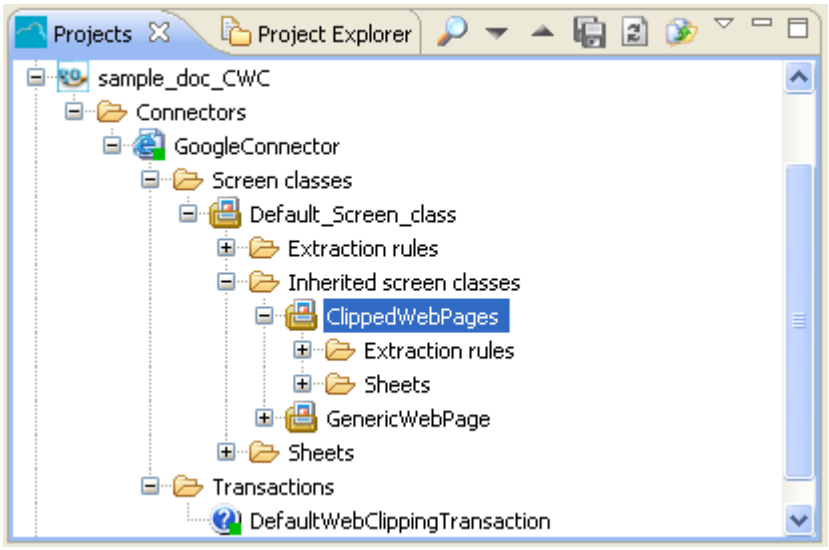


Figure 2 - 21: Selection of the parent screen class

2 Right-click on the Google logo in the Google search page displayed in the **Web Browser**.

The element is outlined in green in the **Web Browser** and its element node is highlighted in green in the **DOM Tree**.

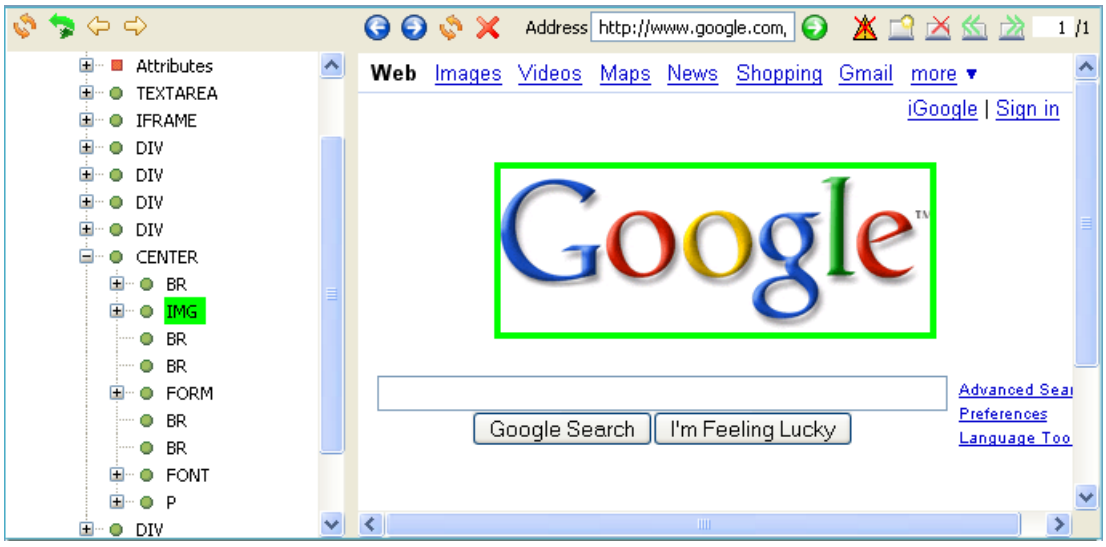


Figure 2 - 22: Selection of a detection criterion for the googleSearchPage screen class

This detection criterion is selected because the Google search page does include a Google logo, whereas results pages do not. Therefore, it will be a good differentiator between the `GoogleSearchPage` and `GoogleResultsPage` screen classes.

3 Expand the **DOM Tree** to select the most relevant element node for this detection criterion:

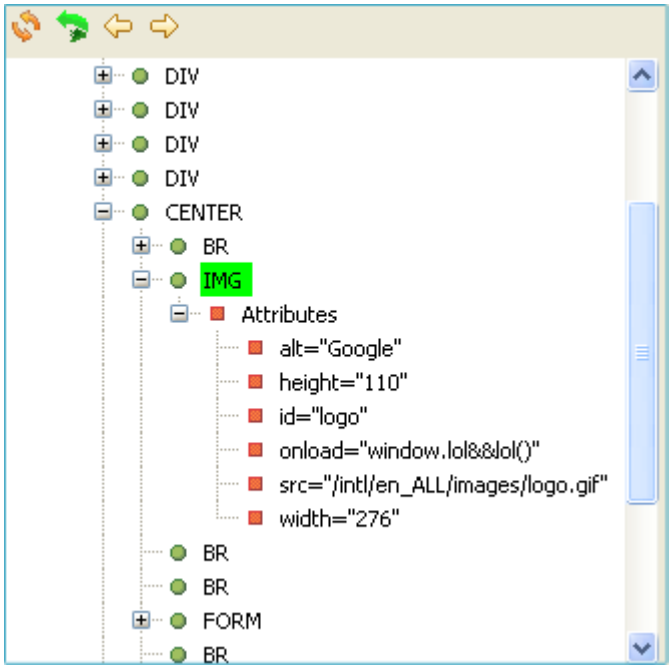


Figure 2 - 23: Attribute nodes of the HTML element selected as detection criterion for the GoogleSearchPage screen class

An `id` attribute node appears in the tree. This attribute always define an HTML element in a unique manner, so this is the best candidate for generating the XPath corresponding to the `IMG` element.

When present, the `NAME` attribute is also a good candidate. On the contrary, avoid using attribute nodes such as SRC and HREF, which are never good candidates for generating XPath corresponding to detection criteria (their value often changes).

4 Right-click the `id="logo"` attribute node and select **Generate selection XPath (Enter)** or select the attribute node with a left-click and press **Enter**.

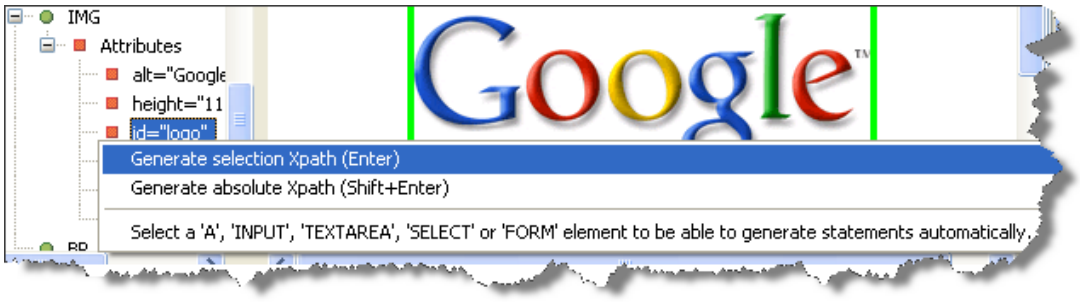


Figure 2 - 24: Selection of nodes and generation of XPath

*You can also right-click the parent `IMG` element and select **Generate selection XPath (Enter)** or select the `IMG` element with a left-click and press **Enter**: the XPath will intelligently be generated with the `id` attribute, since it exists.*

The XPath expression of the selected node appears in the **XPath Evaluator**.

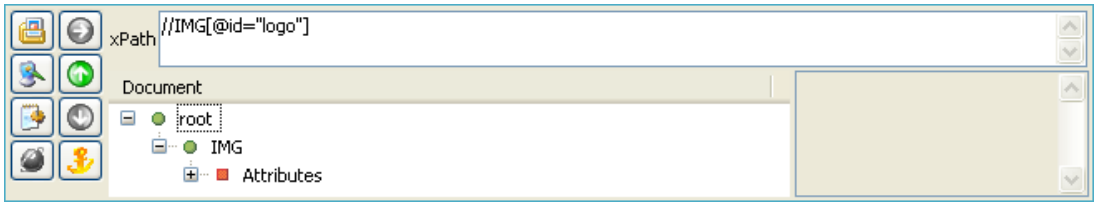


Figure 2 - 25: XPath expression of the selected node (corresponding to Google logo)

5 In the **XPath Evaluator**, click on the **Create new child screen class from current XPath** icon  . The **New Screen Class** wizard opens:

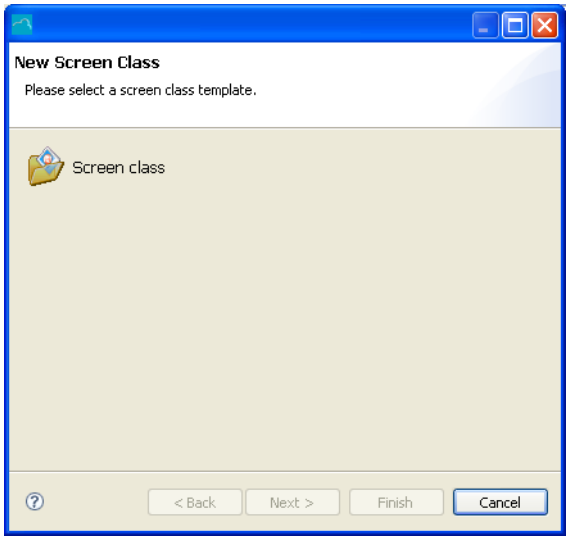


Figure 2 - 26: New Screen Class wizard

6 Click on Screen class.

7 Click on **Next**:

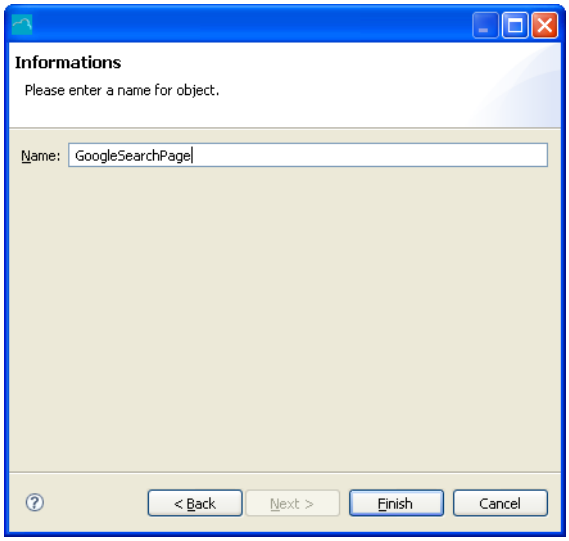


Figure 2 - 27: Giving a name to the new screen class

8 Type in the name of the new screen class (for example `GoogleSearchPage`).

9 Click on **Finish.**

The screen class now appears in the **Projects View** as an *inherited* screen class of the `ClippedWebPages` screen class:

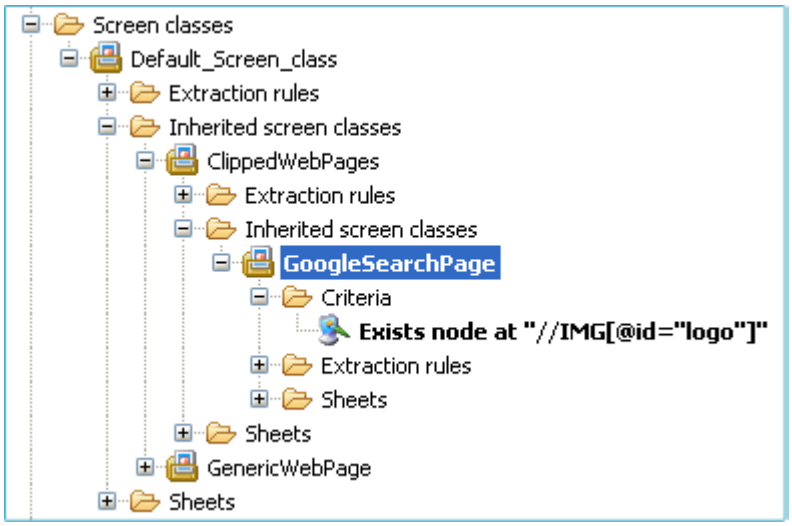


Figure 2 - 28: GoogleSearchPage screen class and defined criterion

The criterion previously defined for the screen class is automatically generated and appears in the related **Criteria** folder.

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

Remember to save your project on a regular basis.

The next step consists in adding a second detection criteria to the screen class. This second criterion is the "Web" link displayed in the upper left corner of Google Web pages.

A second detection criterion is added to the screen class definition to minimize the risk of "misdetection" when accessing a Google page with a Google logo, which could be an Image or Video Google page, and not a "Web" one.

11 In the **Web Browser**, right-click on the word "Web" in the upper left corner of the Google search page.

The element is outlined in green in the **Web Browser** and its element node is highlighted in green in the **DOM Tree**:

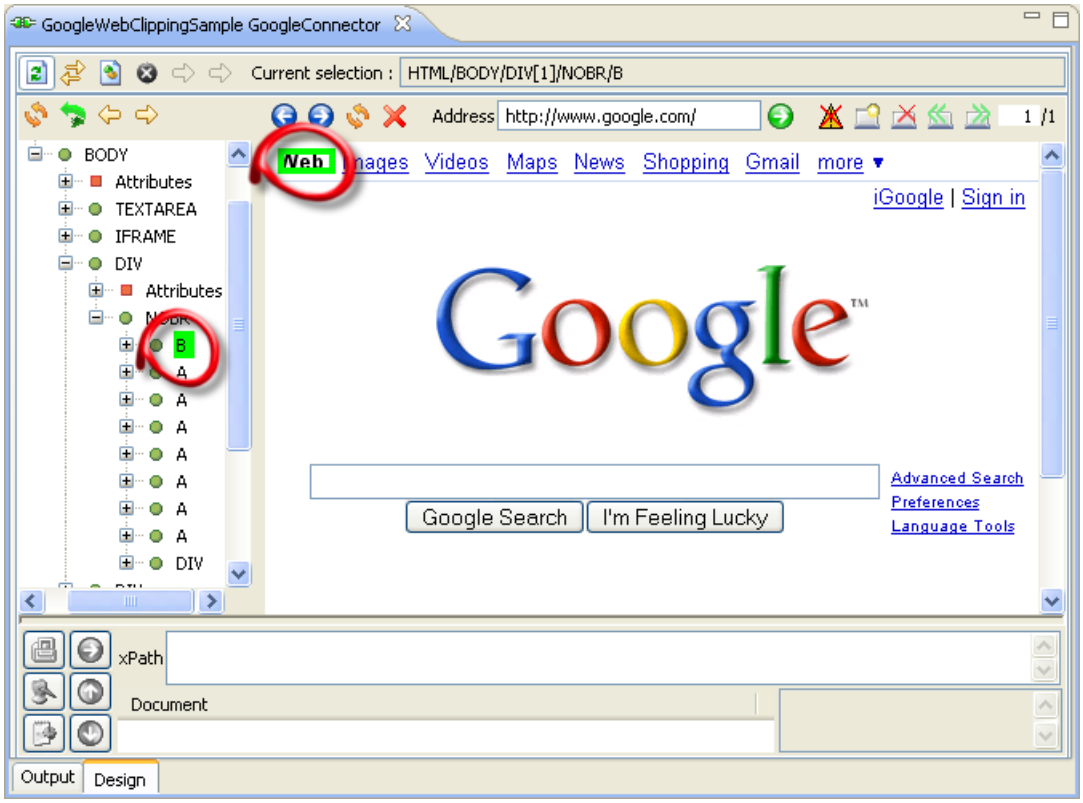


Figure 2 - 29: Second detection criterion of the GoogleSearchPage screen class

12 Expand the `B` element in the **DOM Tree**:

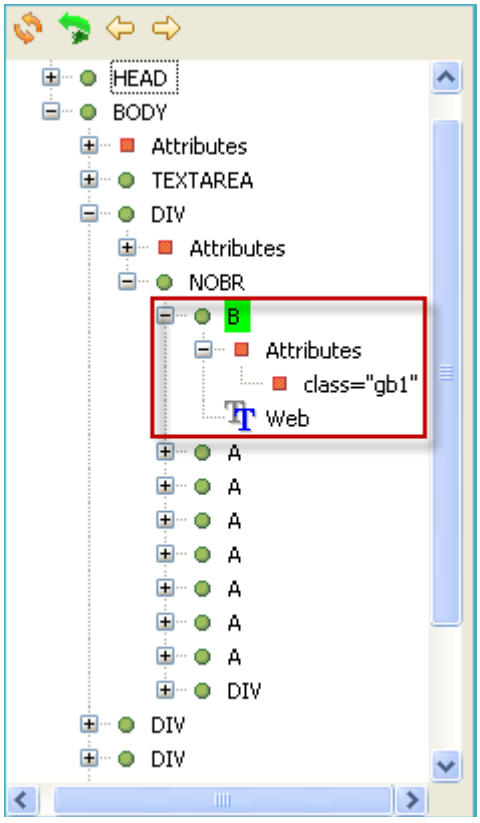


Figure 2 - 30: Expanding an element to find relevant attribute nodes in the DOM Tree

Two nodes are associated with the `B` element: an attribute node (`class="gb1"`), which

represents the element class, and a text node (`Web`), which represents the element content.

We will select both and generate the XPath from the selection:

13 While keeping the `Shift` button pressed, select `class="gb1"` and `Web` in the **DOM Tree** with a left-click.

14 Press **Enter** or right-click on the selection and select **Generate selection XPath (Enter)**.

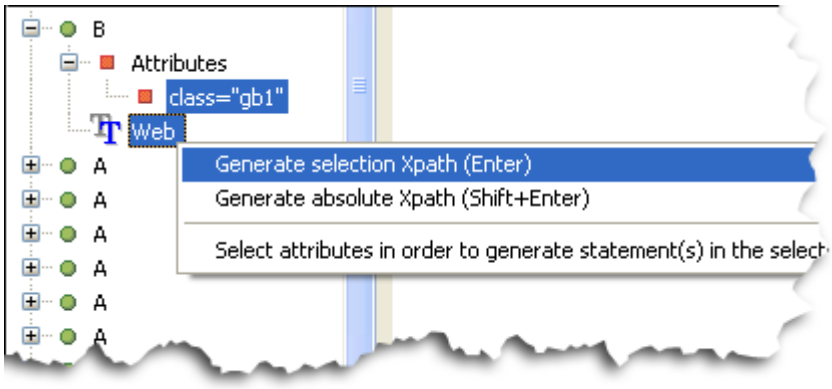


Figure 2 - 31: Generating XPath from class and text content

An XPath expression corresponding to the selected HTML element (i.e. "Web" item) and based on the two nodes selected as detection criteria appears in the **XPath Evaluator**:

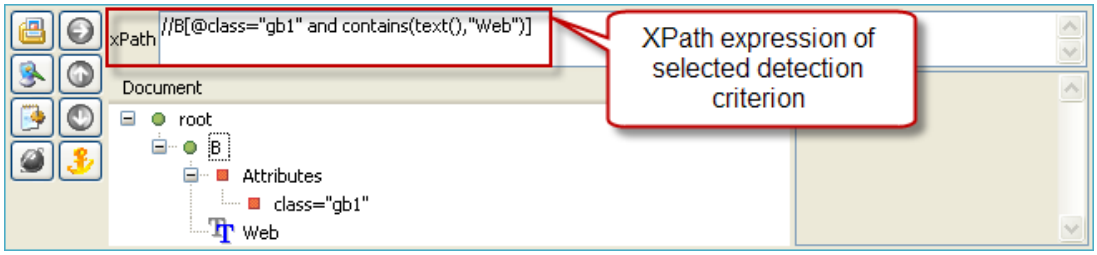


Figure 2 - 32: XPath expression of second selected criterion of the GoogleSearchPage screen class

The results from running the XPath expression on the DOM appear in the **Document** window.

We will now add this XPath as extra criterion to the corresponding screen class, `GoogleSearchPage`.

15 Select `GoogleSearchPage` with a left-click in the **Projects View**.

16 In the **XPath Evaluator**, click on the **Create new criterion from current XPath** icon  .

The **New Criteria** wizard opens:

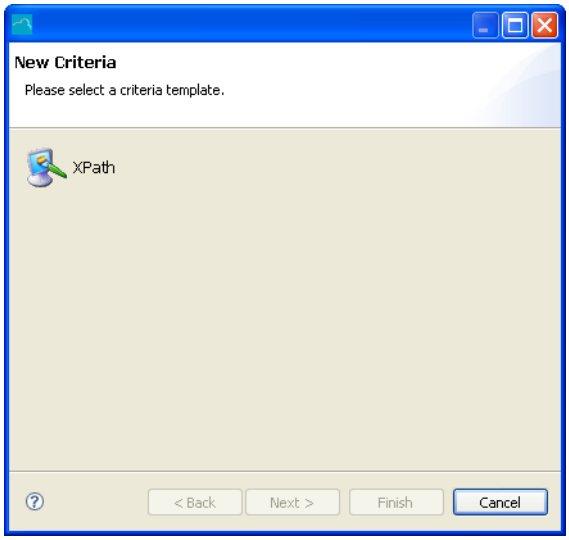


Figure 2 - 33: New Criteria wizard

17 Click on XPath, then on **Next**.

18 Type in the name of the new criterion (for example `XPath_Web`) in the **Name** field.

19 Click on **Finish.**

The new criterion appears in the **Criteria** folder of the `GoogleSearchPage` screen class:

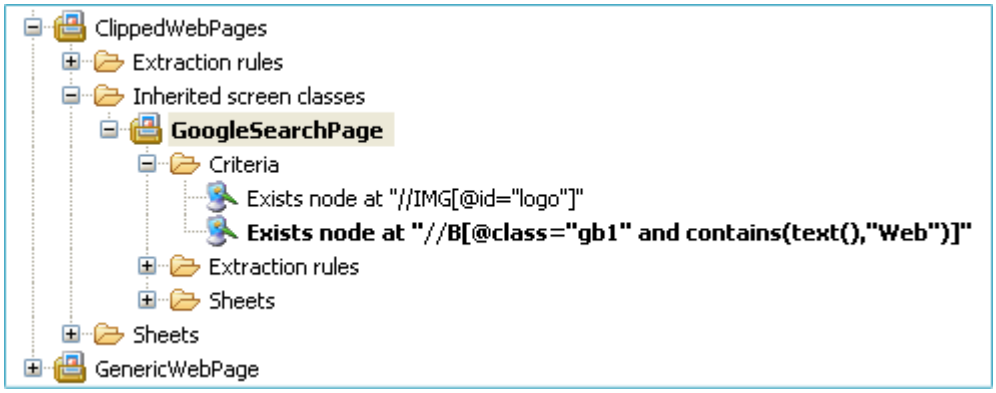


Figure 2 - 34: New criterion in the Criteria folder of the screen class

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

Remember to save your project on a regular basis.

To check if detection criteria are correct for this screen class

Assuming that the **Web Browser** currently displays a Google search page, click on the **Show current screen class** icon  in the **Connector View** toolbar.

Expected result: In the **Projects View**, the `GoogleSearchPage` screen class is automatically highlighted, showing that detection criteria identifying this screen class do all match on the accessed Web page.

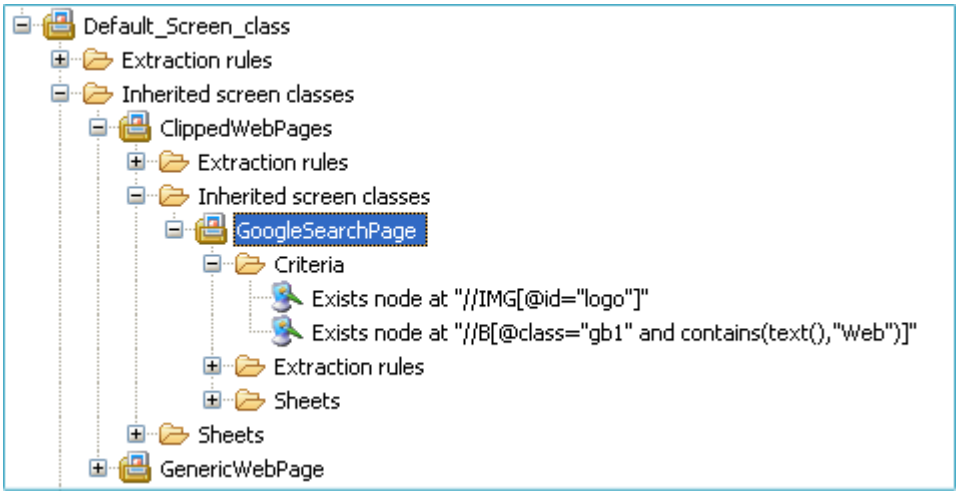


Figure 2 - 35: Highlighted screen class after Go to current screen class object command

Definition of the `GoogleResultsPage` Screen Class

The following step consists in creating a second screen class corresponding to Google result pages: `GoogleResultsPage`.

Why is the `GoogleResultsPage` screen class needed?

Here are the first few steps of the clipping process:

1. Once the project transaction is launched, CWC connects to the `www.google.com` Web page (Google search page), which is detected as a `GoogleSearchPage` screen class.

> *For Internet users accessing the `www.google.com` Web site from a country for which a country-specific Google homepage exists, another step is needed here (redirection from country-specific page to Google.com page in English). For more information, see "Case of a Country-Specific Google Search Page - Definition of the `GoogleSearchPageFR` Screen Class" on page 2-25.*

2. The Google logo and search form are clipped and presented to the user in a cliplet.
3. The user launches a search thanks to the cliped form.
4. Google results are displayed in the updated cliplet.

Google result pages must then be detected for CWC to be able to clip results together with the navigation bar. This is the reason why a `GoogleResultsPage` screen class is created.

What are the `GoogleResultsPage` screen class detection criteria?

The `GoogleResultsPage` is detected using two detection criteria:

- The presence of Google result items in the page.
- The "Web" plain link in the upper left corner of the Google search page.

What are the basic steps for creating the `GoogleResultsPage` screen class?

The basic steps for creating the `GoogleResultsPage` screen class are the following:

1 *Send* a keyword-based Google search request from the **Web Browser**.

2 Once results are displayed, right-click *close to* a Google result item.

3 *Select* an appropriate element node in the **DOM Tree**.

4 *Generate* the XPath corresponding to the previously selected element.

5 *Create* the new screen class from generated XPath.

6 *Add* the second criterion by *copying-pasting* the `XPath_Web` criterion.

To create the GoogleResultsPage screen class

1 In the Google search page displayed in the **Web Browser**, type in a search keyword (for example `convertigo cliplet`) in the search field.

2 Launch the Google search by clicking on the **Google Search** button.

3 Once results are displayed, right-click to the right of the first result:

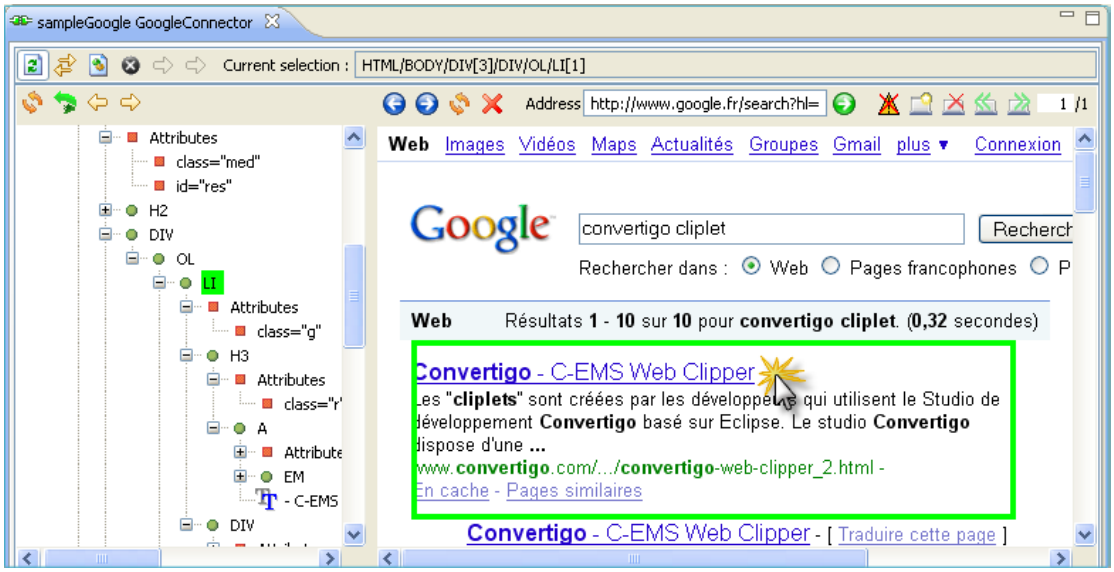


Figure 2 - 36: Selection of a detection criterion for the GoogleResultsPage screen class

Searching for a detection criteria is an empirical process that requires trying different elements on the page to select the best suited for detecting a screen class.

The `LI` container is highlighted in the **DOM Tree**. This container contains all nodes modeling the first result returned by the Google search engine. It does not include any node that could possibly serve as strong differentiator for the screen class. We must now look in parent nodes to find one.

4 To do so, click on  to select the parent node.

An `OL` container is outlined in green.

This element is useless as regards detection because it does not contain any attribute or text content that could serve to generate an accurate XPath for the selected HTML element.

5 Click on  .

A `DIV` container is outlined in green. For the same reasons as previously mentioned, this node cannot be used to generate an accurate XPath.

6 Click on  .

Another DIV container is highlighted in green:

Figure 2 - 37: Searching for a better detection criterion

This node contains an `id` attribute, which can be used as unique attribute for generating an element's XPath, as mentioned in the definition of the previous screen class.

7 Right-click the `id="res"` attribute node and select **Generate selection XPath (Enter)** or select the attribute node with a left-click and press **Enter**:

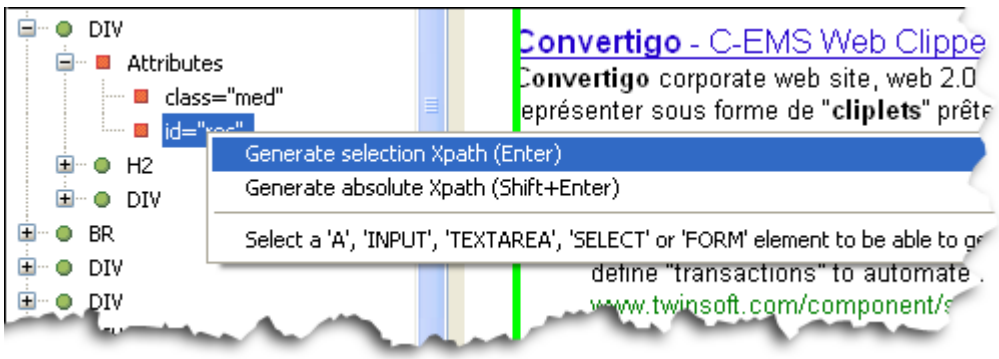


Figure 2 - 38: Generation of XPath for the GoogleResultsPage screen class

The XPath expression of the selected node appears in the **XPath Evaluator**.

*You can also right-click the parent `DIV` element and select **Generate selection XPath (Enter)** or select the `DIV` element with a left-click and press **Enter**: the XPath will intelligently be generated with the `id` attribute, since it exists.*

8 Select the parent screen class, `ClippedWebPages` in the **Projects View**.

9 In the **XPath Evaluator**, click on the **Create new child screen class from current XPath** icon  .

The **New Screen Class** opens.

10 Click on Screen class, then on **Next**.

11 Type in the name of the new screen class (for example `GoogleResultsPage`).

12 Click on **Finish.**

The screen class now appears in the **Projects View** as an *inherited* screen class of the `ClippedWebPages` screen class:

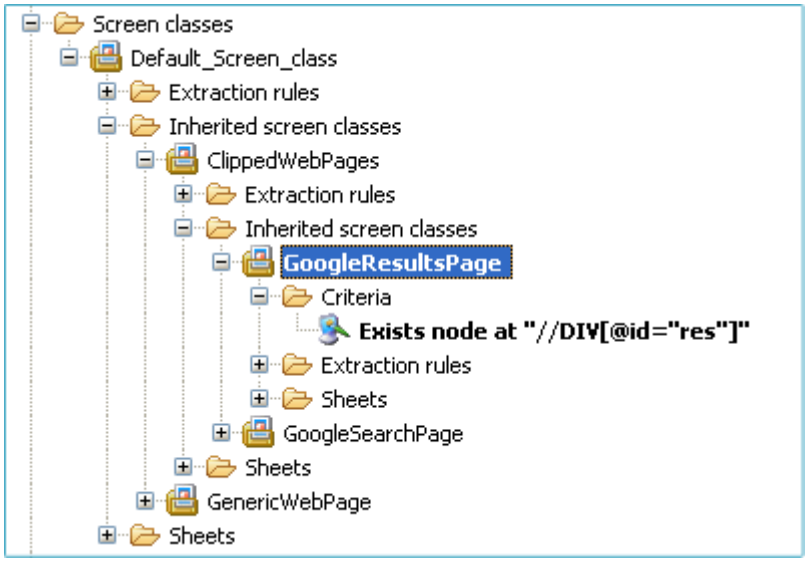


Figure 2 - 39: GoogleResultsPage screen class and defined criterion

The criterion defined for the screen class is automatically generated and appears in the related **Criteria** folder.

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

We now need to add to this screen class a second criterion: the `XPath_Web` criterion defined as second criterion for the `GoogleSearchPage` screen class. The criterion can simply be copied then pasted in the **Criteria** folder of the screen class.

A second detection criterion is added to the screen class definition to minimize the risk of "misdetection" when accessing a Google page with a Google logo, which could be an Image or Video Google page, and not a "Web" one.

To copy-paste a criterion from one screen class to another

1 In the **Projects View**, right-click on the `XPath_Web` criterion of the `GoogleSearchPage` screen class and select **Copy** in the contextual menu:

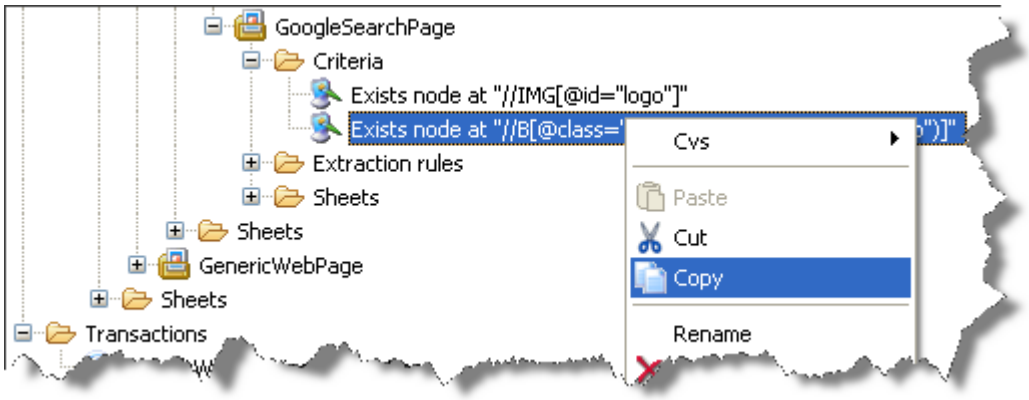


Figure 2 - 40: Copy criterion

2 Right-click on the `GoogleResultsPage` screen class then select **Paste** in the contextual menu:

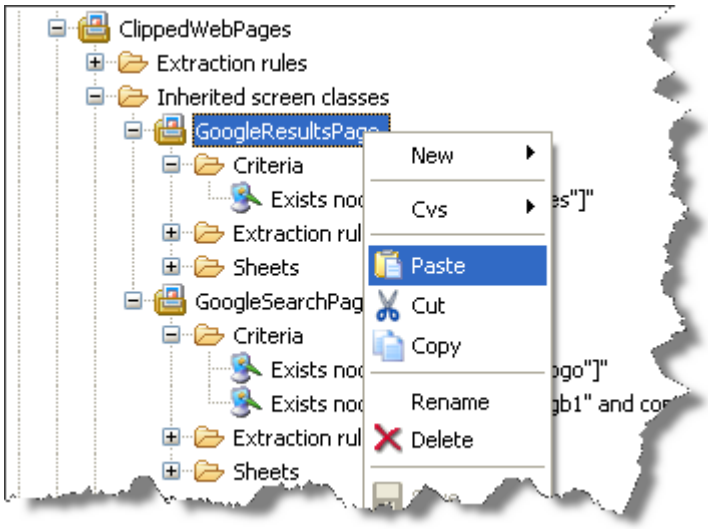


Figure 2 - 41: Paste criterion

The criterion is added to the **Criteria** folder:

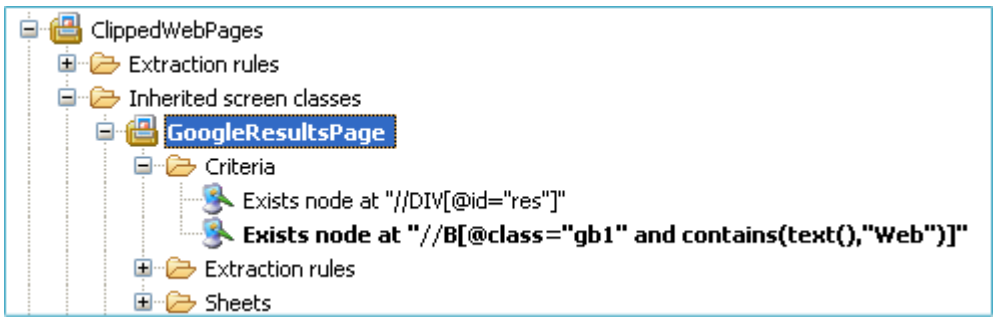


Figure 2 - 42: Pasted criterion in Criteria folder

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

The main screen classes are now created.

As mentioned before, Google servers automatically redirect Internet users to country-specific Google search pages, if a country-specific page has been created for their country. The following section addresses such a case.

Case of a Country-Specific Google Search Page - Definition of the `GoogleSearchPageFR` Screen Class

When CWC connects to the `www.google.com` Web site, if a Google country-specific page exists, it automatically opens in the browser.

Country-specific Google homepages can be defined as standard Google search pages with a *redirecting* hypertext link in the lower right corner of pages. This link can be:

- a **Google.com in English** link in non-anglophone country specific pages:

Figure 2 - 43: Example of country-specific non-anglophone Google search page (France)

- a **Go to Google.com** link in anglophone country-specific pages:

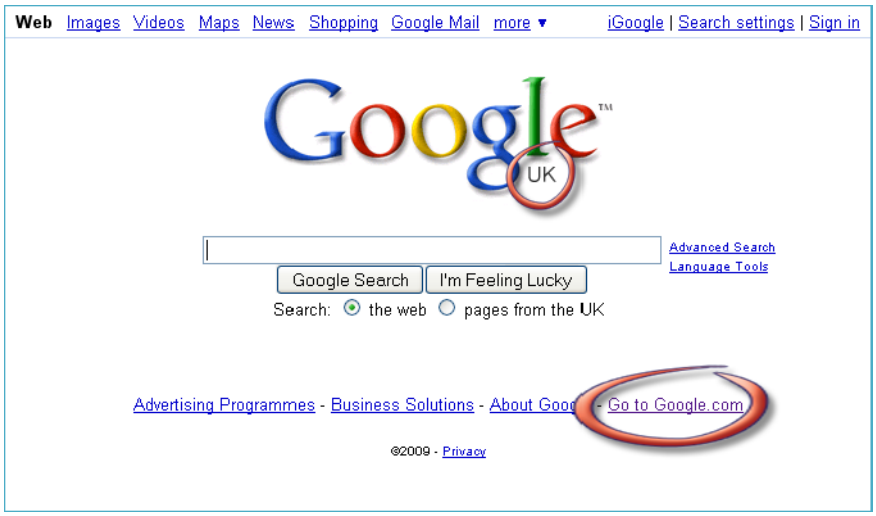


Figure 2 - 44: Example of country-specific anglophone Google search page (UK)

To detect country-specific pages, we will define a country-specific screen class (called

`GoogleSearchPageFR` in our project) having as detection criteria the redirecting link as well as criteria already defined for the `GoogleSearchPage` screen class.

The country-specific screen class can be defined as follows:

- it is a *child* screen class of the `GoogleSearchPage` screen class and, as such, inherits its criteria:
  - "Web" link in upper left corner of the Google page;
  - Google logo;
- it is characterized by the presence of a redirecting link (**Google.com in English** or **Go to Google.com**) in the lower right corner of the page.

To create a country-specific (GoogleSearchPageFR) screen class

1 In the country-specific Google search page displayed in the **Web Browser**, right-click on the redirecting link (**Google.com in English** or **Go to Google.com**).

The link is outlined in green in the **Web Browser** and its corresponding element is highlighted in green in the **DOM Tree**:

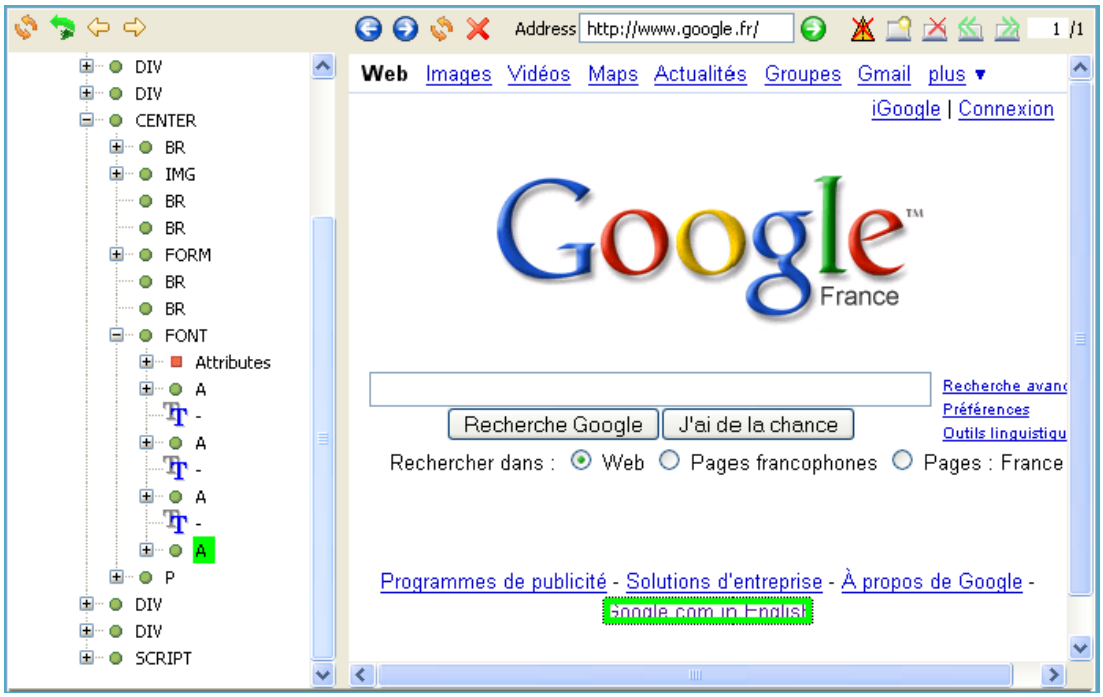


Figure 2 - 45: Selection of a detection criterion for the country-specific screen class

We will expand the A element node to see if it contains interesting attributes to generate the XPath corresponding to the link.

2 Expand the A element in the **DOM Tree** by clicking on ⊞:

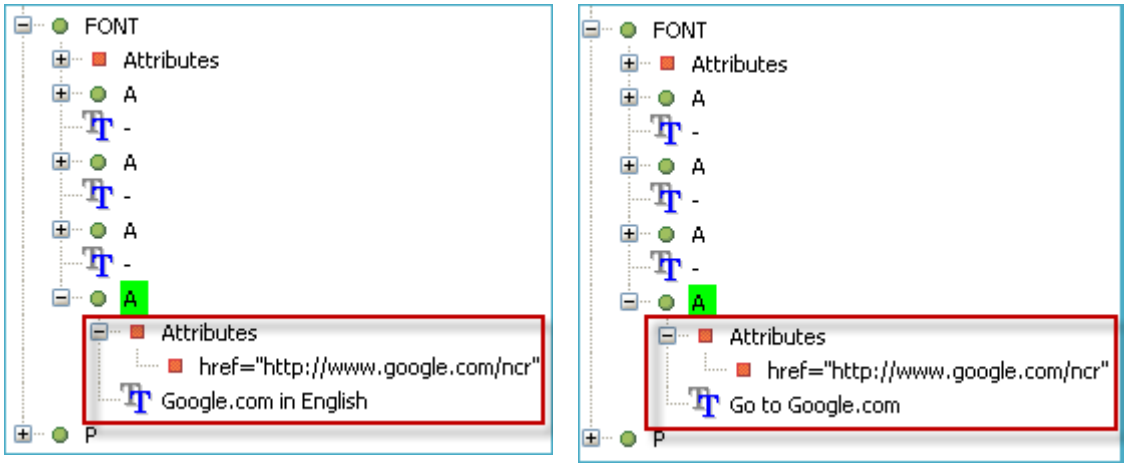


Figure 2 - 46: Expanding an element to find relevant attributes in the DOM Tree for the country-specific screen class

Two nodes are associated with the `A` element: an `href` attribute node, which contains the URL address of the hypertext link, and a text node (`Google.com in English` or `Go to Google.com`), which represents the element text content.

The `href` attribute is not interesting as regards XPath generation because its value may often change. We will select only the text content and generate the XPath from it.

3 Right-click the text node Google.com in English or Go to Google.com and select **Generate selection XPath (Enter)** or select it with a left-click and press **Enter**.

The XPath expression of the selected node appears in the **XPath Evaluator**:

Figure 2 - 47: XPath expression of country-specific redirecting link

4 Select the parent screen class, `GoogleSearchPage`.

5 In the **XPath Evaluator**, click on the **Create new child screen class from current XPath** icon .

The **New Screen Class** opens.

6 Click on Screen class, then on **Next**.

7 Type in the name of the new screen class (for example `GoogleSearchPageFR`).

8 Click on **Finish.**

The screen class now appears in the **Projects View** as an *inherited* screen class of the `GoogleSearchPage` screen class:

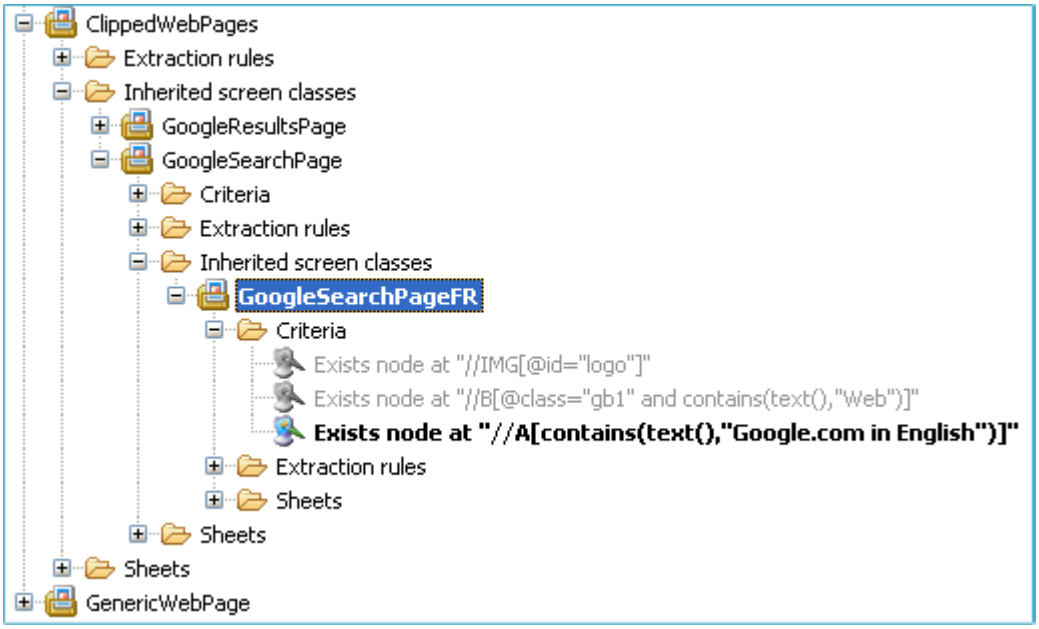


Figure 2 - 48: GoogleSearchPageFR screen class in Projects View

The criterion previously defined for the screen class is automatically generated and appears in the related **Criteria** folder. Criteria inherited from the parent screen class appear greyed.

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

WHAT COMES NEXT?

Now that all screen classes have been defined for the project, we will:

1 *Create* Web clipping *extractions rules* associated with the `GoogleSearchPage` and `GoogleResultsPage` screen classes.

The extraction rules are defined as *Web clipper* rules executed on detection of these screen classes. They "clip" HTML parts of Web pages that we want displayed to users in HTML format (*Google logo* and *search form* for Google search page, *Google results* and *navigation bar* for Google results page).

2 *Define* a *transaction*, that is to say an automated navigation that automatically accesses the Google search page and initiate the clipping process based on detection of screen classes and execution of extraction rules.

*The order in which extraction rules appear in the **Projects View** for a screen class is important, since HTML elements are clipped and presented in the generated cliplet following this order.*

2.2.3 Creating Extraction Rules

This section shows how to create *Web clipper* extraction rules associated with previously defined screen classes.

How can a Web clipper extraction rule be defined?

A *Web clipper* rule:

- *clips* from a Web page user-selected HTML elements (based on the XPath generated from this HTML element);
- *checks* that the header of the HTML page containing the clipped element is extracted as well (see Figure 2 - 96).

The following table describes the four *Web clipper* extraction rules needed in our sample project:

Table 2 - 2: Extraction rules

The extraction rule...	Is associated with the screen class...	And clips...
`WebClipImage`	`GoogleSearchPage`	the Google logo
`WebClipForm`		the Google search form
`WebClipResults`	`GoogleResultsPage`	the Google results list
`WebClipNavigationBar`		the Google navigation bar

What are the basic steps for creating a Web clipper extraction rule?

The logical process for creating a *Web clipper* extraction rule is the following:

1 *Select* in the **Web Browser** the element to be clipped.

2 *Select* in the **DOM Tree** the element node corresponding to this element.

3 *Generate* the corresponding Xpath.

4 *Launch* the **New Extraction Rule** wizard and choose the *Web Clipper* extraction rule template.

Web Clipper Extraction Rules associated with the `GoogleSearchPage` Screen Class

The following procedure shows how to create a *Web clipper* extraction rule. The first two extraction rules to be created are `WebClipForm` and `WebClipImage`.

Both extraction rules are executed on detection of the `GoogleSearchPage` screen class following their order in the **Extraction rules** folder of the screen class.

This screen class is detected as soon as CWC connects to the `www.google.com` address (**HTTP server** parameter set for the connector (see *"To create a project and set a connector" on page 2-6*)) and accesses the Google search page.

Extraction rules associated with this screen class are therefore the first to be executed in the process. They clip the Google logo and search form and generate the corresponding cliplet, from which the user can launch a search.

If you're accessing the Google Web site from a country having a country-specific Google homepage, the first screen class to be detected is `GoogleSearchPageFR` (see "Case of a Country-Specific Google Search Page - Definition of the `GoogleSearchPageFR` Screen Class" on page 2-25). No extraction rule is defined for the `GoogleSearchPageFR` screen class.

To create a Web Clipper extraction rule

1 Assuming that the **Web Browser** currently displays a Google search page, select the Google search field with a right-click in the **Web Browser**.

The search field is outlined in green in the **Web Browser** and the corresponding node is highlighted in green in the **DOM Tree**:

Figure 2 - 49: Selection of Google search field in Web Browser

2 Go up the **DOM Tree** by clicking on the **Parent node** icon until the `FORM` element node including the Google search field, links and buttons is highlighted:

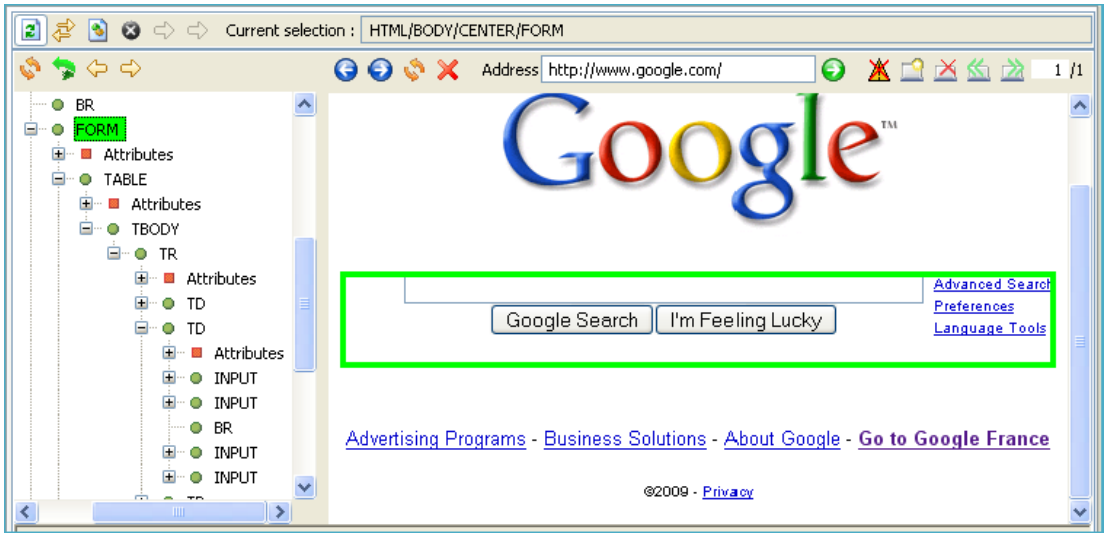


Figure 2 - 50: Going up the DOM Tree to find an appropriate node for the Google search form

Selecting the `FORM` element in the **DOM Tree** outlines the whole search form in the **Web Browser**.

3 Expand the **DOM Tree** to select the most relevant node for generating Xpath corresponding to this HTML element.

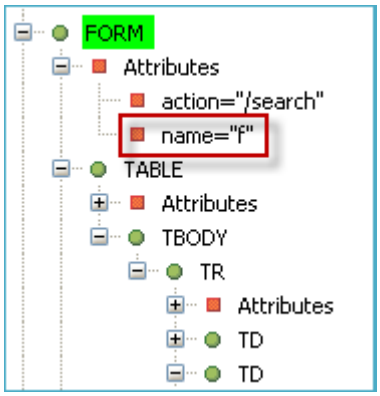


Figure 2 - 51: Expanding an element to find relevant attributes in the DOM Tree

A `name="f"` attribute node appears in the tree. As mentionned before, the `name` attribute is a good candidate for generating Xpaths.

4 Right-click on the `name="f"` attribute and select **Generate selection XPath (Enter)** or select the attribute with a left-click and press **Enter**:

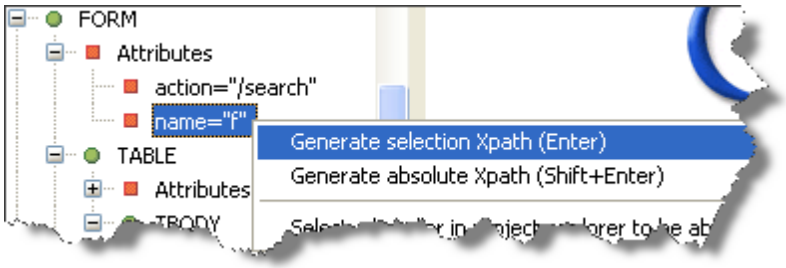


Figure 2 - 52: Generating XPath expression corresponding to Google search form

The XPath corresponding to the Google search form is displayed in the **xPath** field of the **XPath Evaluator**:

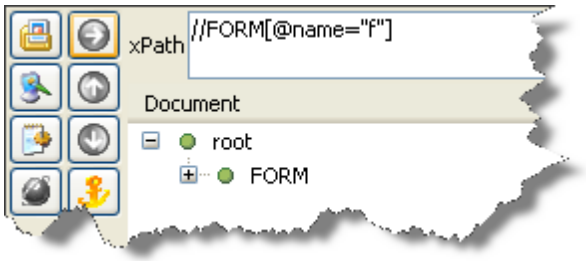


Figure 2 - 53: XPath expression on which the WebClipForm extraction rule will be based

*You can also right-click the parent `FORM` element and select **Generate selection XPath (Enter)** or select the `FORM` element with a left-click and press **Enter**: the XPath will intelligently be generated with the `name` attribute, since it exists.*

5 In the **Projects View**, select the `GoogleSearchPage` screen class, because the extraction rule applies to this screen class.

6 In the **XPath Evaluator**, click on the **Create new extraction rule from current XPath**

icon  in the **XPath Evaluator**.

A **New Extraction Rule** wizard opens:

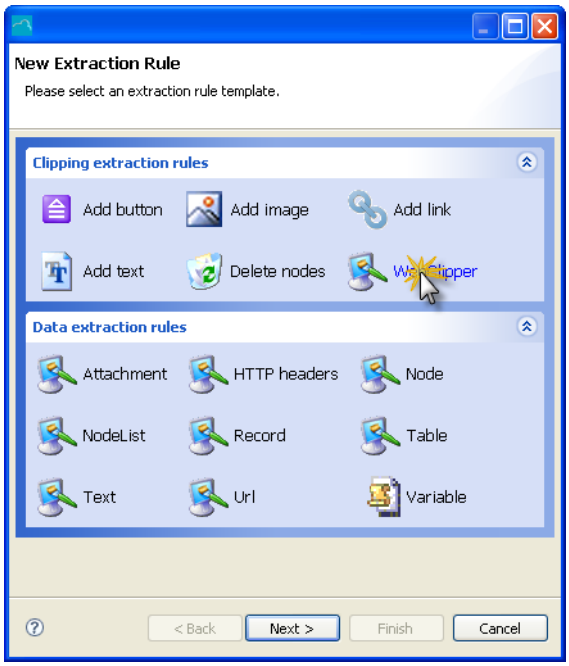


Figure 2 - 54: New Extraction Rule Wizard

7 Click on **WebClipper**.

8 Click on **Next**:

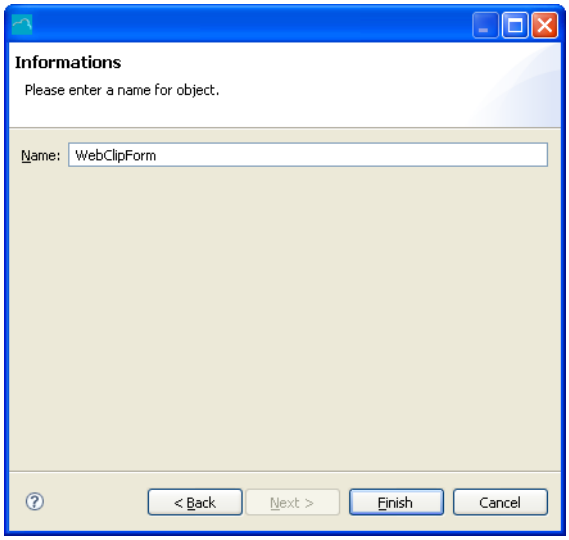


Figure 2 - 55: Giving a name to the new extraction rule

9 In the **Name** field, enter a name (for example `WebClipForm`).

10 Click on **Finish**.

The new extraction rule appears in the **Extraction rules** folder of the `GoogleSearchPage` screen class and is inherited in the **Extraction rules** folder of the `GoogleSearchPageFR` screen class.

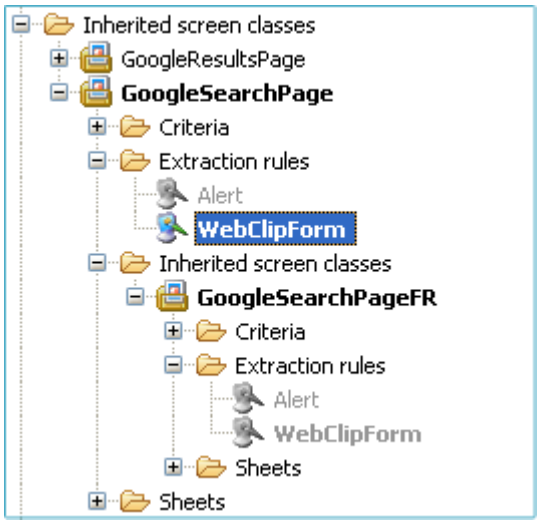


Figure 2 - 56: WebClipForm extraction rule in Extraction rules folders

Its properties are displayed in the **Properties View**:

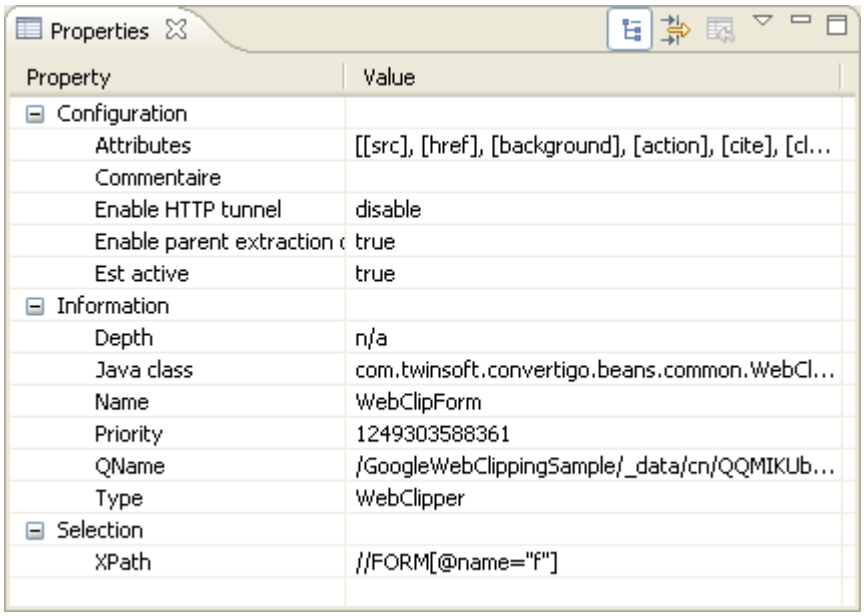


Figure 2 - 57: Extraction rule properties

Default parameters are correct and do not need any setting.

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

We can see that an additional extraction rule, `Alert`, appears greyed in the **Extraction Rules** folder of the screen class:

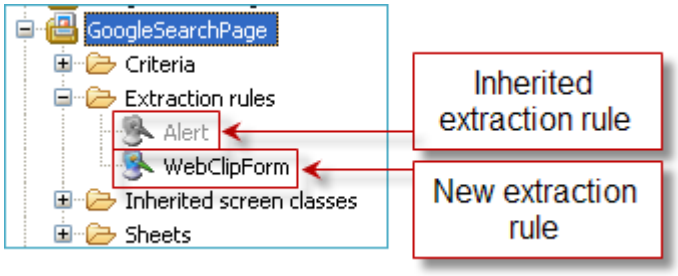


Figure 2 - 58: Inherited and new extraction rules

This extraction rule is inherited from its parent `Default_Screen_Class`. It is used to handle javascript pop-ups.

Now that the first extraction rule has been defined for the `GoogleSearchPage` screen class, we need to define the second extraction rule: `WebClipImage`. It is executed on detection of the `GoogleSearchPage` screen class to clip the Google logo.

12 Right-click on the Google logo in the Google search page displayed in the **Web Browser**:

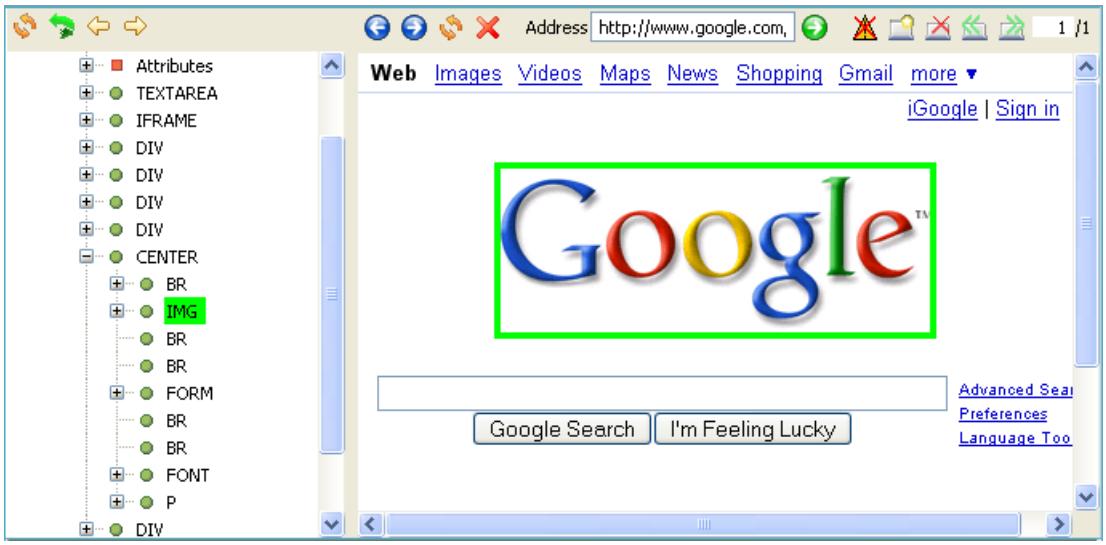


Figure 2 - 59: Selection of the Google logo to be clipped

13 Expand the **DOM Tree** to select the most relevant node for generating the XPath corresponding to this HTML element:

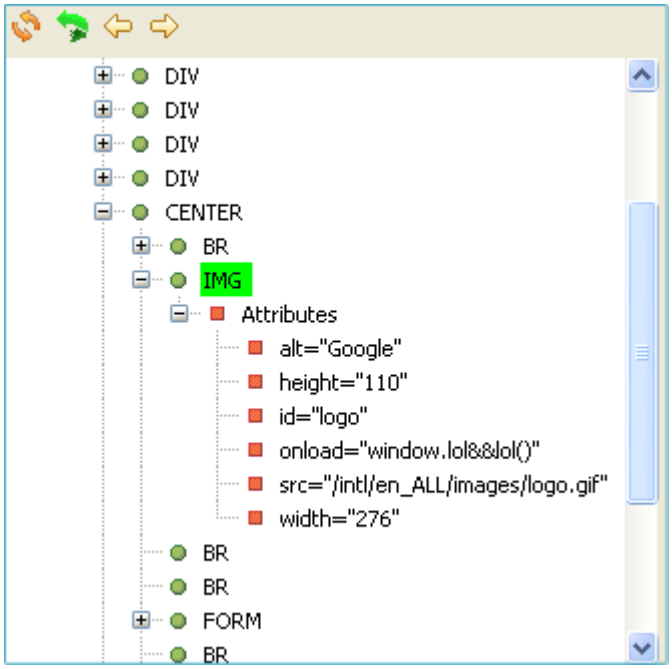


Figure 2 - 60: Attribute nodes of the HTML element to be clipped

An `id` attribute node appears in the tree. As mentioned when creating the `GoogleSearchPage` and `GoogleResultsPage` screen classes, this attribute always define an HTML element in a unique manner, so this is the best candidate for generating the XPath corresponding to the `IMG` element.

When present, the `NAME` attribute node is also a good candidate. On the contrary, avoid using attribute nodes such as `SRC` and `HREF`, which are never good candidates for generating XPath corresponding to detection criteria (their value is highly variable).

14 Right-click the `id="logo"` attribute node and select **Generate selection XPath (Enter)** or select it with a left-click and press **Enter**:

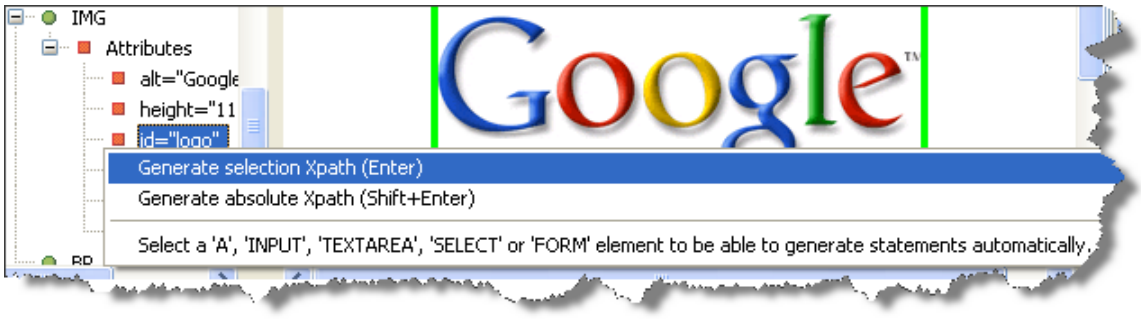


Figure 2 - 61: Generating XPath expression before creating the WebClipImage extraction rule

The XPath expression of the selected node appears in the **XPath Evaluator**.

*You can also right-click the parent `IMG` element and select **Generate selection XPath (Enter)** or select the `IMG` element with a left-click and press **Enter**: the XPath will intelligently be generated with the `id` attribute, since it exists.*

15 In the **Projects View**, click on the `GoogleSearchPage` screen class.

16 In the **XPath Evaluator**, click on the **Create new extraction rule from current XPath** icon  .

The **New Extraction Rule** wizard opens.

17 Click on **WebClipper**.

18 Click on **Next**.

19 In the **Name** field, enter a name (for example `WebClipImage`).

20 Click on **Finish**. The new extraction rule appears in the **Extraction rules** screen class folder:

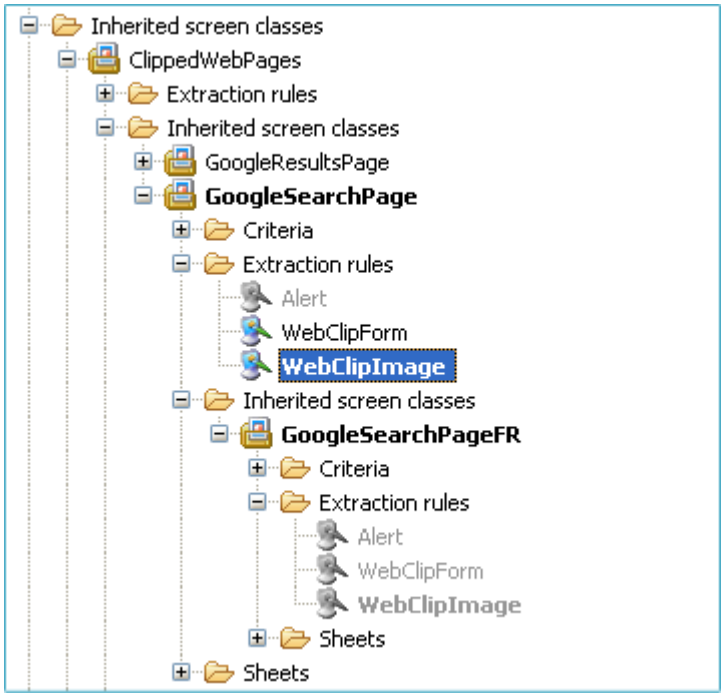


Figure 2 - 62: WebClipImage extraction rule in Extraction rules folders

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

As mentioned in the *Note* preceding Table *"Extraction rules"* on page 2-29, HTML elements are clipped and displayed in the final cliplet following their order in this folder.

We want the generated cliplet to display the Google logo *on top* of the search form, so we need to sort extraction rules in this order: first `WebClipImage` then `WebClipForm`.

22 Select `WebClipImage` extraction rule in the folder.

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

The `WebClipImage` object now appears above `WebClipForm` in the **Extraction rules** folder:

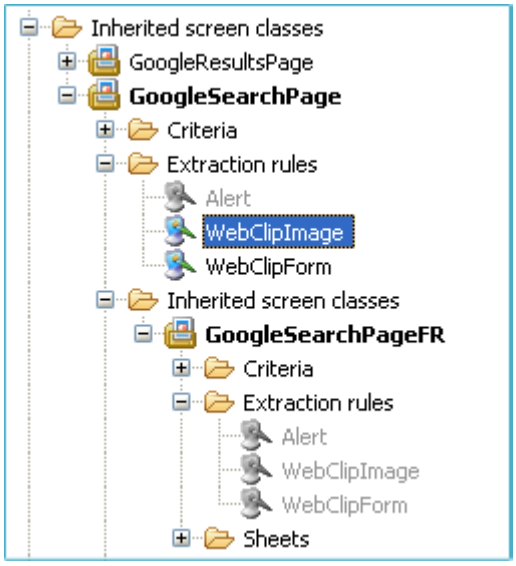


Figure 2 - 63: Increasing the WebClipImage priority

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

Web Clipper Extraction Rules associated with the `GoogleResultsPage` Screen Class

The following procedure describes how to create the Web clipper extraction rules associated with the `GoogleResultsPage` screen class. It follows the same logic as the previous procedure (for a description of this logical process, see *"Creating Extraction Rules"* on page 2-28, beginning of section).

The two extraction rules we are about to create are called `WebClipResults` and `WebClipNavigationBar`. Both are associated with the `GoogleResultsPage` screen class.

After a search has been launched, results are displayed in a Google result page. This page is detected by CWC as a `GoogleResultsPage` screen class and extraction rules associated with this screen class are executed - they clip Google results together with the navigation bar and generate the corresponding cliplet (a light version of a Google results page).

To create the GoogleResultsPage Web clipper extraction rules

1 Assuming that the **Web Browser** currently displays a Google search page, type in a keyword (`convertigo cliplet` for example) in the Google search field.

2 Click on the **Google Search** button.

The **Web Browser** displays results.

3 Right-click to the right of the first result.

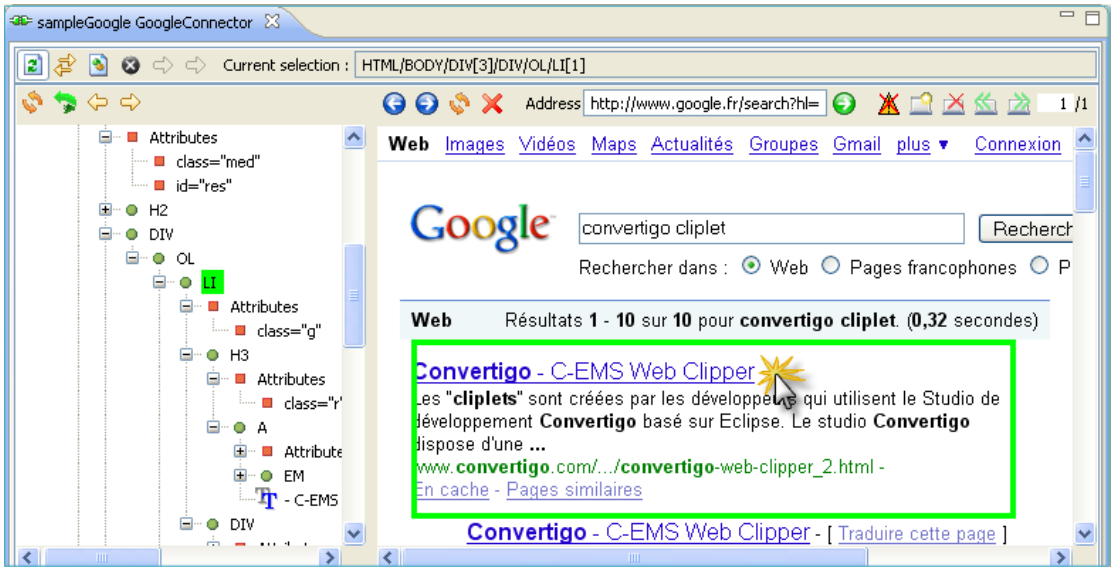


Figure 2 - 64: Selection of an XPath for Google result items

An `LI` element node is highlighted in green in the **DOM Tree**.

As mentioned before when defining the `GoogleResultsPage` screen class, this element does not contain any interesting attribute in terms of XPath generation. Moreover, we want to select here the container element that will be clipped by the extraction rule.

4 Go up the **DOM Tree** by clicking three times on the **Parent node** icon to find an appropriate element for Google results:

Figure 2 - 65: Going up the DOM Tree to select an appropriate node

The `DIV` element contains the whole results page. This is the HTML part that we want to clip.

5 Expand the **DOM Tree** to select the most relevant node for generating the Xpath corresponding to this element (see Figure 2 - 65).

An `id` node appears in the tree. As mentionned before, this attribute always defines an

HTML element in a unique manner, so it is the best candidate for generating the `DIV` element Xpath.

6 Right-click the attribute node `id="res"` and select **Generate selection XPath (Enter)** or select it with a left-click and press **Enter**:

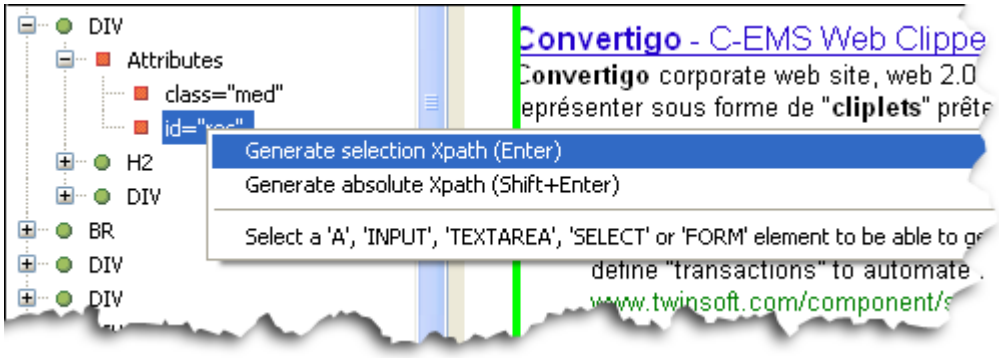


Figure 2 - 66: Generating XPath expression before creating the WebClipResults extraction rule

The XPath expression (`//DIV[@id="res"]`) of the selected node appears in the **XPath Evaluator**.

*You can also right-click the parent `DIV` element and select **Generate selection XPath (Enter)** or select the `DIV` element with a left-click and press **Enter**: the XPath will intelligently be generated with the `id` attribute, since it exists.*

7 In the **Projects View**, click on the `GoogleResultsPage` screen class.

8 In the **XPath Evaluator**, click on the **Create new extraction rule from current XPath** icon  .

The **New Extraction Rule** wizard opens.

9 Click on **WebClipper**.

10 Click on **Next**.

11 In the **Name** field, enter a name (for example `WebClipResults`).

12 Click on **Finish**.

The new extraction rule appears in the **Extraction rules** screen class folder:

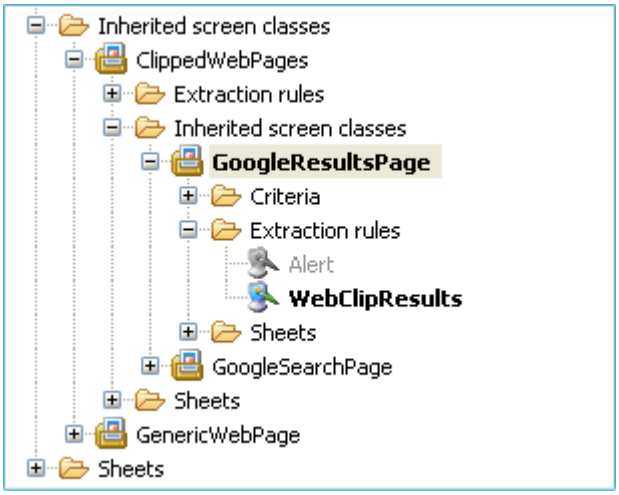


Figure 2 - 67: WebClipResults extraction rule in Extraction rules folder

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

We now want to enable the user to navigate from a result page to another as if in a standard Google page. To this end, we will clip the Google navigation bar displayed at the bottom of Google results pages.

A fourth extraction rule is therefore created: `WebClipNavigationBar`. This extraction rule is executed on detection of a Google results page, and displayed as in standard Google results pages, i.e. below results. After having created it, we wil check that it appears at the proper place in the **Extraction rules** folder of the `GoogleResultsPage` screen class.

14 Assuming that the **Web Browser** currently displays a Google results page, right-click on an element of the Google navigation bar (for example the first letter, "G").

A SPAN node is highlighted in green in the **DOM Tree**:

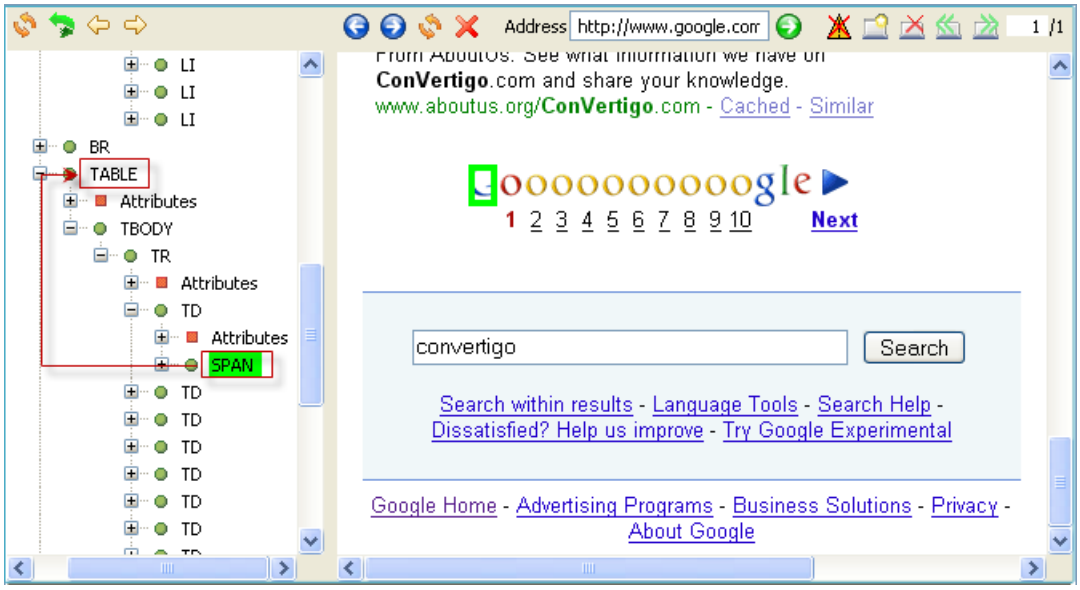


Figure 2 - 68: Selection of an XPath for the Google navigation bar

This container includes only the first letter of the navigation bar. We will go up the node

tree to find a parent node including the whole bar.

15 Go up the **DOM Tree** by clicking four times on the **Parent node** icon to find an appropriate node for the Google navigation bar.

The TABLE node seems a suitable element node since by selecting it, the whole navigation bar is outlined in green in the **Web Browser**:

Figure 2 - 69: Table node corresponding to the whole Google navigation bar

16 Expand the **DOM Tree** to select the most relevant node for generating the Xpath corresponding to this element.

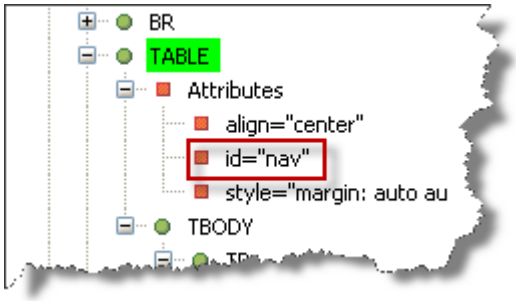


Figure 2 - 70: Expanding an element to find relevant attributes in the DOM Tree

An `id` node appears in the tree. As mentionned, this attribute always defines an HTML element in a unique manner, so it is the best candidate for generating the TABLE element Xpath.

17 Right-click the `id="nav"` node and select **Generate selection XPath (Enter)** or select it with a left-click and press **Enter**:

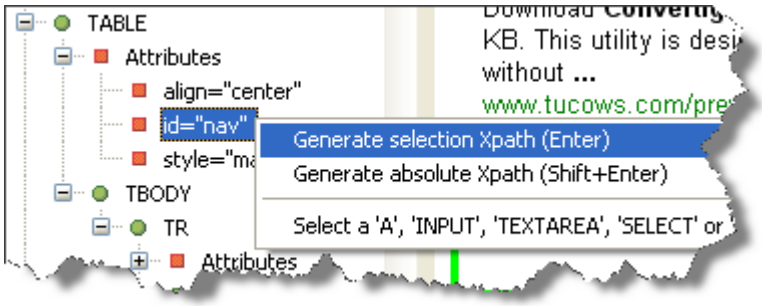


Figure 2 - 71: Generating XPath corresponding to navigation bar

The XPath expression (`//TABLE[@id="nav"]`) of the selected node appears in the **XPath Evaluator**.

*You can also right-click the parent TABLE element and select **Generate selection XPath (Enter)** or select the TABLE element with a left-click and press **Enter**: the XPath will intelligently be generated with the `id` attribute, since it exists.*

18 In the **Projects View**, click on the `GoogleResultsPage` screen class.

19 In the **XPath Evaluator**, click on the **Create new extraction rule from current XPath** icon  .

The **New Extraction Rule** wizard opens.

20 Click on **WebClipper**.

21 Click on **Next**.

22 In the **Name** field, enter a name (for example `WebClipNavigationBar`).

23 Click on **Finish**.

The new extraction rules appears in the **Extraction rules** screen class folder:

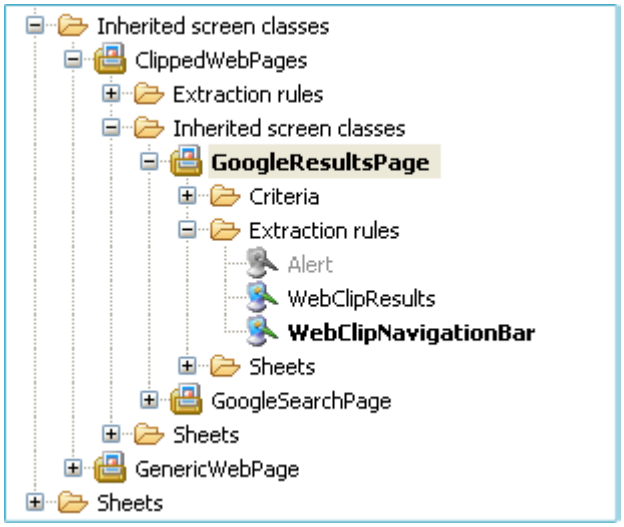


Figure 2 - 72: WebClipNavigationBar extraction rule in Extraction rules folder

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

Both extraction rules of the `GoogleResultsPage` screen class are sorted in the proper order: first results, then navigation bar.

What comes next?

All required screen classes and extraction rules are now defined:

- When CWC connects to the Web page corresponding to the **HTTP Server** parameter set as connector parameter (`www.google.com` displaying the Google search page), the `GoogleSearchPage` screen class is detected and associated extraction rules, `WebClipImage` and `WebClipForm`, are executed. HTML parts of the Google search page we want displayed to the user (Google logo and Google search form) are clipped and presented in a generated cliplet.

- After the user has launched a search from this *search* cliplet, the `GoogleResultsPage` screen class is detected and associated extraction rules, `WebClipResults` and `WebClipNavigationBar`, are executed. HTML parts of Google results page we want displayed to the user (Google results and Google navigation bar) are clipped and presented in an updated cliplet.
- As long as the user clicks on the navigation bar **Next** link, next Web screens are detected as `GoogleResultsPage` screen classes and corresponding extraction rules are executed.

We will now see if a transaction is needed in our project and when.

2.2.4 Defining a Transaction

This section shows how to create and define a transaction.

A transaction can be defined as an automated navigation based on:

- *detection* of defined screen classes;
- *triggering* of screen class handlers and *execution* of associated statements (data inputs, mouse clicks, checkbox checks, event creation, etc.).

The very first step of the navigation process is the connection to the Web site accessed through the **HTTP Server** parameter set for the connector (see *"To create a project and set a connector"* on page 2-6).

As regards this first connection, two situations are involved:

- you're accessing the `www.google.com` Web site from the USA or from a country not having a country-specific Google page - see *"Definition of a Standard Connection Transaction"* on page 2-43;
- you're accessing the `www.google.com` Web site from a country having a country-specific Google page - Google servers automatically redirect CWC to the "localized" version of the Google search engine - see *"Definition of a Country-Specific Connection Transaction"* on page 2-46.

In both cases, a connection transaction (for example `GoGoogleCom`) must be defined to connect to the Google Web site (as set as connector parameters when creating the project, see *"Creating a Project and Setting Connector Parameters"* on page 2-6).

If need be (that is, if a country specific page is accessed), the transaction must also redirect the connector to the standard Google search page (and allow CWC to detect the `GoogleSearchPage` screen class).

Definition of a Standard Connection Transaction

This sub-section applies only to users accessing the `www.google.com` Web site from a country not having a country-specific Google search page.

To define a standard transaction

1 In the **Projects View**, right-click on the connector name (for example

`GoogleConnector`) or on the **Transactions** folder.

2 Select ***New > Transaction***:

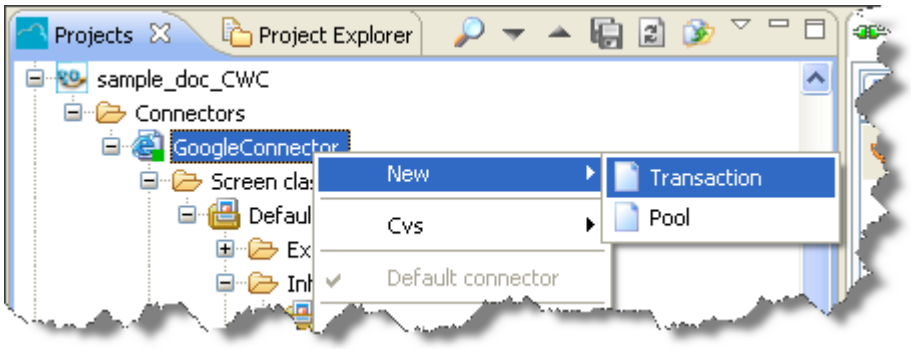


Figure 2 - 73: Creating a new transaction

A **New Transaction** wizard opens:

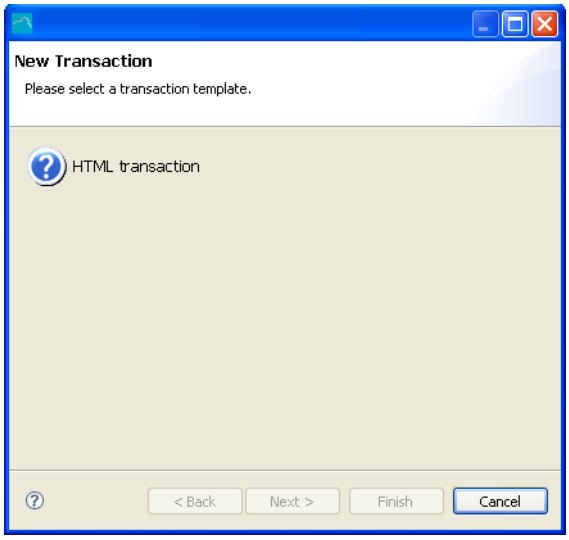


Figure 2 - 74: New Transaction wizard

3 Select **HTML Transaction**.

4 Click on **Next**:

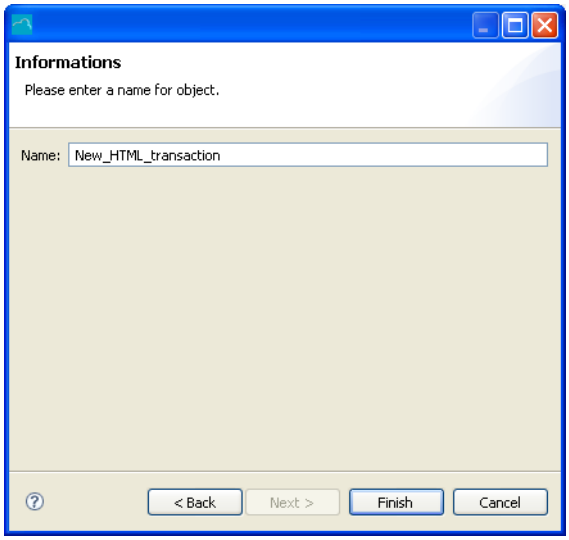


Figure 2 - 75: Entering a new transaction name

5 Enter the transaction name (for example `GoGoogleCom`).

6 Click on **Finish**.

The new transaction appears in the **Transactions** folder of the **Projects View**:

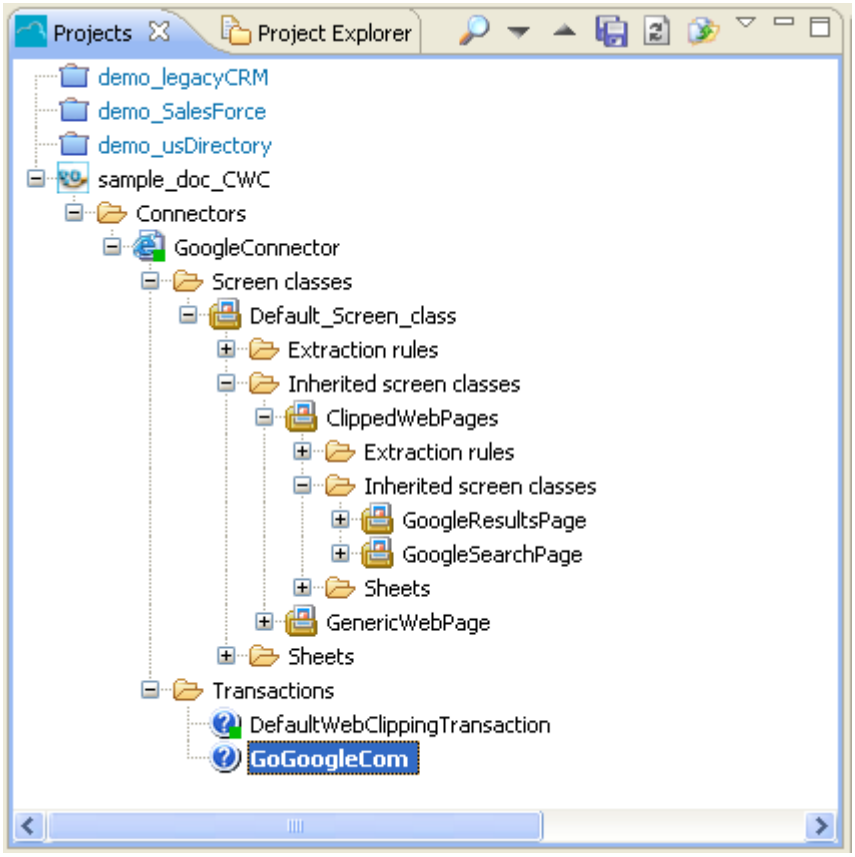


Figure 2 - 76: New transaction in Projects View

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

A property differs between the `DefaultWebClippingTransaction` and the `GoGoogleCom` transaction: **Maintains connector state**.

The table below describes this property:

Table 2 - 3: Maintains connector state property

Value	Description
false	Each time the transaction is executed, the connector connects to the target Web page specifed in the **Sub path** property using the **HTTP Server** parameter set for the connector, and the **Synchronizer** property for synchronization.
true	The transaction uses the browser in the state it has been left by the preceding transaction execution.

In our project, we want CWI to connect to the `www.google.com` Web site (**HTTP Server** parameter set for the connector) each time the `GoGoogleCom` transaction is executed. We can leave this property set to its default value (`false`).

Definition of a Country-Specific Connection Transaction

When accessing the `www.google.com` from outside the USA or from a country having a country-specific Google page, a country-specific transaction must be created. It is similar to the previously defined transaction, but needs a screen class handler to be defined as well. This screen class handler handles the redirecting of CWC form the country-specific Google search page to the standard Google.com search page

How can the `GoGoogleCom` transaction be defined?

This transaction can be defined as follows:

- it contains a screen class handler associated with the screen class identifying country-specific Google homepages (`GoogleSearchPageFR` in our project, see *"Case of a Country-Specific Google Search Page - Definition of the `GoogleSearchPageFR` Screen Class"* on page 2-25).
- the screen class handler is triggered on detection of the country specific Google homepage screen class. This statement performs a "mouse click" statement on the redirecting link of the country-specific homepage (**Google.com in English** or **Go to Google.com**).

How can the `GoGoogleCom` transaction be described?

In the context of our sample project, this transaction can be summed up as follows:

1. CWC accesses the `www.google.com` Web site.
2. Google servers automatically redirect CWC to the country-specific Google search page.
3. The country-specific version of the Google homepage is detected as a country-specific screen class (for example `GoogleSearchPageFR`).
4. On detection of the country-specific screen class (`GoogleSearchPageFR`), the corresponding country-specific screen class handler (`onGoogleSearchPageFREntry`) is triggered.
5. The statement associated with the screen class handler ("click on the redirecting link") is performed.

6 CWC is automatically redirected to the standard Google search page in English (detected as a `GoogleSearchPage` screen class).

7 The transaction stops, extracting data from this page and presenting the cliplet to the user.

The Web clipping automatic process starts. A search can be launched by the user from the generated cliplet. Screen classes are then automatically detected and data automatically clipped (as explained in the previous section (page 2-42)).

For more information on screen class handlers and statements, see "Screen class handler" on page 1-8 and see "Statement" on page 1-9.

The following procedure shows how to create the country-specific transaction, screen class handler and statement previously identified.

To define a country-specific transaction

1 In the **Projects View**, right-click on the connector name (for example `GoogleConnector`) or on the **Transactions** folder.

2 Select ***New > Transaction***:

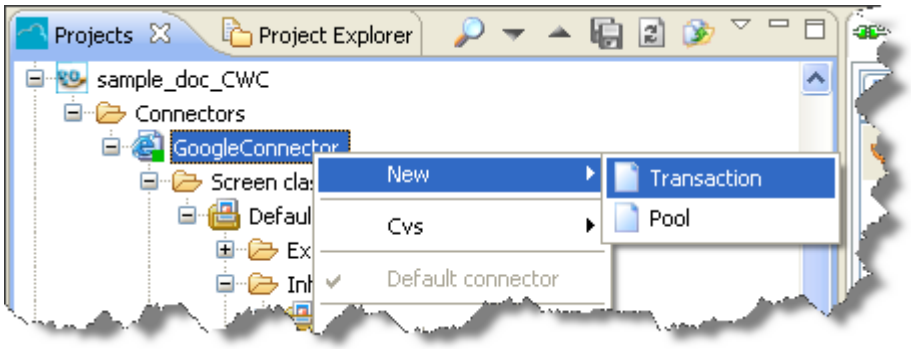


Figure 2 - 77: Creating a new transaction

A **New Transaction** wizard opens:

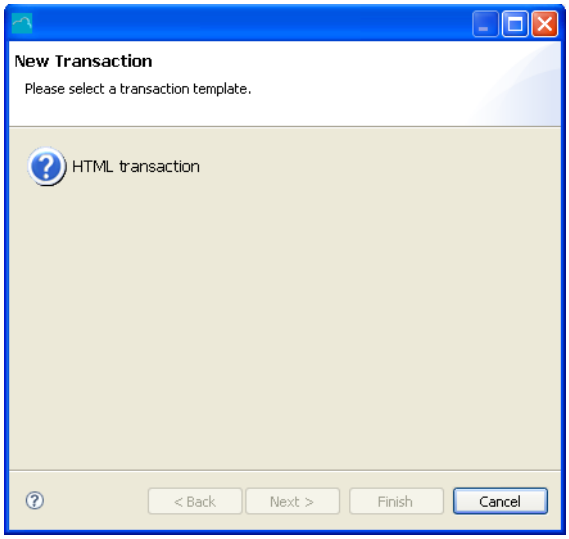


Figure 2 - 78: New Transaction wizard

3 Select **HTML Transaction**.

4 Click on **Next**:

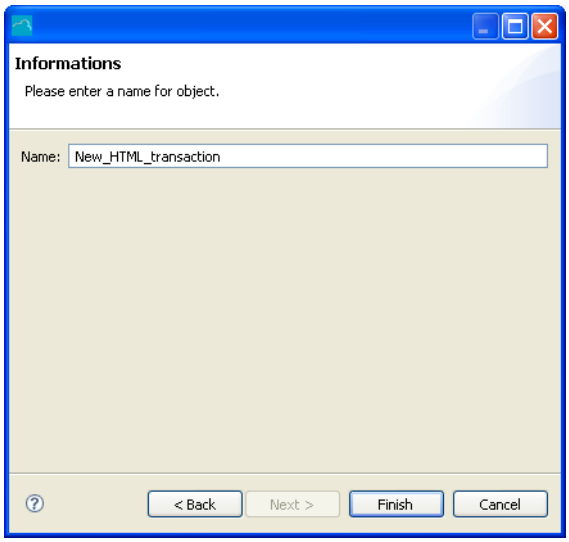


Figure 2 - 79: Entering a new transaction name

5 Enter the transaction name (for example `GoGoogleCom`).

6 Click on **Finish**.

The new transaction appears in the **Transactions** folder of the **Projects View**:

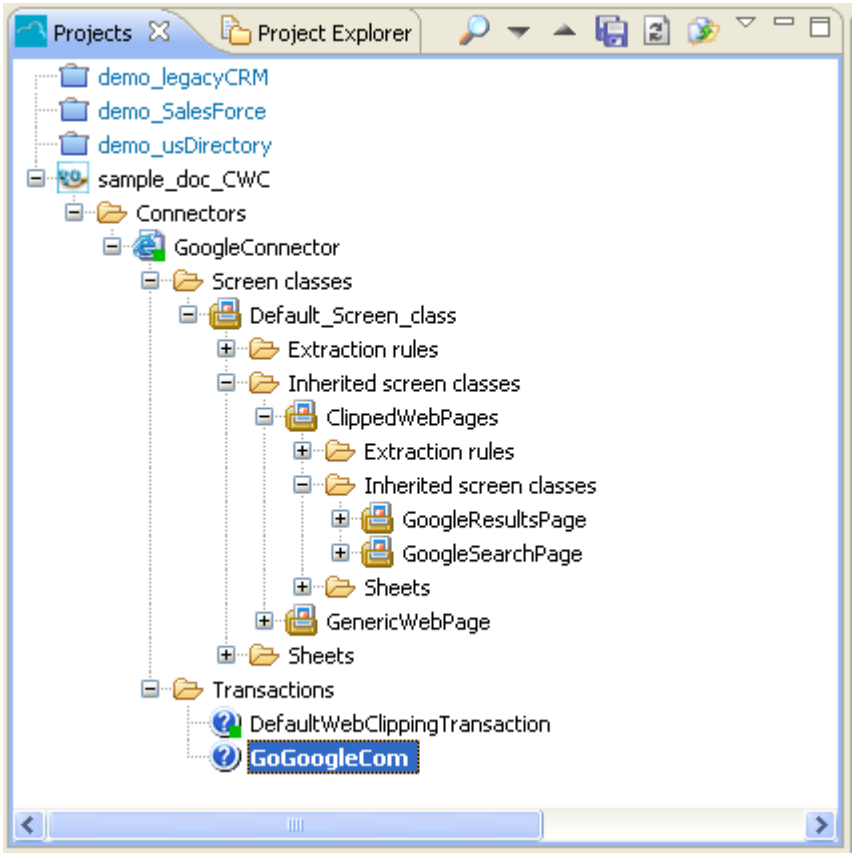


Figure 2 - 80: New transaction in Projects View

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

A property differs between the `DefaultWebClippingTransaction` and the `GoGoogleCom` transaction: **Maintains connector state**.

The table below describes this property:

Table 2 - 4: Maintains connector state property

Value	Description
false	Each time the transaction is executed, the connector connects to the target Web page specifed in the **Sub path** property using the **HTTP Server** parameter set for the connector, and the **Synchronizer** property for synchronization.
true	The transaction uses the browser in the state it has been left by the preceding transaction execution.

In our project, we want CWI to connect to the `www.google.com` Web site (**HTTP Server** parameter set for the connector) each time the `GoGoogleCom` transaction is executed. We can leave this property set to its default value (`false`).

We will now create the screen class handler triggered on detection of the country-specific screen class (`GoogleSearchPageFR`). A statement will then be associated with this screen class handler: "click on the redirecting link".

To create a country-specific screen class handler

1. Right-click on the transaction name (for example `GoGoogleCom`).
2. Select ***New > Handler*** in the contextual menu.

   A **New transaction handler** window opens:

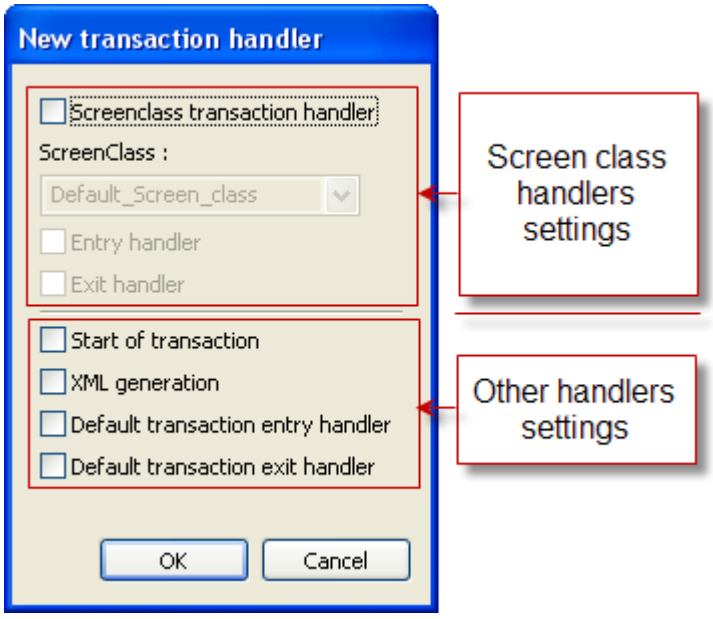


Figure 2 - 81: New transaction handler window

This window is divided into:

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

The table below describes these elements:

Table 2 - 5: New transaction handler window description

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

The screen class handler we are about to create is triggered when accessing the country-specific Google search page.

It can be defined as follows:

- this is a *screen class transaction handler*, meaning that all the screen class handler settings of the **New transaction handler** window (see Figure 2 - 81) must be set;
- this is an ***entry*** handler, because no data has to be extracted from this page;
- it is invoked on detection of a country-specific screen class (for example `GoogleSearchPageFR`).

We will now set the scren class handler parameters:

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

4 In the **ScreenClass** drop-down menu, select the country-specific screenc class (for example `GoogleSearchPageFR`).

5 Check the **Entry handler** checkbox:

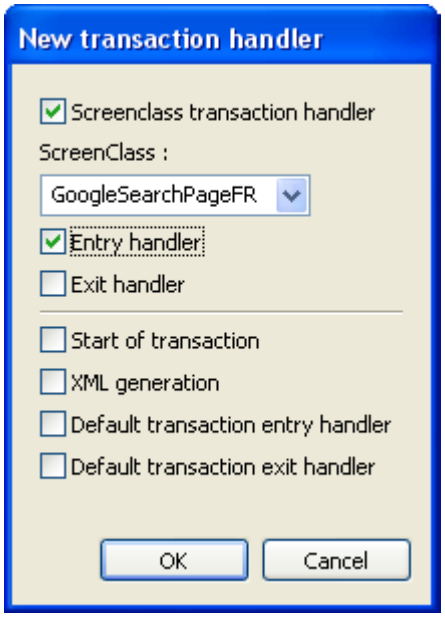


Figure 2 - 82: ongoogleSearchPageFREntry settings

6 Click on **OK**.

The country-specific screen class handler (for example `ongoogleSearchPageFREntry`) appears in the **Functions** folder of the `GoGoogleCom` transaction of the **Projects View**:

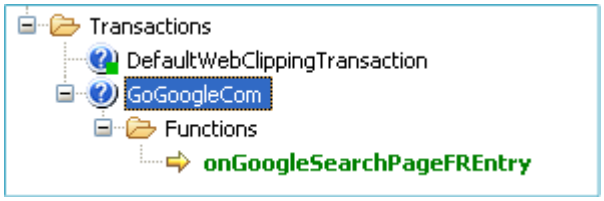


Figure 2 - 83: New entry screen class handler in the Projects View

Note the left to right yellow arrow  identifying *entry* screen class handlers.

We now need to define the statement that performs the required action on detection of the country-specific screen class: "click on the redirecting link".

To this end, we will use the same XPath as the one used to create the country-specific screen class, which corresponds to the link.

To create a statement

1 Assuming that the **Web Browser** displays a country-specific Google search page, right-click on the redirecting link (**Google.com in English** or **Go to Google.com**).

The link is outlined in green in the **Web Browser** and its corresponding element is highlighted in green in the **DOM Tree**:

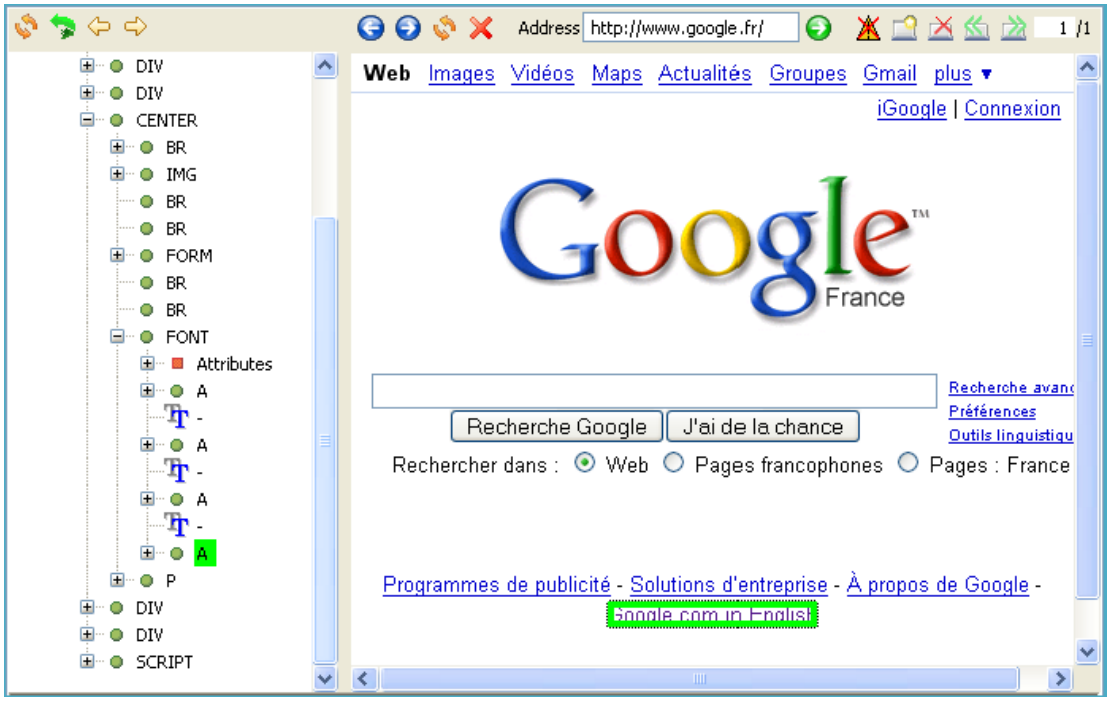


Figure 2 - 84: Selection of a detection criterion for the GoogleSearchPageFR screen class

We will expand the A element node to see if it contains interesting attributes to generate the XPath corresponding to the link.

2 Expand the A element in the **DOM Tree** by clicking on ⊞:

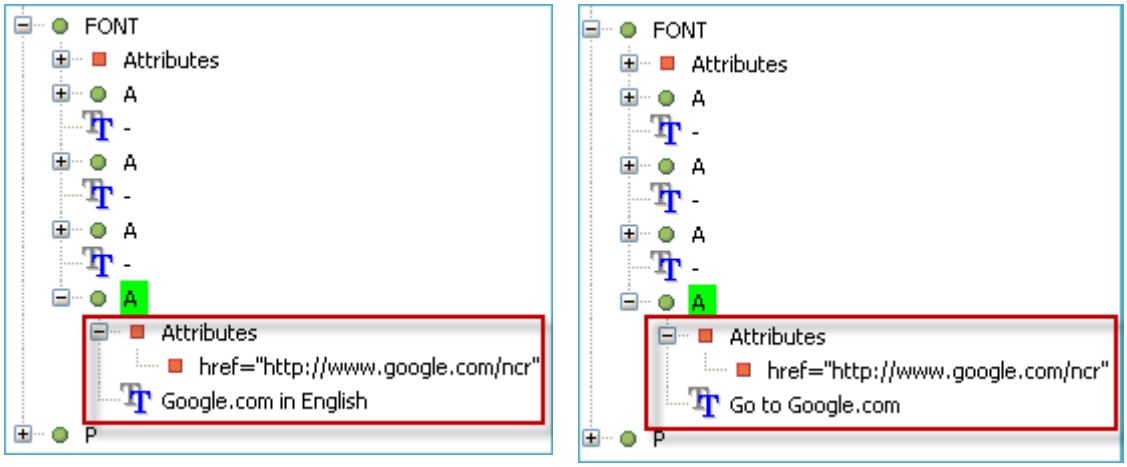
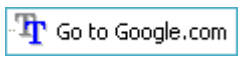


Figure 2 - 85: Expanding an element to find relevant attributes in the DOM Tree

Two nodes are associated with the A element: an `href` attribute node, which contains the URL address of the hypertext link, and a text node (`Google.com in English` or `Go to Google.com`), which represents the element content.

The `href` attribute is not so interesting as regards XPath generation because its value may often change. We will select the text content only and generate the XPath from it.

3 Right-click the text node 

or 

and select **Generate selection XPath (Enter)** or select it with a left-click and press **Enter**:

The XPath expression of the selected node appears in the **XPath Evaluator**:

Figure 2 - 86: XPath expression of country-specific redirecting link

4 Select the country-specific screen class handler object (`onGoogleSearchPageFREntry`) in the **Projects View**.

5 In the **XPath Evaluator**, click on the **Create new statement from current XPath** icon  , which is activated because a suitable handler or container is selected in the Projects view.

A **New Statement** wizard opens.

6 Click on **Mouse action** (this statement emulates mouse clicks):

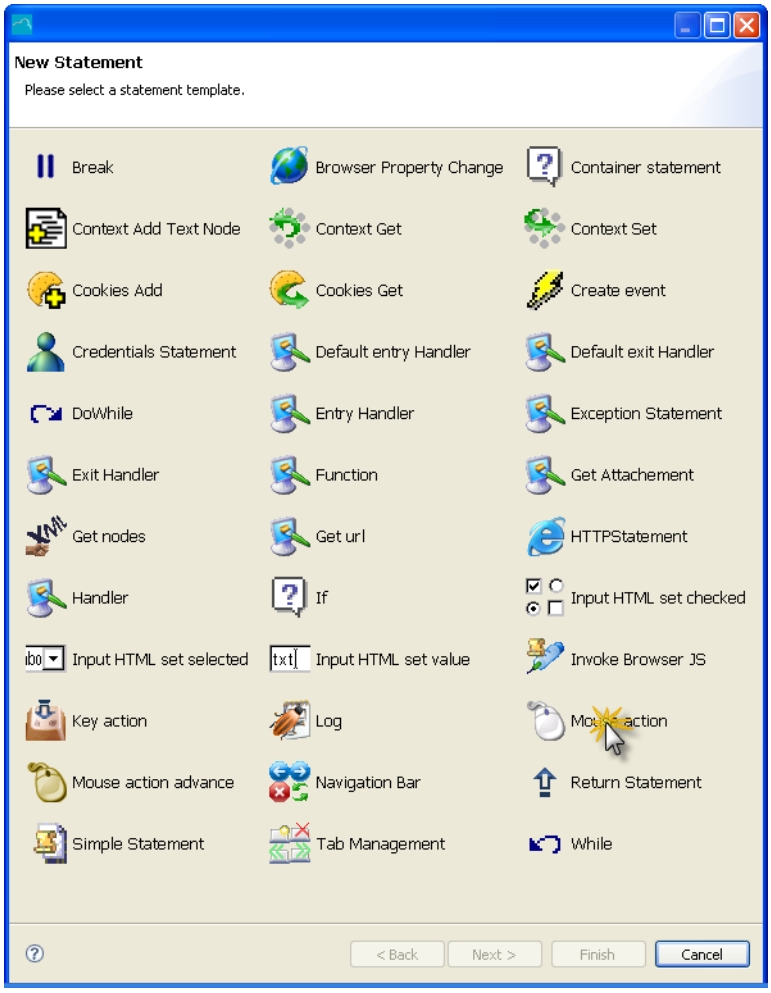


Figure 2 - 87: New Statement wizard - Mouse action

7 Click on **Next**.

8 Leave the statement name untouched or enter a new name in the **Name** field: for

example `clickGoogleCom`.

If the statement name is several words long, separate each word with an underscore instead of a blank space. Otherwise, an error message will appear in the Informations part of the window (see Figure 2 - 89).

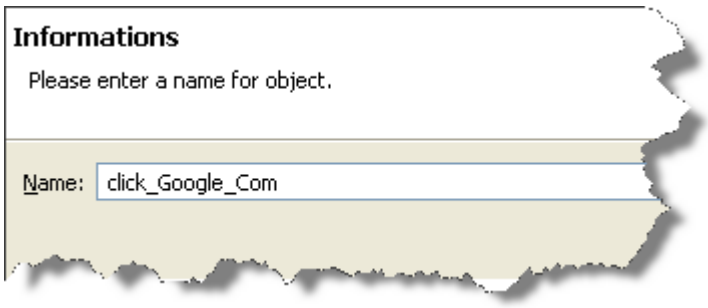


Figure 2 - 88: Proper statement name

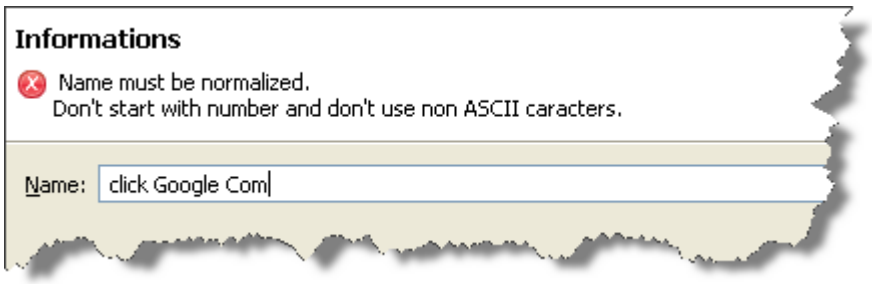


Figure 2 - 89: Improper statement name

9 Click on **Finish**.

The new statement appears in the **Projects View**:

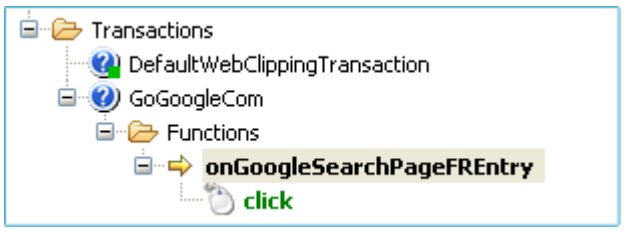


Figure 2 - 90: New Mouse click statement on entry screen class handler

The statement's **Synchronisation** property is set by default to the following value:

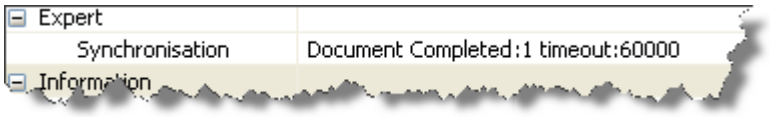


Figure 2 - 91: clickGoogleCom statement - Synchronization value

This property answers the following question: "Once the click has been performed, when can the transaction resume (for example once the Google.com page has has been loaded), and what is the allowed maximum waiting time?".

10 In the **Properties View**, click on the value of the **Synchronisation** property.

11 Click on  .

A **Trigger Editor** window opens:

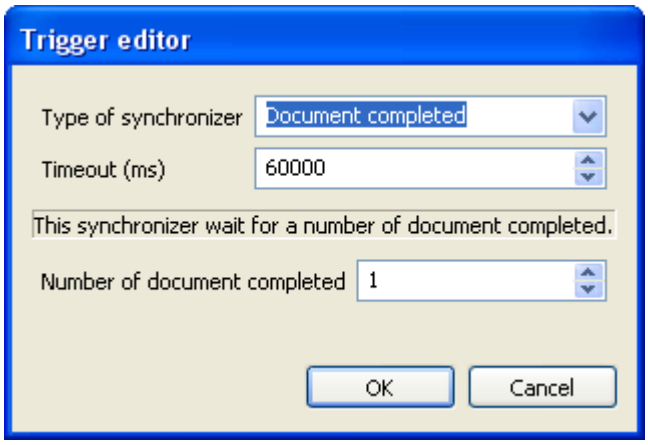


Figure 2 - 92: Trigger editor window

Three types of synchronizer are available:

Table 2 - 6: Synchronizer types

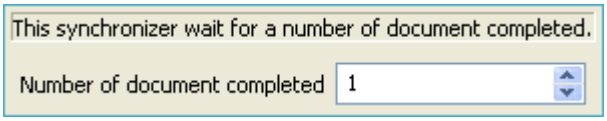
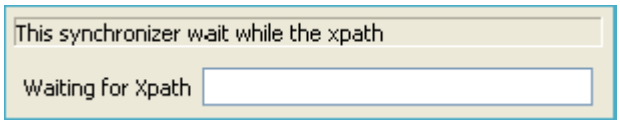
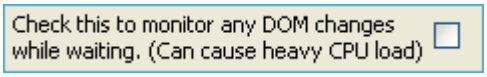
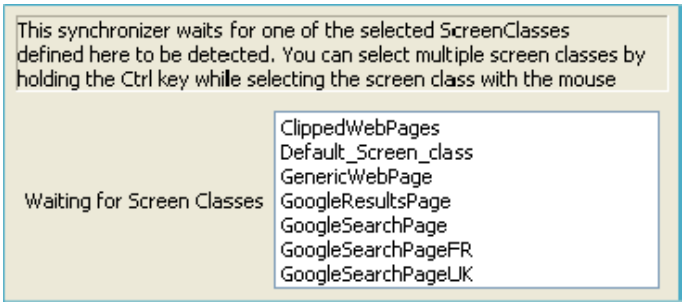
Synchronizer type	Description
Document completed	Transaction continues running once a number of document have been successfully loaded. The number of documents must be set in the **Number of document completed** field of the window: 
Xpath	Transaction continues running once an Xpath has been found. The XPath must be set in the **Waiting for XPath** field: 
Wait time	Transaction continues running after the specified wait time has been reached. With this parameter, you can also decide to monitor any DOM change while waiting: 

Table 2 - 6: Synchronizer types

Synchronizer type	Description
Screen class	Transaction continues running once one of the specified screen classes has been reached. The waited screen classes must be selected in the **Waiting for Screen Classes** list: 

12 Leave default values unchanged.

13 Click on **OK**.

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

The actions to be performed on detection of the `googleSearchPageFR` screen class are now defined.

We must inform the transaction that once the screen class handler statement has been executed, a new criterion match detection must be carried out on the next accessed Web page without extracting data.

This is done through the **Result** entry screen class property, which can take the following values:

Table 2 - 7: Result property values for *entry* screen class handlers

The value...	Means that the application will...
<blank>	stop the transaction and extract data from last detected screen class using extraction rules
continue	stop the transaction and extract data from last detected screen class using extraction rules.
redetect	redetect a new screen class without extracting data. This is the default value, which corresponds to what the transaction should do in our case.
skip	stop the transaction without extracting data from last detected screen class.

This property is set by default to `redetect`. You can leave it as is.

The `onGoogleSearchPageFREntry` screen class handler is now properly set together with the required statement.

No handler needs to be created for the `GoogleSearchPage` screen class, because we only want the transaction to stop when detecting this screen class and extract data, which is implied by the "return redetect" process of the previous handler.

WHAT COMES NEXT?

Now that the transaction has is created, test it by executing it to see if it gives expected results.

2.2.5 Executing a Transaction

The Web clipping project is now set up.

You can execute the transaction either:

- by executing it from the Studio using the **Execute** command;
- by calling a properly formatted URL from a Web browser to access a test platform included in the project.

To execute the transaction from the Studio

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

2 Select **Execute**.

The transaction starts. The successive steps of the transaction (connect to `www.google.com` then click on the **Google.com in English** link and access the standard Google search engine, if applicable) are executed; you can see them in real time in the **Connector View** of the studio.

The HTML content of extracted (i.e. clipped) data appears in the **XML** tab of the **Output** tab of the **Connector View**:

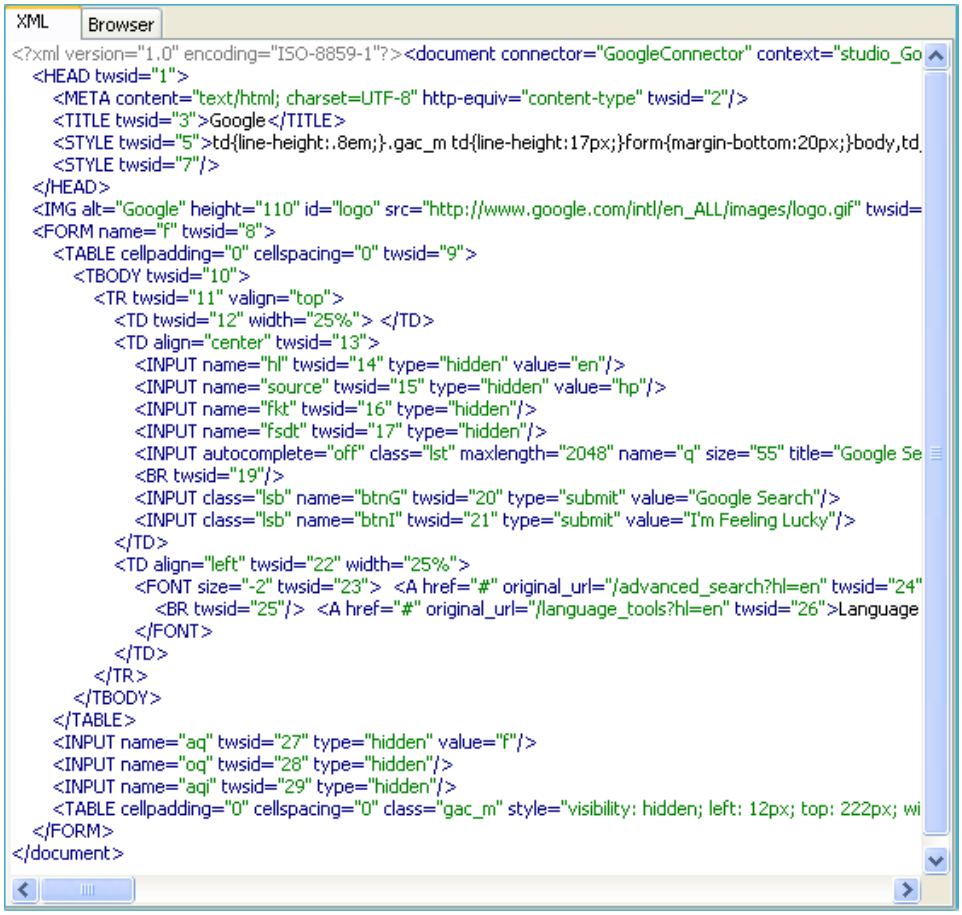


Figure 2 - 93: XML output of HTML elements clipped from Google search page

The last detected screen class is displayed in the lower left corner of the **Connector View**:

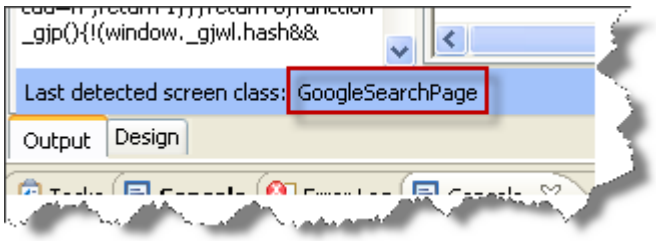


Figure 2 - 94: Last detected screen class

When a screen class is detected, corresponding extraction rules are executed. As regards the `GoogleSearchPage` screen class, these extraction rules are the following:

- `WebClipImage` - clips the Google logo;
- `WebClipForm` - clips the Google search form.

The HTML code of clipped elements appear in the **XML** tab:

Figure 2 - 95: HTML code of clipped Google logo and Google search form

The Google search page HTML header also appears in the output:

Figure 2 - 96: HTML page header in XML output

The header is automatically extracted by the *Web clipper* extraction rule. In this example, only one header is present because clipped elements belong to the same HTML page (Google search page).

This test shows that :

- The `GoogleSearchPage` screen class has been correctly detected.
- Corresponding extraction rules have been performed in consequence.

Data have been extracted as expected - the test is conclusive.

To test the transaction using the test platform

1 Launch a standard Web browser.

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

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

For example, in our project:

- *`HostName`* = `localhost` (default server for studio);
- *`ConvertigoPortNumber`* = `18080` (default port for studio);
- *`ProjectName`* = `sample_doc_CWC`;

For example, in a browser URL address bar:

Figure 2 - 97: Convertigo test platform URL

The test platform opens:

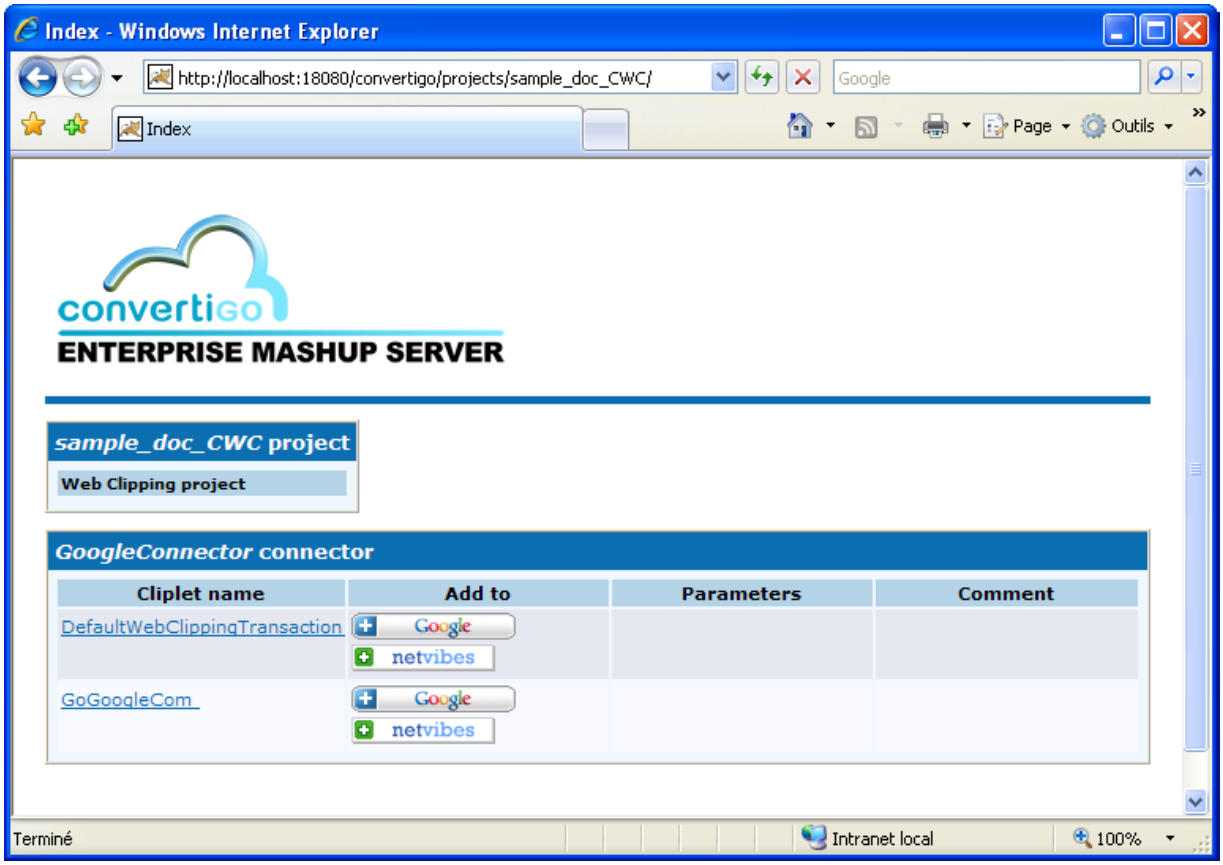


Figure 2 - 98: Test platform page

It displays:

- in a first table:
  - the **project name** (in our example, `sample_doc_CWC`);
  - the **project type**;
- in a second table:
  - the ***connector*** defined for the project (in our example, `GoogleConnector`);
  - all ***transactions*** defined for the connector (in our example, `DefaultWebClippingTransaction`, `GoGoogleCom`).

3 Click on the `GoGoogleCom` link.

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

The HTML content of extracted (i.e. clipped) data appear in an **Execution result** window in the test platform:

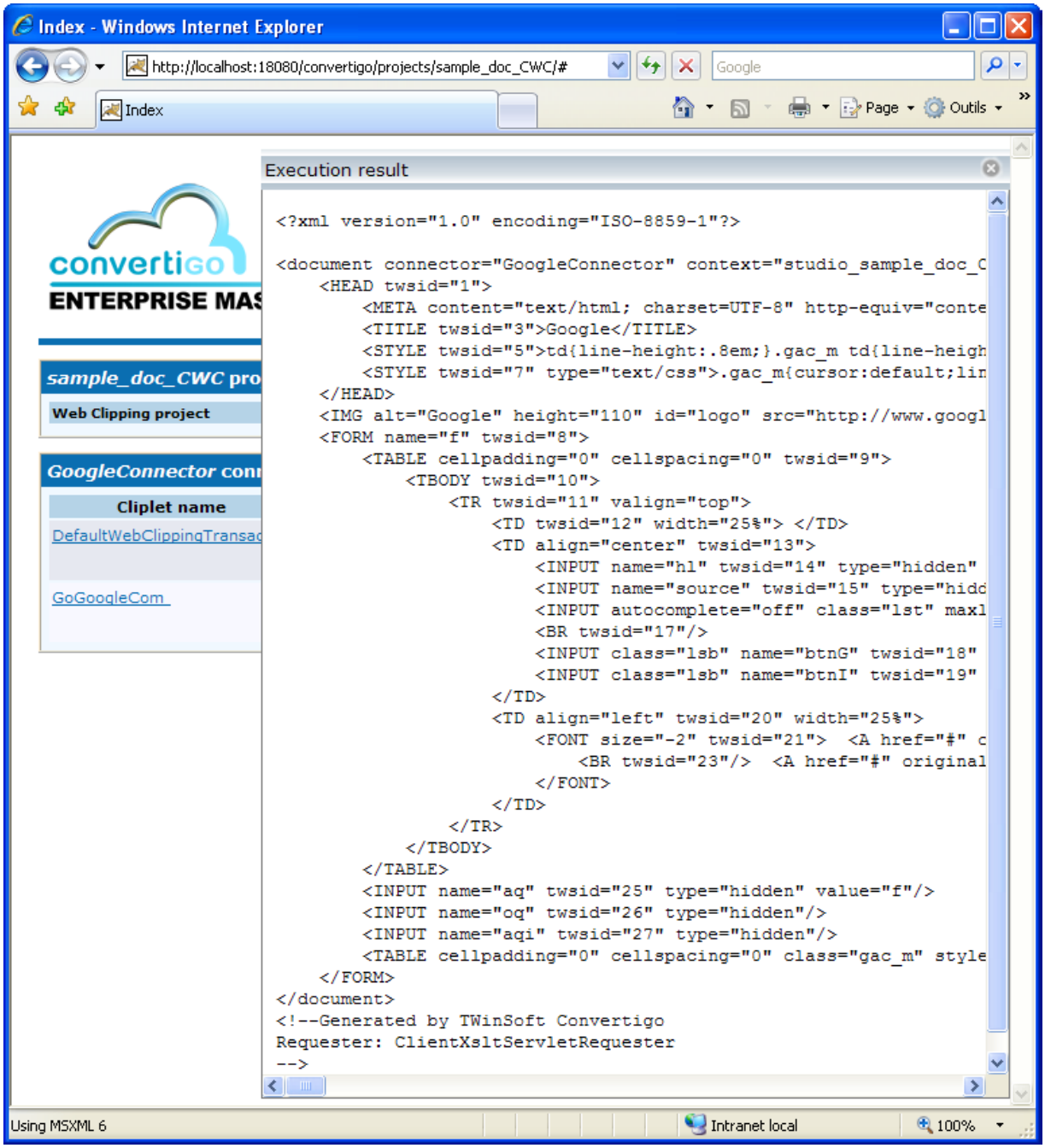


Figure 2 - 99: XML output of HTML elements clipped from Google search page

As mentioned in the previous procedure (see *"To execute the transaction from the Studio"* on page 2-57), the XML output contains:

- the header of the page from which data have been extracted (here, one header only because elements have been clipped on same HTML page);
- the HTML code corresponding to the Google logo;
- the HTML code corresponding to the Google search form.

Figure 2 - 100: HTML code of clipped elements

Data have therefore been extracted as expected - the test is conclusive.

We now need to associate the transaction with a style sheet, so as to perform an XSL transformation of the XML output into an HTML page (see *"Presenting Data with Convertigo Web Clipper"* on page 3-1).

3 Presenting Data with Convertigo Web Clipper

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

This chapter introduces the notion of XSL transformation. It describes how to transform XML data into an HTML document for display purpose using XSL style sheets.

3.1 Presenting Data

To be able to present data in HTML format, a style sheet in XSL format must be associated with the transaction and used to process the resulting XML output.

In CWC, a style sheet is created by default when creating a project: `DoNothing`. It is stored in the **Sheets** folder of the `Default_Screen_Class` screen class. All defined screen classes therefore inherits it.

3.1.1 The `DoNothing` Style Sheet

The `DoNothing` style sheet is associated with the default screen class, `Default_Screen_Class`:

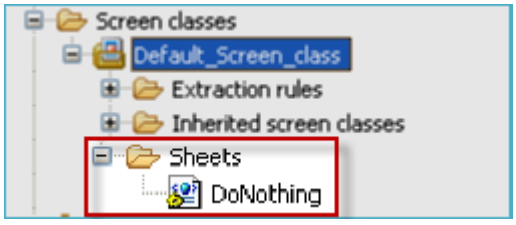


Figure 3 - 1: DoNothing style sheet

It "does nothing" in the sense that it does not process XML nodes in any specific way. Clipped data are initially extracted in HTML format - the `DoNothing` style only recopies these HTML elements as they are. When associating the transaction with the `DoNothing` style sheet, clipped elements just appear as they do in source HTML pages.

*The `DoNothing` style sheet that appears in the **Projecs View** is a CWC object. It is associated with an XSL file via its **URL** property: `donothing.xsl`.*

Screen classes previously defined (see *"Defining Screen Classes"* on page 2-10) are child screen classes of the `Default_Screen_Class`.

As such, they inherit the `DoNothing` style sheet:

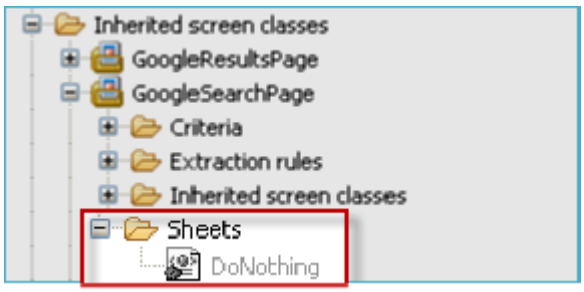


Figure 3 - 2: Inherited DoNothing style sheet

3.1.2 Associating the `DoNothing` Style Sheet with Transactions

We will now associate the `DoNothing` style sheet with the transaction (for example `GoGoogleCom`), through the **Stylesheet** transaction property.

To associate a style sheet with a transaction

1 Select the transaction with a left-click.

The **Properties View** is updated:

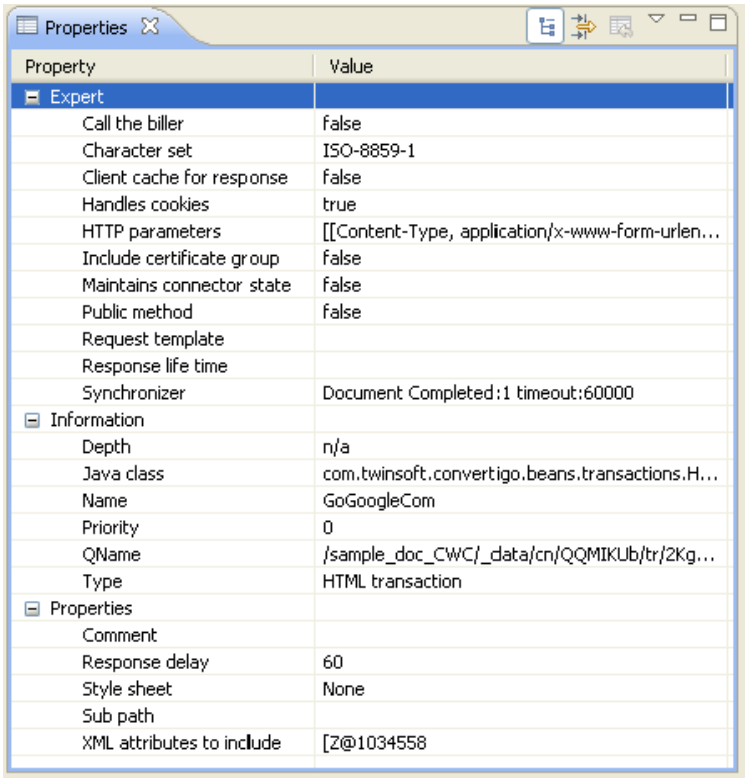


Figure 3 - 3: Transaction properties

The **Style Sheet** property can take three values:

Table 3 - 1: Style Sheet transaction property

Style Sheet property value	Description
None	Does not process with XSLT. Usually set for Web services (SOAP or REST), as plain XML data is then to be returned.
From transaction	Uses the XSL style sheet attached to this transaction.
From last detected screen class	Uses XSL style sheet attached to the last detected screen class.

In our case, we want the style sheet attached to detected screen classes to be processed. We must set the **Style Sheet** property to **From last detected screen class**.

2 Click on the value of the **Style Sheet** property.

3 Select **From last detected screen class** in the drop-down menu:

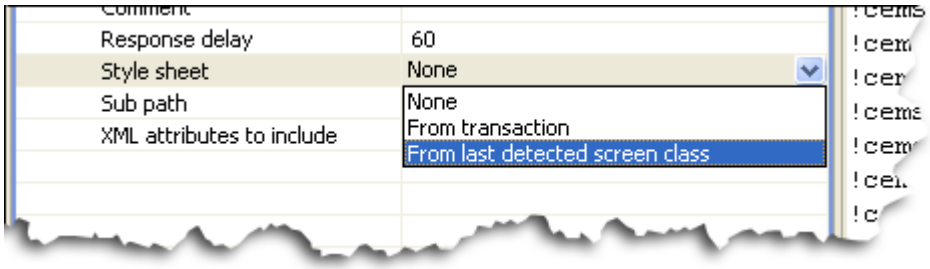


Figure 3 - 4: Setting the Style Sheet property to From last detected screen class

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

We will now re-test the transaction to check that the XSL transformation gives expected results.

To re-test the transaction

1 Repeat steps 1 and 2 of procedure *"To execute the transaction from the Studio"* on page 2 - 57 OR repeat steps 1 to 3 of procedure *"To test the transaction using the test platform"* on page 2 - 59.

If the transaction is executed from the Studio, the result of the XSL transformation (i.e. the generated cliplet) appears in the **Browser** tab of the **Output** tab of the **Connector View**:

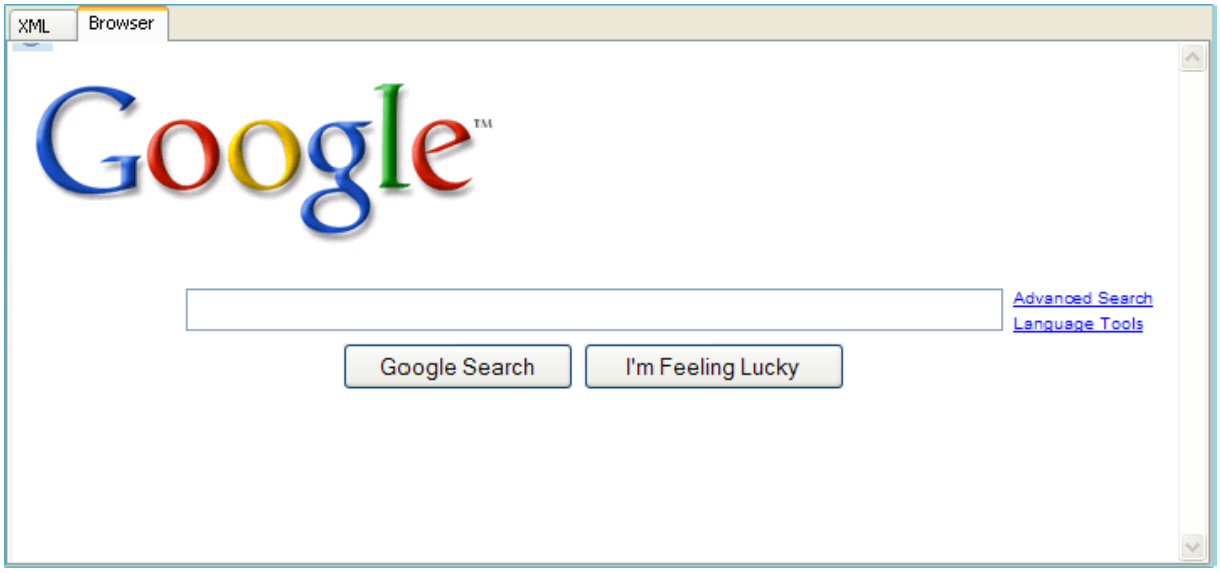


Figure 3 - 5: Cliplet based on clipped HTML elements

If the transaction is executed from the test platform, it appears in the **Execution result** window of the test platform:

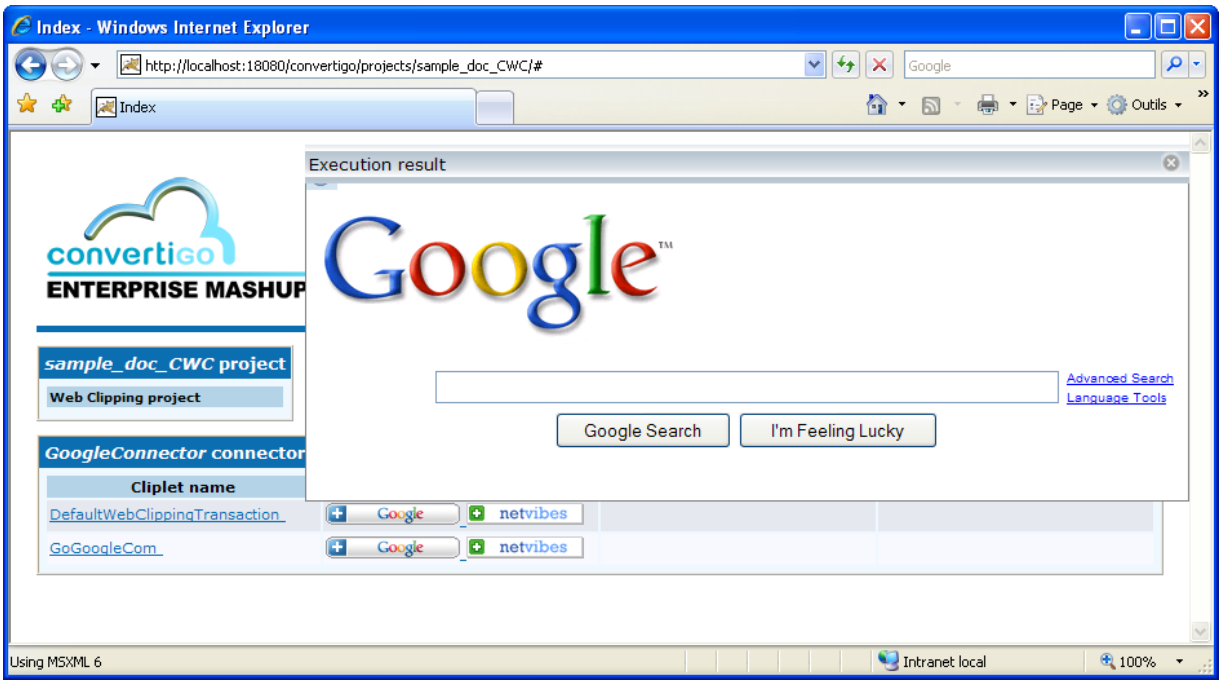


Figure 3 - 6: XML output in the Execution result window of the test platform

In both cases, the Google logo and search form appear off-centered (see Figure 3 - 5 or see Figure 3 - 6) whereas they appear centered in the original Google page. Image centering is represented in the **DOM Tree** by the `CENTER` element node, which is a *parent* element of `IMG` (element corresponding to the Google logo):

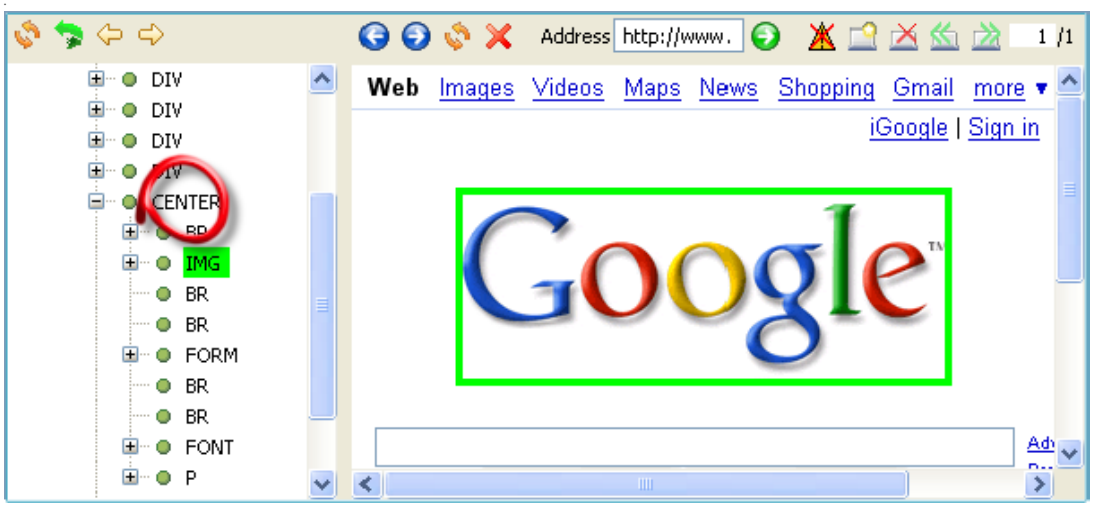


Figure 3 - 7: Element node managing Google logo centering

Extracting the whole `CENTER` element with the Web clipper extraction rule is inappropriate because the element contains unwanted HTML elements. Instead, we will use the extraction rule **Enable parent extraction of the cliplet** property, which allows the extraction rule to extract the full parent tree of the clipped elements when set to `true`.

To enable the parent extraction of clipped elements

1 In the **Projects View**, select the `WebClipImage` extraction rule in the **Extraction rules** folder of the `GoogleSearchPage` screen class:

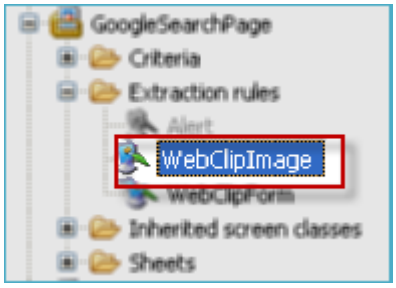


Figure 3 - 8: Selection of extraction rule in Projects View

The **Properties View** is updated:

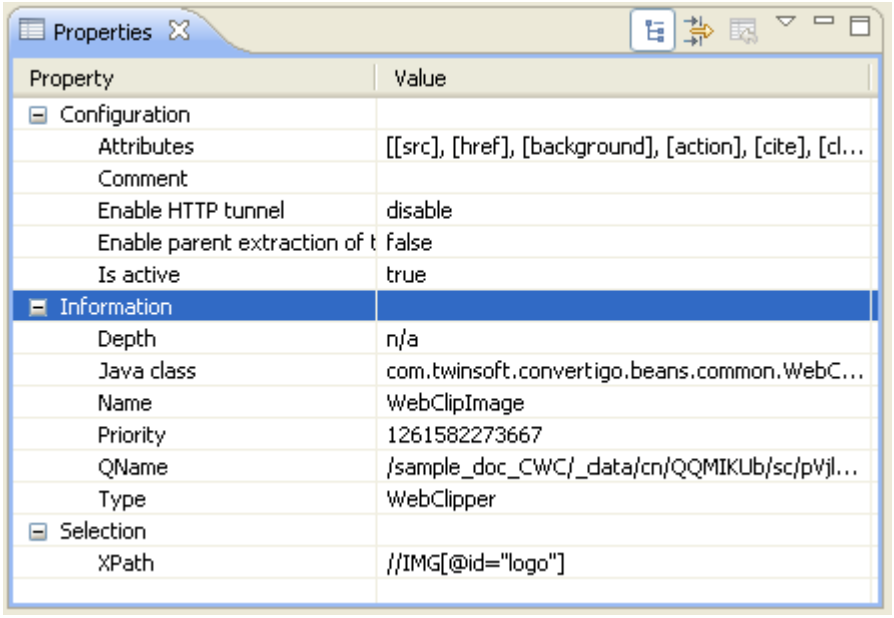


Figure 3 - 9: Extraction rule properties

2 Set the **Enable parent extraction of the cliplet** property to `true`:

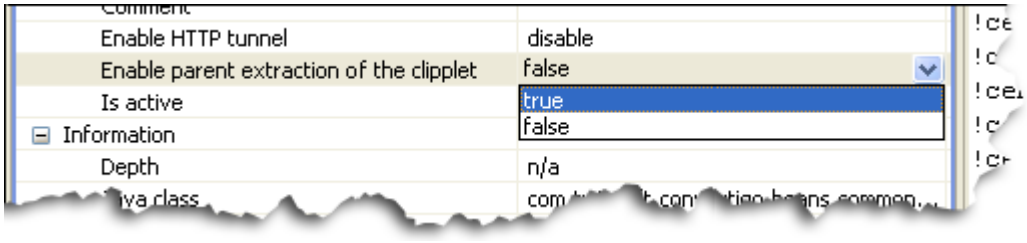


Figure 3 - 10: Setting Enable paretn extraction of the cliplet property to true

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

4 Repeat the procedure for the `WebClipForm` extraction rule.

When executing the transaction again, clipped elements appear centered:

- in the **Browser** tab of the **Output** tab of the **Connector View:**

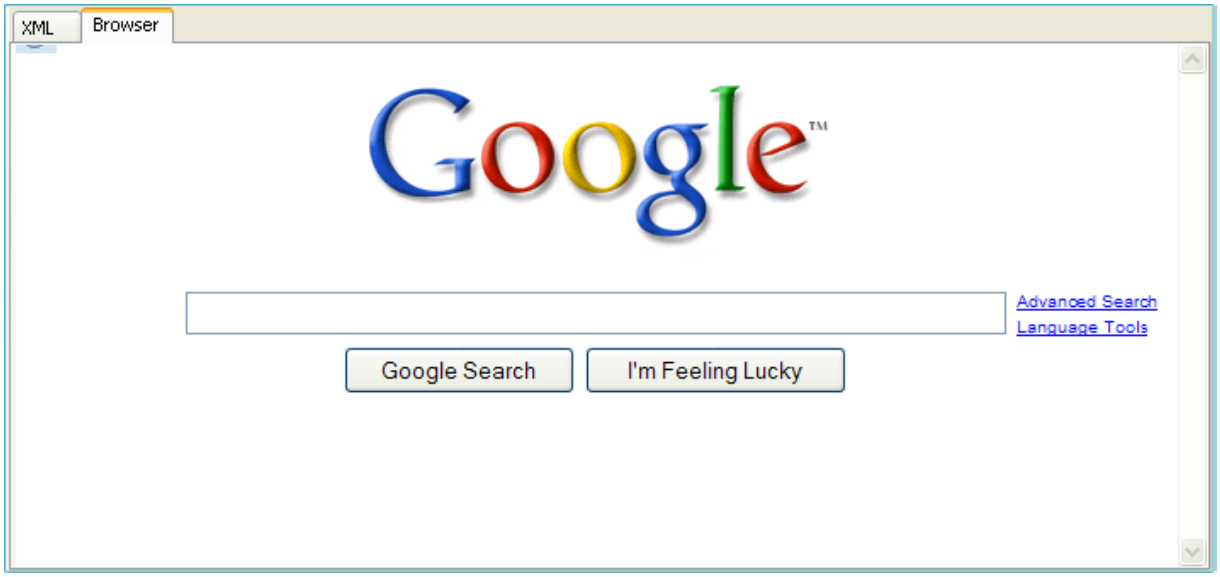


Figure 3 - 11: Cliplet in Browser tab of Studio with parent extraction enabled

- in the **Execution window** of the test platform:

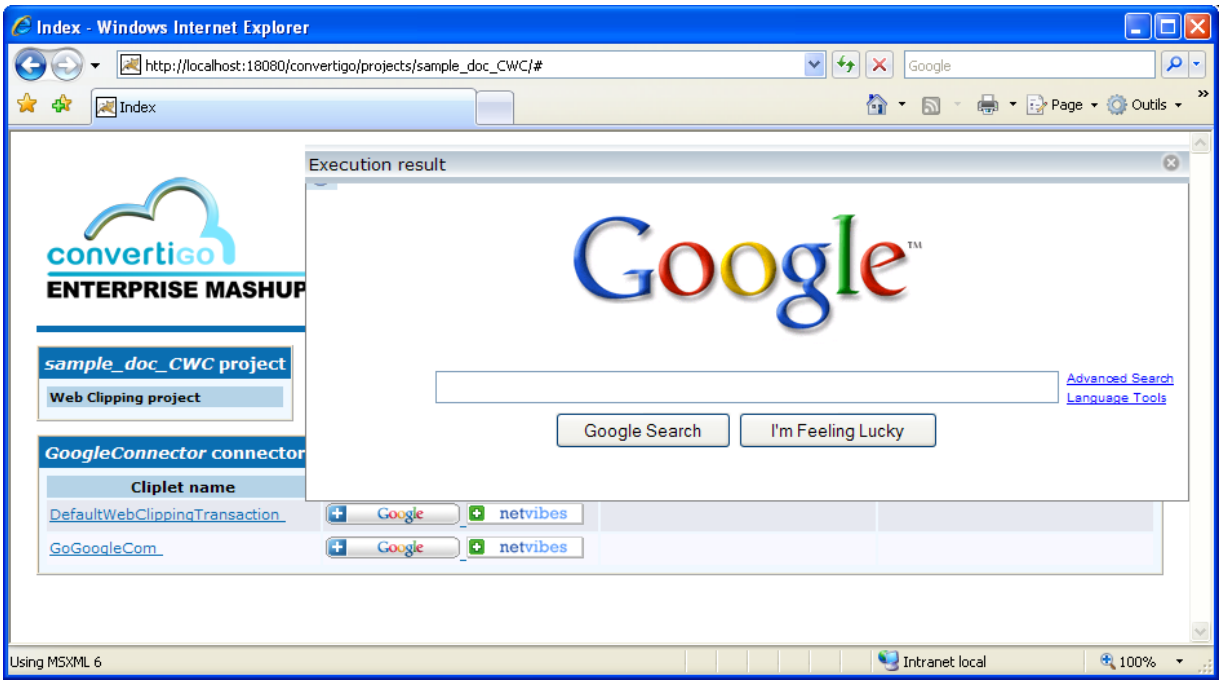


Figure 3 - 12: Cliplet in Execution result window of test platform with parent extractio enabled

- in the **XML** tab of the **Output** tab of the **Connector View**, through the adding of a `CENTER` tag in the code:

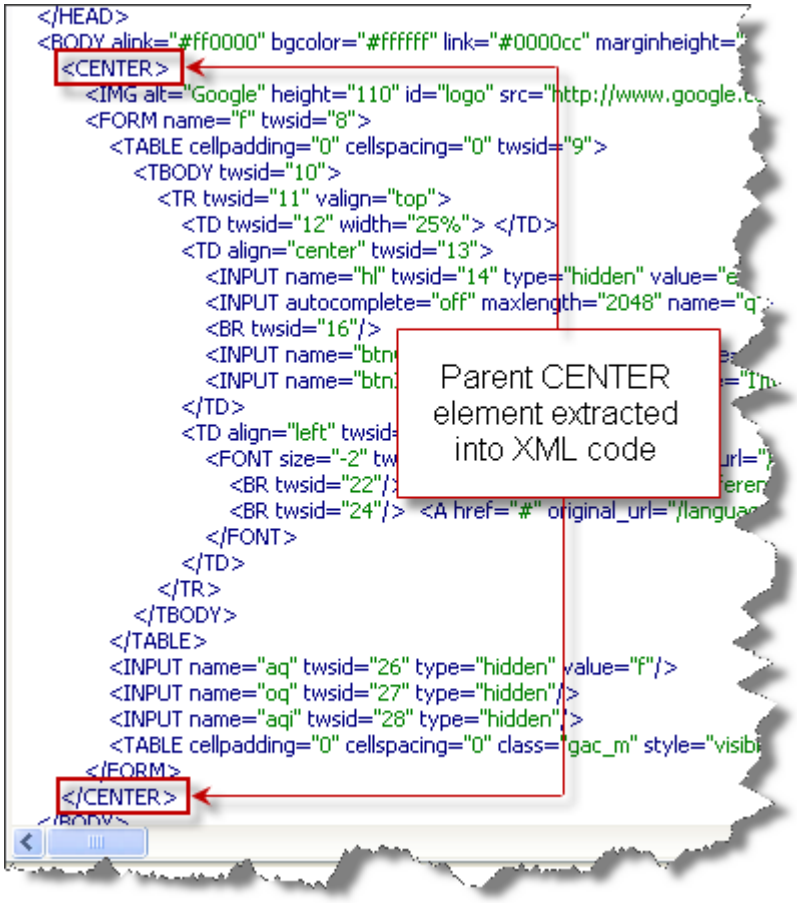


Figure 3 - 13: Center tag added to the code after parent extraction

NOTE ON THE DONOTHING STYLE SHEET

The `DoNothing` style sheet also includes a set of standard browser icons represented by a pull-down tab. This set of icons appears hidden in the upper left corner of the result cliplet:

Figure 3 - 14: Hidden set of icons

This set of icons is displayed when pointing the mouse cursor over it:

Figure 3 - 15: Set of icons displayed when pointing mouse cursor over it

The table below describes this set of icons:

Table 3 - 2: Execution result window icons

Icon	Description
	*Refresh* icon. Refreshes current window. Click on this icon to see if changes in the project have been correctly taken into account (for example after having modified the **Enable parent extraction from cliplet** property, see *"To enable the parent extraction of clipped elements"* on page 3-5).
	*Restart* icon. Restarts cliplet with its original transaction.
	*Backward* icon. Displays previous page in the cliplet history.
	*Forward* icon. Displays next page in the cliplet history.

From now on, the user can launch a Google search from the cliplet as if from a standard Google search page.

5 Type in `convertigo` in the search form of the **Execution result** window of the test platform.

6 Click on the **Google Search** button.

Manual actions (inputing a keyword and clicking on a button) performed in the cliplet are reproduced by Convertigo in the Google search engine. The keyword entered in the search is read and sent to the Google search engine, which processes it.

Once results are returned by the Google search engine, the `GoogleResultsPage` screen class is detected and associated extraction rules (`WebClipResults` and `WebClipNavigationBar`) are executed.

Results are clipped together with the navigation bar and a new cliplet is generated based on these elements:

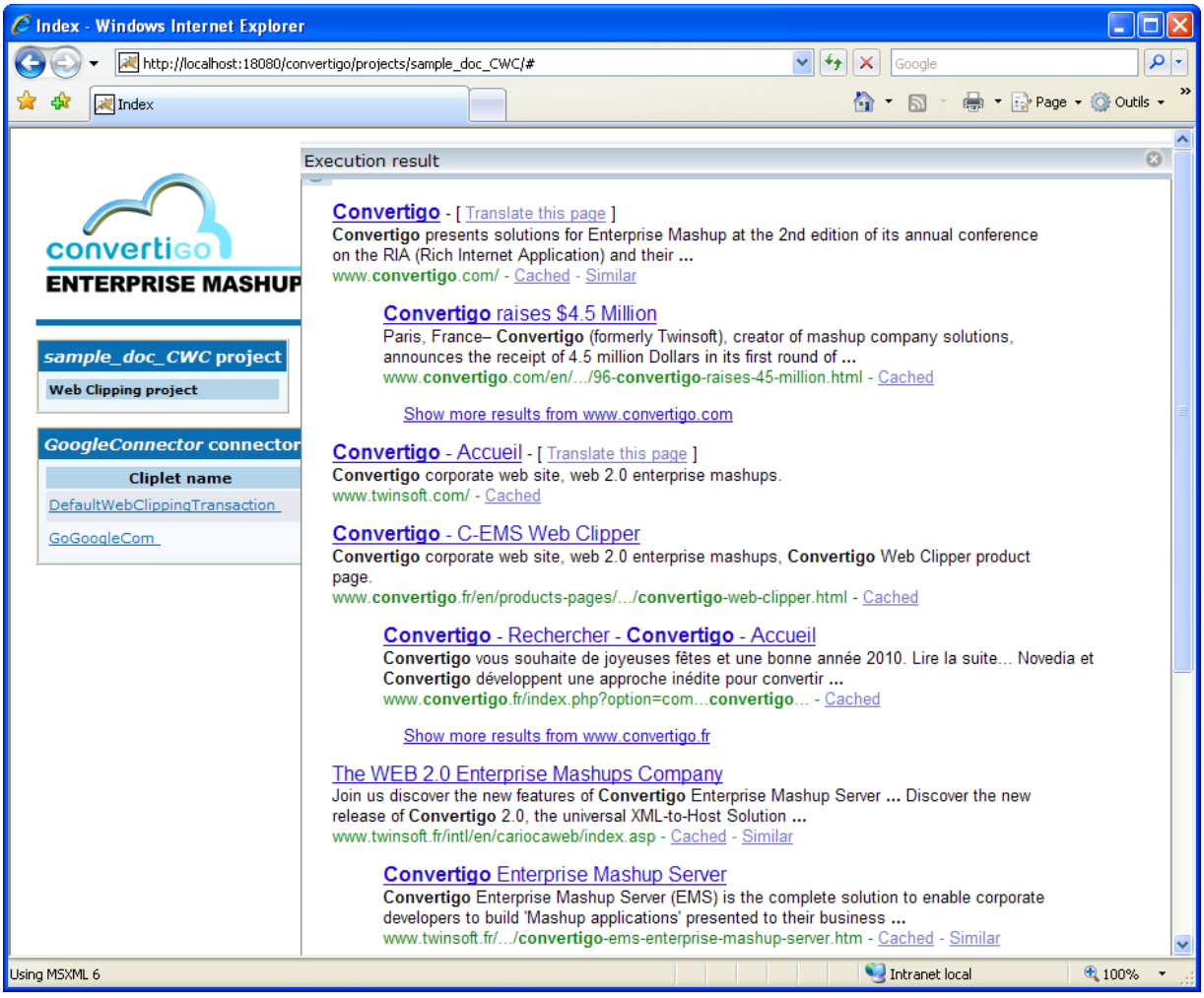


Figure 3 - 16: Google results in updated cliplet

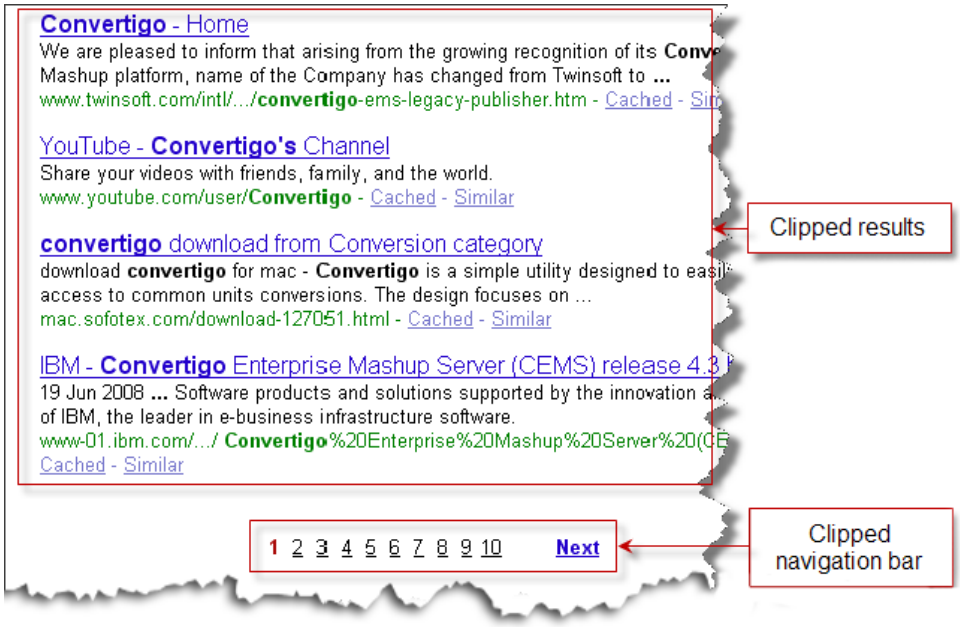


Figure 3 - 17: Execution window - Clipped results and navigation bar

*The navigation bar displayed in the **Execution result** window of the test platform is a lighter version of the Google navigation bar. It does not contain images, which are represented by specific elements not selected by the XPath generated for the navigation bar Web Clipper extraction rule.*

The user can navigate through results as if in a standard Google page.

7 Click on **Next**.

A second results page is displayed:

Figure 3 - 18: Execution result window - second result page

8 Click on a link.

The corresponding Web site opens.

The accessed Web page is detected by CWC as a `GenericWebPage` screen class, since no criterion (other than the one defined for the `GenericWebPage` screen class, see *"Criterion"* on page 1-6) matches.

The user can navigate freely through this Web site but throughout Convertigo clipping the whole content of each page, since specific Web Clipper extraction rules are defined only for Google search and result pages. The user has lost access to the Google results page.

The only ways a user can go back to results page are by using unconvenient processes, such as:

- restarting the cliplet, re-launching the search and navigating to the page, or
- going backward in the cliplet history as many time as required to access the results page.

WHAT COMES NEXT?

What could be interesting would be to open target Web sites in new tabs of the browser. The user would then be able to visit Web sites while keeping Google results accessible in a tab and without accessing them troughout Convertigo clipping. Accessed pages would then open faster.

To do so, we need to create a new XSL style sheet and edit it. .

3.2 Editing an XSL Style Sheet

As mentioned before, the standard style sheet used for web clipping XSL transformation is the `DoNothing` style sheet (see *"The `DoNothing` Style Sheet"* on page 3-1). This style sheet is a Convertigo object, *physically* represented in the **Project Explorer** by the `donothing.xsl` file:

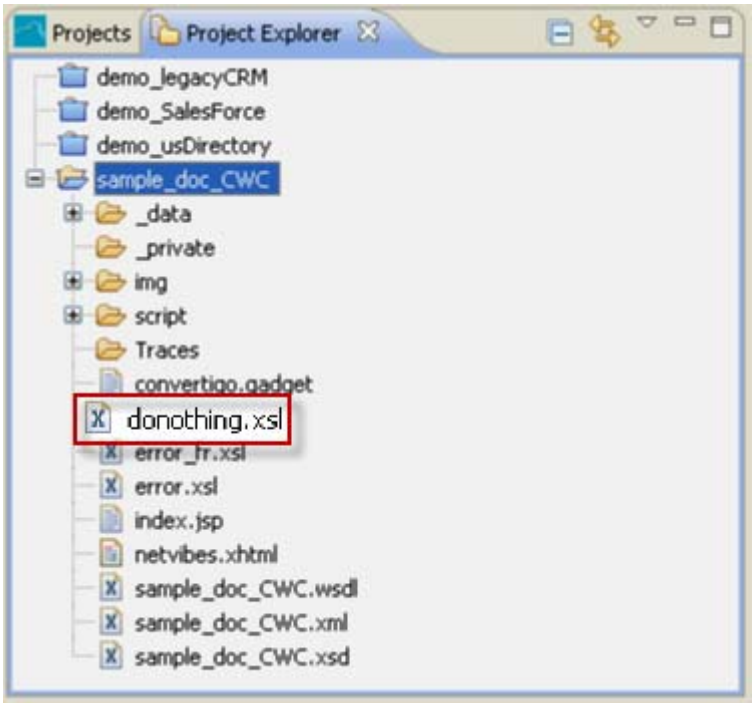


Figure 3 - 19: donothing.xsl file in Project Explorer

Should the user have any specific need in terms of data presentation (for example, if he wants open Google results to open in new tabs, see *"To create and edit a new XSL file"* on page 3-13), a specific style sheet in XSL format can be defined. The user can edit it to meet his/her needs, then associate it to the relevant screen class.

In our case, we want Web sites accessed from the Google results page to open in new tabs. The style sheet we want to create and edit should therefore be associated with the `GoogleResultsPage` screen class.

In CWC, creating a new style sheet and editing it means:

1. *Creating* a **Sheet** object in the **Projects View**.
2. *Creating* the corresponding XSL file in the **Project Explorer**.
3. *Editing* the XSL structure of the file in an **Editor**.

The following procedure describes how to implement these three steps.

To create and edit a new XSL file

We will first create in the **Sheets** folder of the `GoogleResultsPage` screen class a new **Sheet** object called `results`. The `results` stylesheet should be an edited copy of the `DoNothing` stylesheet, so that it can keep the same functionnalities and automatic processes.

To this end, we will duplicate the `donothing.xsl` file and rename it `results.xsl`. The `results.xsl` will eventually be edited to reflect our need (open Web sites in new tabs).

1. In the **Projects View**, right-click on the screen class (`GoogleResultsPage`) or on the **Sheets** folder of `GoogleResultsPage` screen class.
2. Select ***New > Sheet***.

   A **New Sheet** wizard opens:

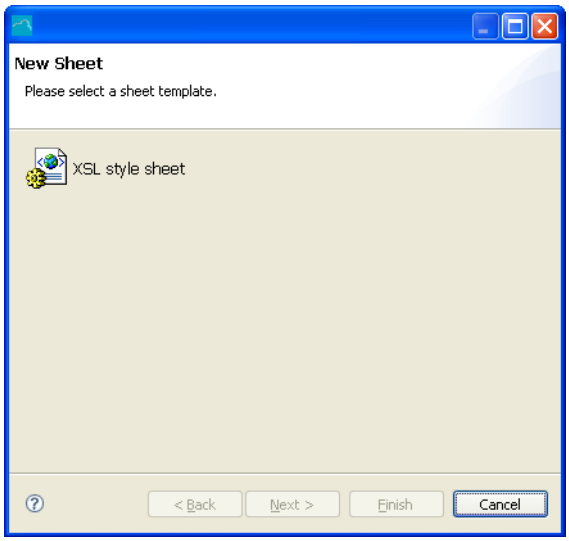


Figure 3 - 20: New Sheet wizard

3. Click on **XSL style sheet**.
4. Click on **Next**:

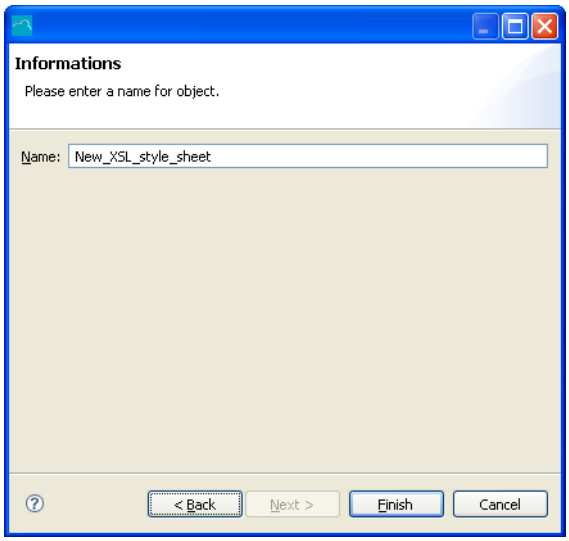


Figure 3 - 21: Entering a name for the new style sheet

5 Enter a name in the **Name** field (for example `results`).

6 Click on **Finish**.

In the screen class **Sheets** folder, the inherited `DoNothing` style sheet object is replaced by the newly created style sheet, `results`:

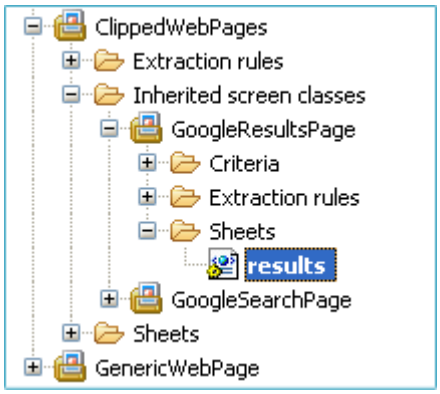


Figure 3 - 22: New results style sheet in Projects View

Now that the style sheet object has been created, we need to:

- *Specify* to which XSL file the `results` style sheet refers, through the style sheet **URL** property.
- *Create* the corresponding XSL file in the **Project Explorer**, `results.xsl`.

7 Select the `results` style sheet with a left-click. The **Properties View** is updated:

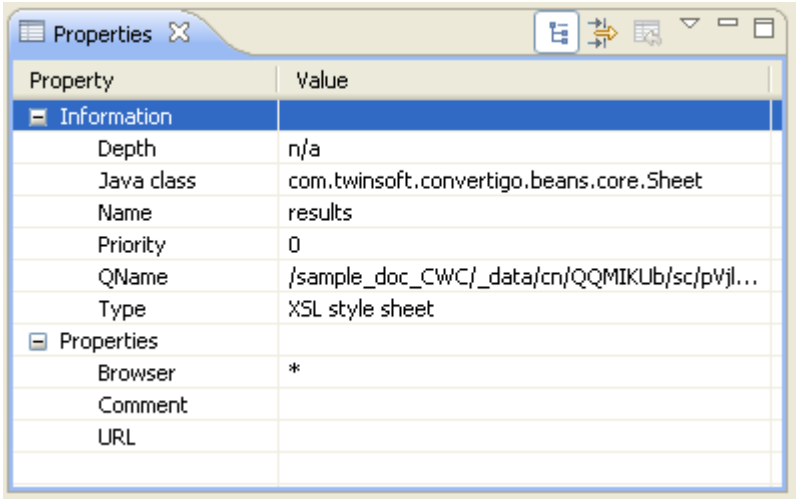


Figure 3 - 23: Style sheet properties

8 Click on the empty value of the **URL** property.

The field is highlighted then a cursor appears in the empty field:

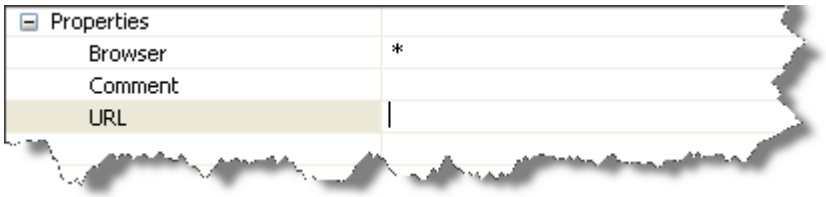


Figure 3 - 24: URL style sheet property

9 Type in `results.xsl`.

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

The `results.xsl` doest not exist in the files of the project yet. We will create it in the **Projects Explorer**.

11 Click on the **Project Explorer** tab.

The project files tree structure appears:

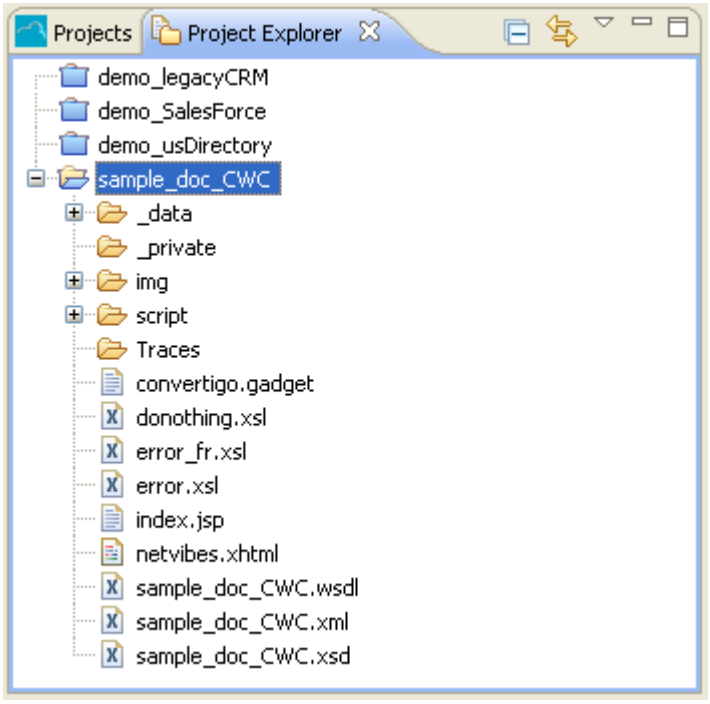


Figure 3 - 25: Project tree structure in Project Explorer

To create the `results.xsl` file, we will duplicate the `donothing.xsl` style sheet because it initially processes the XML output of any Web clipping transaction.

12 Select `donothing.xsl` with a left-click.

13 Copy-paste the file by pressing `Ctrl+C` then `Ctrl+V` or right-click on the `donothing.xsl` file, then select **Copy**, and right-click on the `sample_doc_CWC` folder, then select **Paste**.

14 In the **Name conflict** window that opens, type in the name of the new style sheet: `results.xsl`:

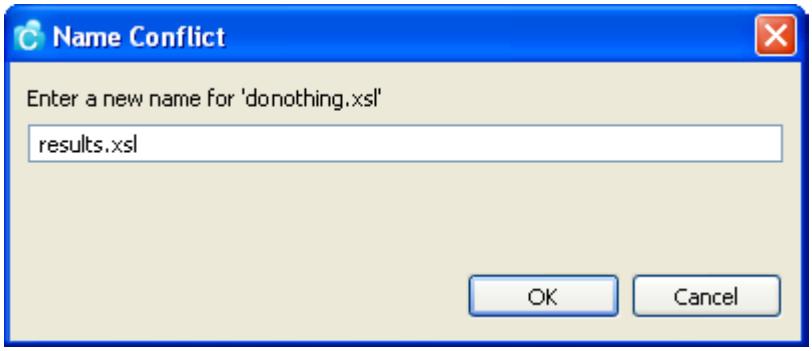


Figure 3 - 26: Name conflict window

The `results.xsl` file is now created in the **Project Explorer**:

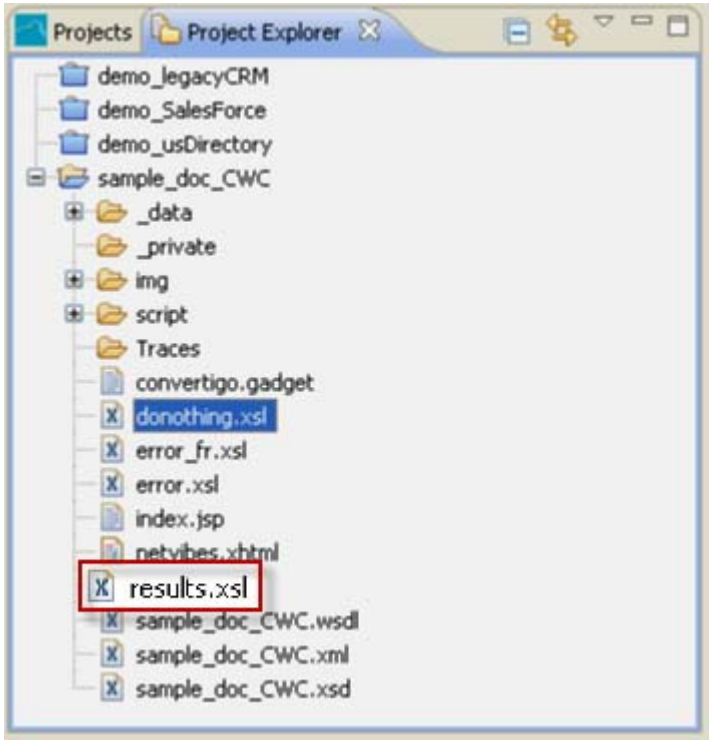


Figure 3 - 27: New style sheet in Project Explorer

We will now edit this XSL file.

15 Double-click on `results.xsl`.

*To edit a style sheet, you can also right-click on it in the **Projects View** and select **Edit**.*

The file opens in an Eclipse XML **Editor**:

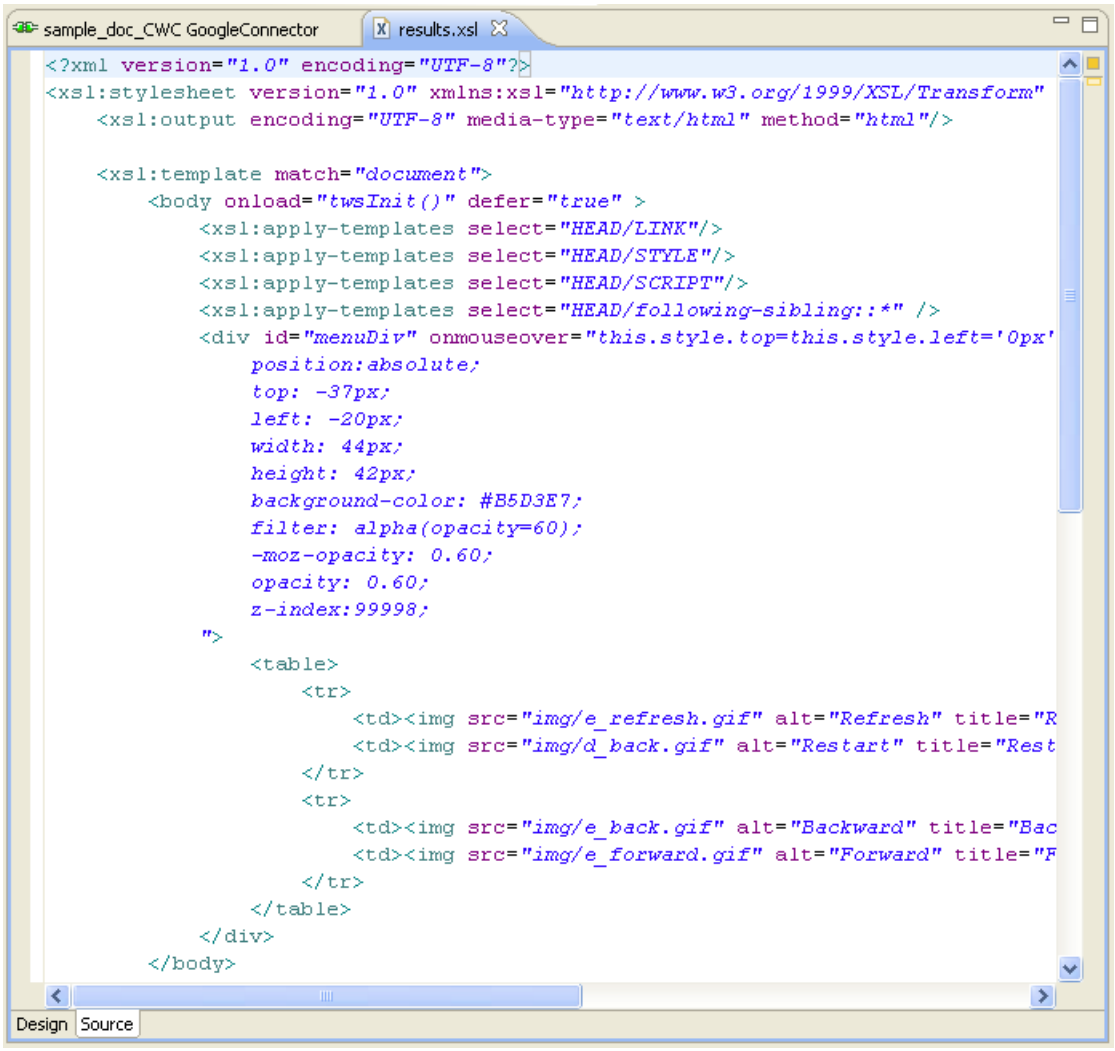


Figure 3 - 28: results.xsl style sheet in Editor

The file includes a template rule which matches on any node and attribute and copies - using the `<xsl:copy>` transformation element - these nodes and attributes in the HTML output:

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

Figure 3 - 29: Template rule matching on any node

The XML resulting from the transaction being already in HTML format, nodes can be recopied as they are without generating specific HTML content.

We will first copy this template rule, then modify it so that it matches on the XPath corresponding to result links. It is indeed on these links that the click we want to modify is performed. It is modelized in HTML by the A element.

16 Copy the template rule by highlighting it and pressing `Ctrl+C`.

17 Place the cursor anywhere in the file before the ending `</xsl:stylesheet>` tag and outside template rules.

18 Paste the template rule by pressing `Ctrl`+V.

19 Display the **Connector View** by selecting the `sample_doc_CWC GoogleConnector` tab.

20 Assuming that the **Connector View** currently displays Google results, right-click on a link:

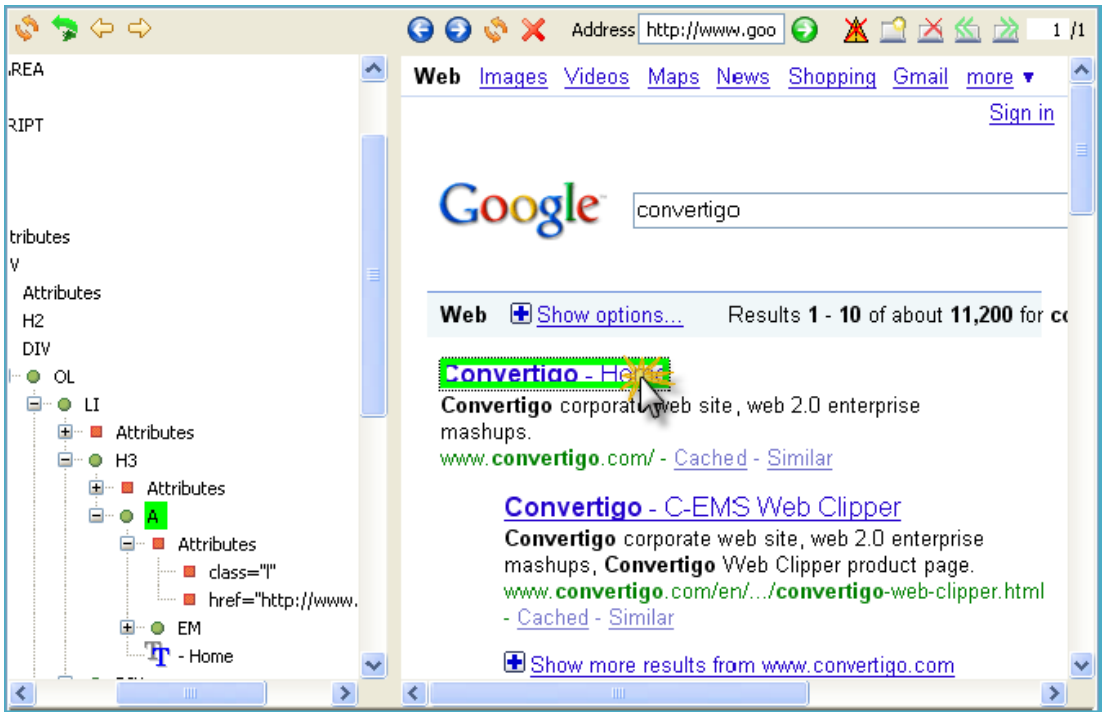


Figure 3 - 30: Selecting an HTML element in the Connector View (result link)

The link is outlined in green in the **Connector View** and the corresponding node (A element) is highlighted in green in the **DOM Tree**.

*Other elements than the A element can be highlighted in the **DOM Tree** depending on where you click exactly in the link. We are generating the Xpath of the element that we want to modify: the A HTML element. If this element is not highlighted in green, go up or down the **DOM Tree** and select it.*

The A element node contains a `class` attribute. We will generate the XPath from this attribute.

21 Right-click the attribute node `class="l"` and select **Generate selection XPath (Enter)** or select the attribute node with a left-click and press **Enter**.

Figure 3 - 31: Selection of node and generation of XPath

The XPath expression of the selected node appears in the **XPath Evaluator**:

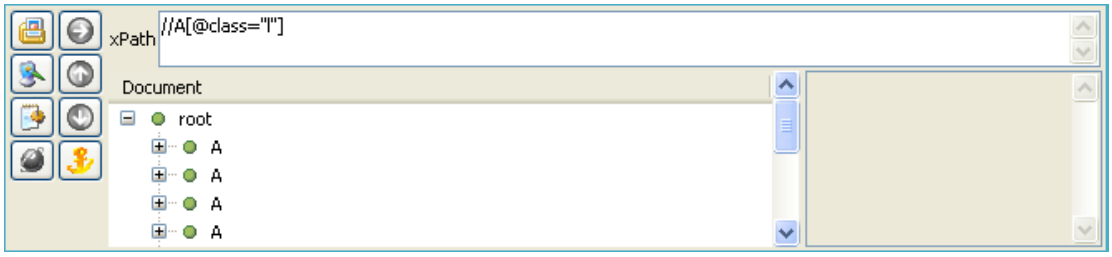


Figure 3 - 32: XPath expression of selected node

The **Document** window shows that the generated XPath matches all A elements of the result lists, which correspond to result links. This XPath can therefore be used as value of the `match` attribute in our template rule.

22 Copy the part of the XPath that will be used as `match` value: `A[@class="l"]`.

23 Click on the `results.xsl` tab in the **Connector View**.

24 Paste it as `match` attribute value of the pasted template rule in the `results.xsl` file (think about replacing the **"** characters by **'**):

```
<xsl:template match="A[@class='l']">
      <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
    </xsl:template>
```

Figure 3 - 33: Pasting the XPath value as match value in XSL file

The element to be processed by the XSL style sheet (result links represented by their XPath expression) has been defined. We will now edit the template rule so that when clicking on a link, the target Web page opens in a new tab. This is done by using:

- the `target` attribute (added through the `<xsl:attribute>` element) with the value `_blank` (new tab);
- the `href` attribute (added through the `<xsl:attribute>` element), the value of which is set to the *original* link value (and not to convertigo server and its clipping process) thanks to data extracted by the Web clipper extraction rule and using the `<xsl:value-of>` element with the attribute `select` set at `"@original_url"`;
- the `<xsl:apply-templates>` element used to spread the copying process to

following HTML nodes.

For more information on XSL syntax, see the XSLT tutorial from W3 schools at http://www.w3schools.com/Xsl/

After editing, the template rule is the following:

```
<xsl:template match="A[@class='l']">
  <xsl:copy>
    <xsl:attribute name="target">_blank</xsl:attribute>
    <xsl:attribute name="href"><xsl:value-of select="@original_url"/></xsl:attribute>
    <xsl:apply-templates select="@*[name(.)!='href' and name(.)!='twsid']|node()"/>
  </xsl:copy>
</xsl:template>
```

Figure 3 - 34: Final template rule matching on links

We will now re-launch the transaction and check that, when clicking on a link, the target Web sites open in a new tab with the correct URL address.

To test that the newly created style sheet returns expected results

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

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/projects/
<ProjectName>/index.jsp?__transaction=
<TransactionName>
```

For example, in our project:

- *HostName* = `localhost`;
- *ConvertigoPortNumber* = `18080`;
- *ProjectName* = `sample_doc_CWC`;
- *TransactionName* = `GoGoogleCom`;

3. For example, in a browser URL address bar:

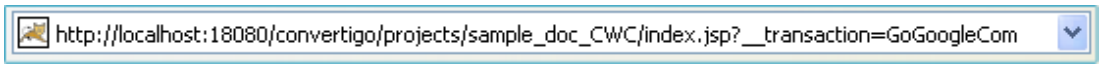


Figure 3 - 35: Example of URL syntax for calling a transaction

The transaction is executed. Clipped elements appear in the browser as a cliplet:

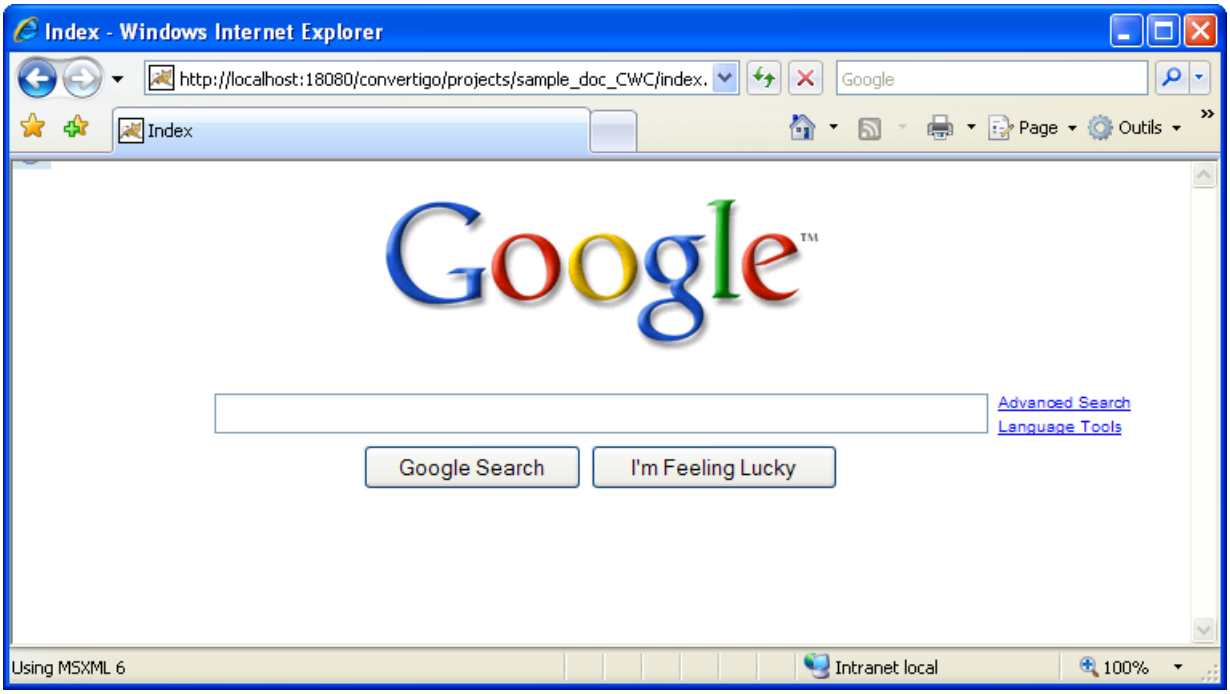


Figure 3 - 36: Cliplet based on Google logo and search form

4 Type in a keyword in the search field (for example `convertigo`).

5 Launch the Google search by clicking on the **Google Search** button.

Results are displayed together with the navigation bar:

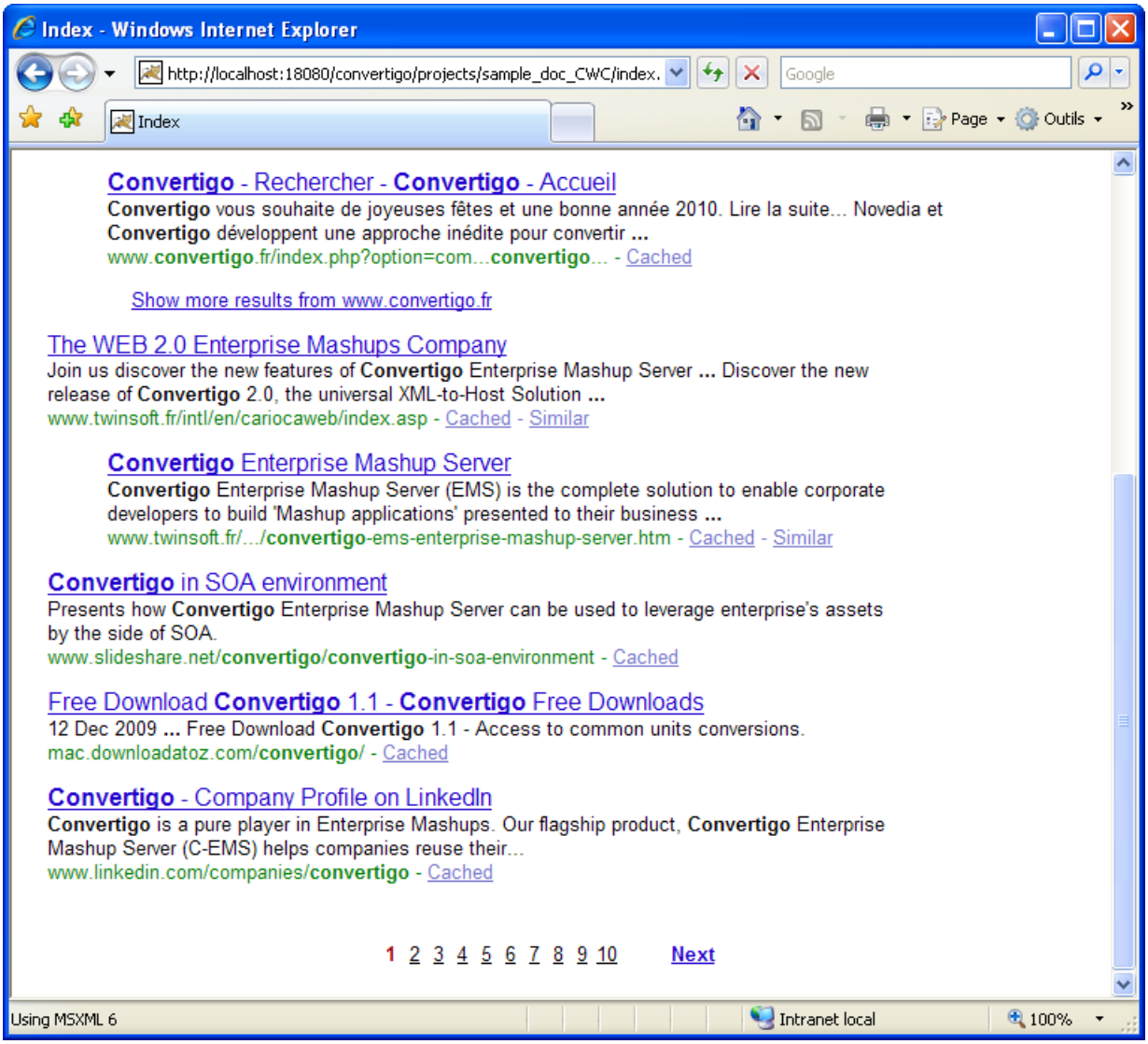


Figure 3 - 37: Cliplet based on results and navigation bar

6 Click on a link:

Figure 3 - 38: Clicking on results page hypertext link

The target Web site appears in a new tab or a new window (depending on the browser):

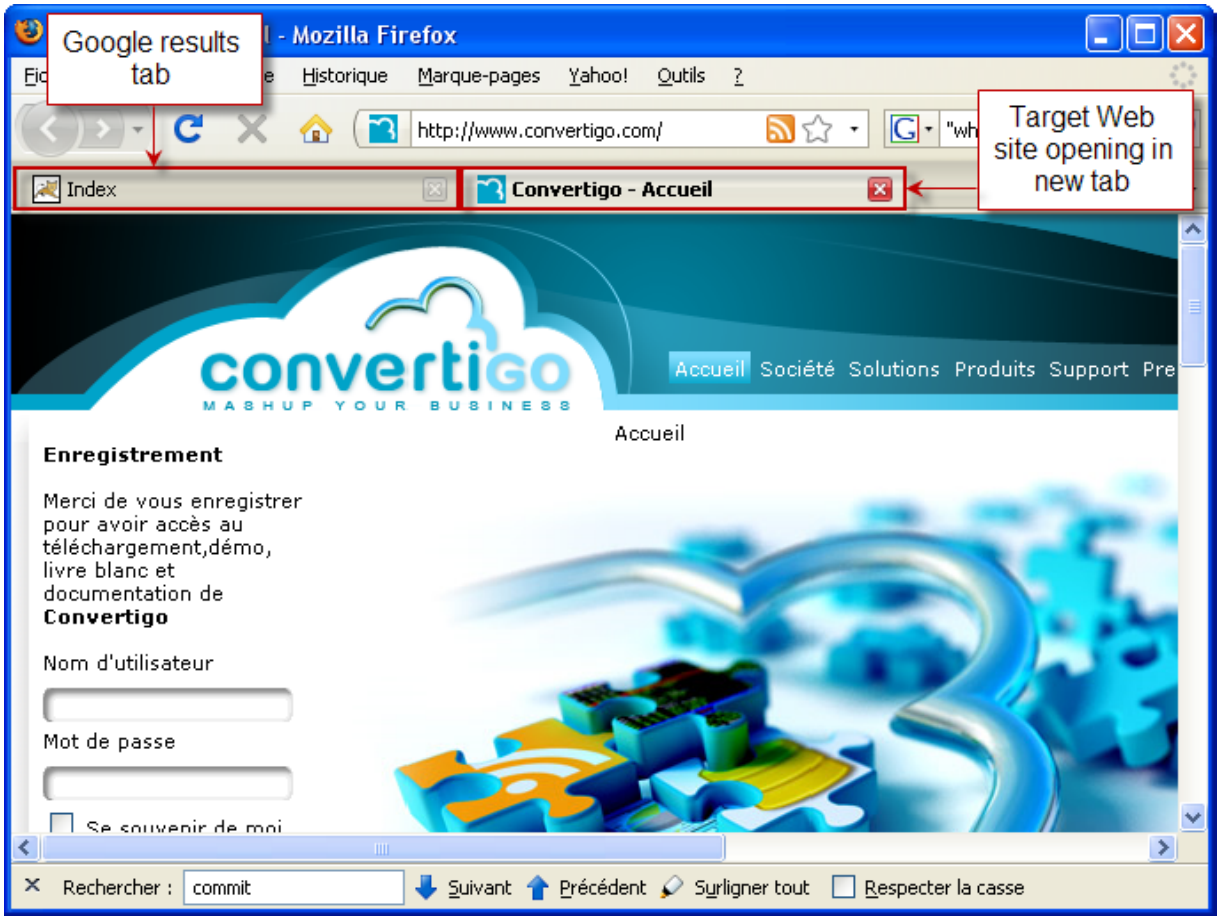


Figure 3 - 39: Target Web site displayed in new tab

Figure 3 - 40: Target Web site displayed in new window