convertigo
MASHUP YOUR BUSINESS

Starting with Convertigo Legacy Publisher

Quick Guide

CEMS 5.2.0

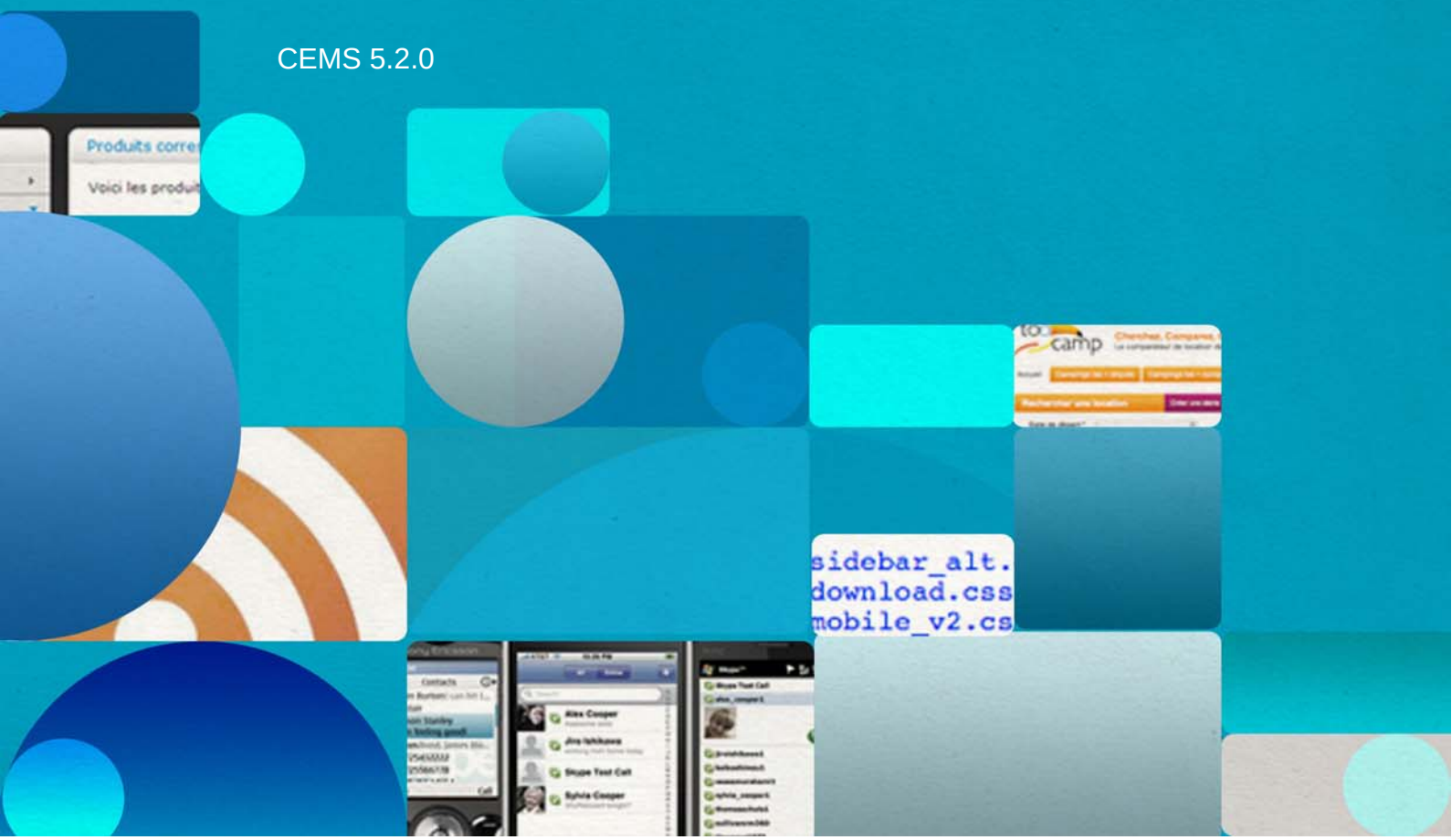


TABLE OF CONTENTS

Getting Familiar with Convertigo Legacy Publisher 1

CLP Purpose 1-1

CLP Concepts and Objects 1-2

General Concept..... 1-2

Block Factory 1-2

CLP Objects..... 1-3

Definitions 1-4

Connector..... 1-4

Screen Class..... 1-5

Transaction 1-12

CLP Studio..... 1-14

General Presentation..... 1-14

View Description 1-15

Projects View 1-16

Properties View 1-17

Connector View / Document Editor 1-21

My First Convertigo Legacy Publisher Project 2

Legacy Publishing Project 2-1

Project Description..... 2-1

Opening the Sample Project from the Studio 2-2

Setting up the project..... 2-3

Creating a Project and Setting Connector Parameters..... 2-3
Dynamically webizing legacy application screens 2-7
Customizing Webization 2-12
Customizing the Webization Page Template..... 2-12
Creating Extraction Rules on the Default Screen Class 2-22
Editing the Generic Webization CSS 2-28
Defining Screen Classes and Screen Class-Specific Extraction Rules 2-32
Introduction 2-33
Creating Extraction Rules on Properly Webized Screen..... 2-33
Creating Extraction Rules on Improperly Webized Screens 2-43

LIST OF FIGURES

Figure 1 - 1: CLP refaces "green screens" into modern graphical user interfaces 1-1

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

Figure 1 - 3: CLP objects 1-4

Figure 1 - 4: Connectors folder 1-5

Figure 1 - 5: Screen classes folder 1-5

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

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

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

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

Figure 1 - 10: Default extraction rules 1-8

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

Figure 1 - 12: Extracted data in XML 1-10

Figure 1 - 13: Default style sheets 1-10

Figure 1 - 14: Date style sheet - <xsl:template> element 1-11

Figure 1 - 15: Generic Webization CSS and XSL style sheet 1-12

Figure 1 - 16: Transactions folder 1-13

Figure 1 - 17: Screen class handler triggered on detection of the LoginScreen screen class .. 1-14

Figure 1 - 18: Screen class handler in transaction Functions folder 1-14

Figure 1 - 19: CLP Studio 1-15

Figure 1 - 20: Formatting of unsaved, inherited and disabled objects in the Projects View 1-17

Figure 1 - 21: Properties View 1-18

Figure 1 - 22: Text property field edition 1-18
Figure 1 - 23: Javascript expression field edition 1-19
Figure 1 - 24: Text property edition via edition window 1-19
Figure 1 - 25: Javascript expression window 1-19
Figure 1 - 26: Edition window 1-20
Figure 1 - 27: Configuration window 1-20
Figure 1 - 28: Text property edition via drop-down menu 1-21
Figure 1 - 29: Selecting a value in the drop-down menu 1-21
Figure 1 - 30: Connector View (Output tab selected by default) 1-22
Figure 1 - 31: Example legacy application screen 1-25
Figure 1 - 32: Block structure of the example legacy application screen 1-25
Figure 1 - 33: Dynamic Webization of legacy application screen 1-26
Figure 1 - 34: XML output after execution of extraction rules 1-26
Figure 1 - 35: Static or field XML type attribute of extracted blocks 1-27
Figure 1 - 36: Style definitions applying to "static" blocks of example legacy screen 1-28
Figure 1 - 37: Style definitions applying to "field" blocks of example legacy screen 1-28
Figure 2 - 1: Launching the New Project wizard 2-2
Figure 2 - 2: Selecting the CLP sample project 2-2
Figure 2 - 3: Sample project in Projects View 2-3
Figure 2 - 4: New Project Wizard 2-3
Figure 2 - 5: Selecting a project type 2-4
Figure 2 - 6: Entering a project name 2-4
Figure 2 - 7: Entering a connector name 2-5
Figure 2 - 8: Setting connection parameters 2-5
Figure 2 - 9: New project summary window 2-6
Figure 2 - 10: sample_doc_CLP project appearing in the Projects View 2-6
Figure 2 - 11: Emulator displaying legacy application screen 2-7
Figure 2 - 12: Generating XML of legacy application screen 2-8
Figure 2 - 13: Generated XML and Last detected screen class 2-9

Figure 2 - 14: Webized legacy application screen 2-9
Figure 2 - 15: Second webized legacy application screen 2-10
Figure 2 - 16: Legacy screen webized using default extraction rules and templates 2-11
Figure 2 - 17: index.jsp file 2-13
Figure 2 - 18: variables.xsl file 2-13
Figure 2 - 19: sample_doc_CLP project files tree structure 2-14
Figure 2 - 20: Webization template files in Project Explorer 2-15
Figure 2 - 21: Studio Save icon 2-16
Figure 2 - 22: Image added in webization page template 2-16
Figure 2 - 23: Viewport positioning variables 2-17
Figure 2 - 24: Webized screen with changed viewport position 2-18
Figure 2 - 25: GroupActionButton variable 2-19
Figure 2 - 26: Selecting connector tab 2-19
Figure 2 - 27: Legacy screen with action buttons 2-20
Figure 2 - 28: Webized screen with grouped action buttons 2-20
Figure 2 - 29: Webized screen with properly layed out grouped buttons and logo 2-21
Figure 2 - 30: Creating a new extraction rule 2-22
Figure 2 - 31: New Extraction Rule wizard - Image 2-23
Figure 2 - 32: New extraction rule name - AddImage 2-23
Figure 2 - 33: New extraction rule (AddImage) in default screen class Extraction rules folder 2-24
Figure 2 - 34: AddImage extraction rule properties 2-24
Figure 2 - 35: Image layout property 2-25
Figure 2 - 36: Screen zone window 2-25
Figure 2 - 37: Selecting the image layout in the Emulator 2-26
Figure 2 - 38: Linking selection to Image layout property 2-26
Figure 2 - 39: Automatic setting of Image layout property value 2-26
Figure 2 - 40: Browser output after execution of default and AddImage extraction rules 2-27
Figure 2 - 41: sample_doc_CLP project files tree structure 2-28
Figure 2 - 42: Expanded css folder 2-29

Figure 2 - 43: Generic Webization cascading style sheet (sna.css) 2-29
Figure 2 - 44: Style definition - Static text outside tables 2-30
Figure 2 - 45: Editing font size (1) 2-30
Figure 2 - 46: Editing font size (2) 2-30
Figure 2 - 47: Studio Save icon 2-30
Figure 2 - 48: Style definition - Foreground color styles for static text 2-31
Figure 2 - 49: Editing foreground color (1) 2-31
Figure 2 - 50: Editing foreground color (2) 2-31
Figure 2 - 51: Webized legacy application screen after page template and CSS customization 2-32
Figure 2 - 52: "Ouverture" text string (detection criterion of LoginScreen screen class) 2-34
Figure 2 - 53: Login screen of emulated legacy application 2-34
Figure 2 - 54: Selecting a detection criterion on screen 2-35
Figure 2 - 55: Newly created screen class and associated detection criterion 2-35
Figure 2 - 56: Renaming a screen class 2-36
Figure 2 - 57: Inherited and new elements 2-37
Figure 2 - 58: Clicking Go to current screen class icon 2-37
Figure 2 - 59: Highlighted screen class after Go to current screen class object command 2-38
Figure 2 - 60: Creating new extraction rule 2-39
Figure 2 - 61: New Extraction Rule wizard - Removing of blocks 2-39
Figure 2 - 62: New extraction rule name - DeleteTopBlocks 2-40
Figure 2 - 63: New extraction rule (DeleteTopBlocks) in screen class Extraction rules folder . 2-40
Figure 2 - 64: DeleteTopBlocks extraction rule properties 2-41
Figure 2 - 65: Screen zone property 2-41
Figure 2 - 66: Screen zone window 2-41
Figure 2 - 67: Selecting the screen zone in Emulator 2-42
Figure 2 - 68: Linking selection to Screen zone property 2-42
Figure 2 - 69: Automatic setting of Screen zone property value 2-42
Figure 2 - 70: Browser output after execution of default and DeleteTopBlocks extraction rules 2-43

Figure 2 - 71: Customer list screen 2-43
Figure 2 - 72: Webized screen with improperly webized table 2-44
Figure 2 - 73: "TIV010" text string (detection criterion of CustomerListScreen screen class) .. 2-45
Figure 2 - 74: Selecting a detection criterion on screen 2-45
Figure 2 - 75: Inherited elements 2-46
Figure 2 - 76: Highlighted screen class after Go to current screen class object command 2-46
Figure 2 - 77: New Extraction Rule wizard - Table 2-48
Figure 2 - 78: New extraction rule name - ExtractCustomers 2-48
Figure 2 - 79: New extraction rule (ExtractCustomers) in screen class Extraction rules folder 2-49
Figure 2 - 80: ExtractCustomers extraction rule properties 2-50
Figure 2 - 81: Screen zone property 2-50
Figure 2 - 82: Screen zone window 2-51
Figure 2 - 83: Selecting the screen zone in Emulator 2-51
Figure 2 - 84: Linking selection to Screen zone property 2-51
Figure 2 - 85: Automatic setting of Screen zone property value 2-52
Figure 2 - 86: Table columns to be set 2-52
Figure 2 - 87: Columns property 2-52
Figure 2 - 88: Columns table window 2-53
Figure 2 - 89: First column setting 2-53
Figure 2 - 90: First column set 2-54
Figure 2 - 91: Columns property updated 2-54
Figure 2 - 92: First column setting in column definition table 2-54
Figure 2 - 93: Settings of table columns 2-55
Figure 2 - 94: Webized customer list screen 2-56
Figure 2 - 95: Sorting columns 2-56
Figure 2 - 96: Table title line webized as static text 2-57
Figure 2 - 97: Table extraction rule Offset property 2-57
Figure 2 - 98: Webized table with offset value 2-58

LIST OF TABLES

Table 1 - 1: CLP objects 1-3

Table 1 - 2: Criterion types 1-7

Table 1 - 3: Extraction rule common properties 1-8

Table 1 - 4: Block standard XML attributes 1-9

Table 1 - 5: CLP views 1-15

Table 1 - 6: Standard Eclipse buttons 1-16

Table 1 - 7: Projects View tabs and icons 1-16

Table 1 - 8: Properties View icons 1-18

Table 1 - 9: Connector View parts 1-21

Table 1 - 10: Connector View icons 1-22

Table 2 - 1: Webization customization methods 2-11

Table 2 - 2: Image extraction rule properties 2-25

Table 2 - 3: `LoginScreen` screen class 2-33

Table 2 - 4: `DeleteTopBlocks` extraction rule 2-38

Table 2 - 5: *Removing of blocks* extraction rule properties 2-42

Table 2 - 6: `LoginScreen` screen class 2-44

Table 2 - 7: `ExtractCustomers` extraction rule 2-47

Table 2 - 8: *Table* extraction rule properties 2-52

Table 2 - 9: Columns definition table 2-53

1 Getting Familiar with Convertigo Legacy Publisher

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

- CLP Purpose
- CLP Concepts and Objects
- CLP Studio

1.1 CLP Purpose

The purpose of Convertigo Legacy Publisher (CLP) is to turn legacy systems (IBM, BULL mainframes, IBM iSeries or character based UNIX servers), often based on "green screen" user interfaces, into new Web 2.0 Ajax enabled Web applications, without modifying the code of original applications:

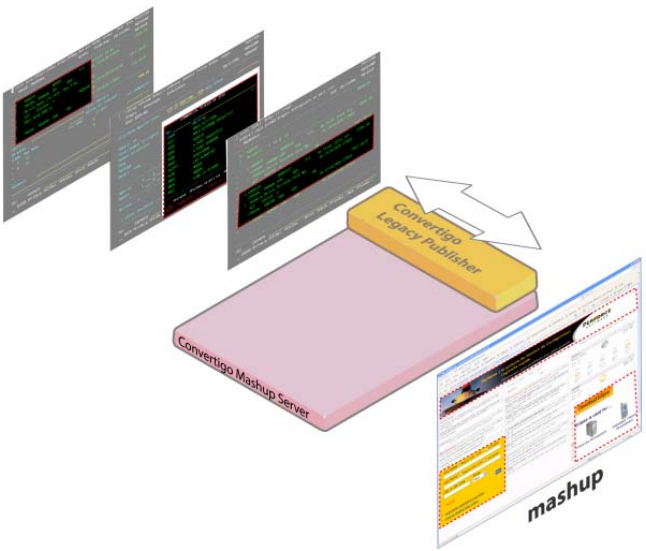


Figure 1 - 1: CLP refaces "green screens" into modern graphical user interfaces

CLP encompasses a large number of applications:

- *Modernizing* existing green screen applications into modern AJAX based Web GUI.
- *Mashing-up* existing legacy applications by linking them to other widgets using the

Convertigo Mashup Composer.

- *Integrating* legacy application in JSR-168 compliant portals using SSO (*Single Sign On*).
- *Automating* complex tasks that users have to manually execute on multiple screens provided by old-fashioned legacy applications.

1.2 CLP Concepts and Objects

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

- General Concept
- CLP Objects
- Definitions

1.2.1 General Concept

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

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

BLOCK FACTORY

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

1 The block factory analyses screen lines.

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

3 For each character string having same attributes, the block factory breaks sentences into single words by performing a separation on blank spaces. For example, the sentence *the_nice_hat* (where "_" represents a blank space) is divided into five blocks: *the*, _, *nice*, _ and *hat*. Consecutive blank spaces are grouped into one single block.

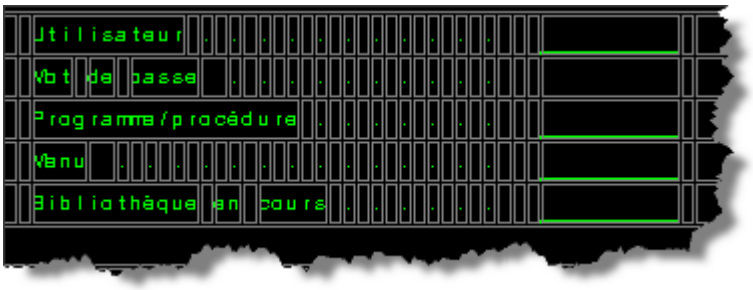


Figure 1 - 2: Block generation on legacy application screen (partial view of screen)

1.2.2 CLP Objects

All of the previously described concepts correspond to as many CLP objects, that are set as part of a Legacy publishing project.

The block factory is not set directly by the user, but is part of CLP projects.

The following table describes CLP objects:

Table 1 - 1: CLP objects

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

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

All of these objects are set when developing a Legacy publishing project. Once the project has been created, these objects appear in the **Projects View** of the CLP Studio (*"CLP Studio"* on page 1 - 14):

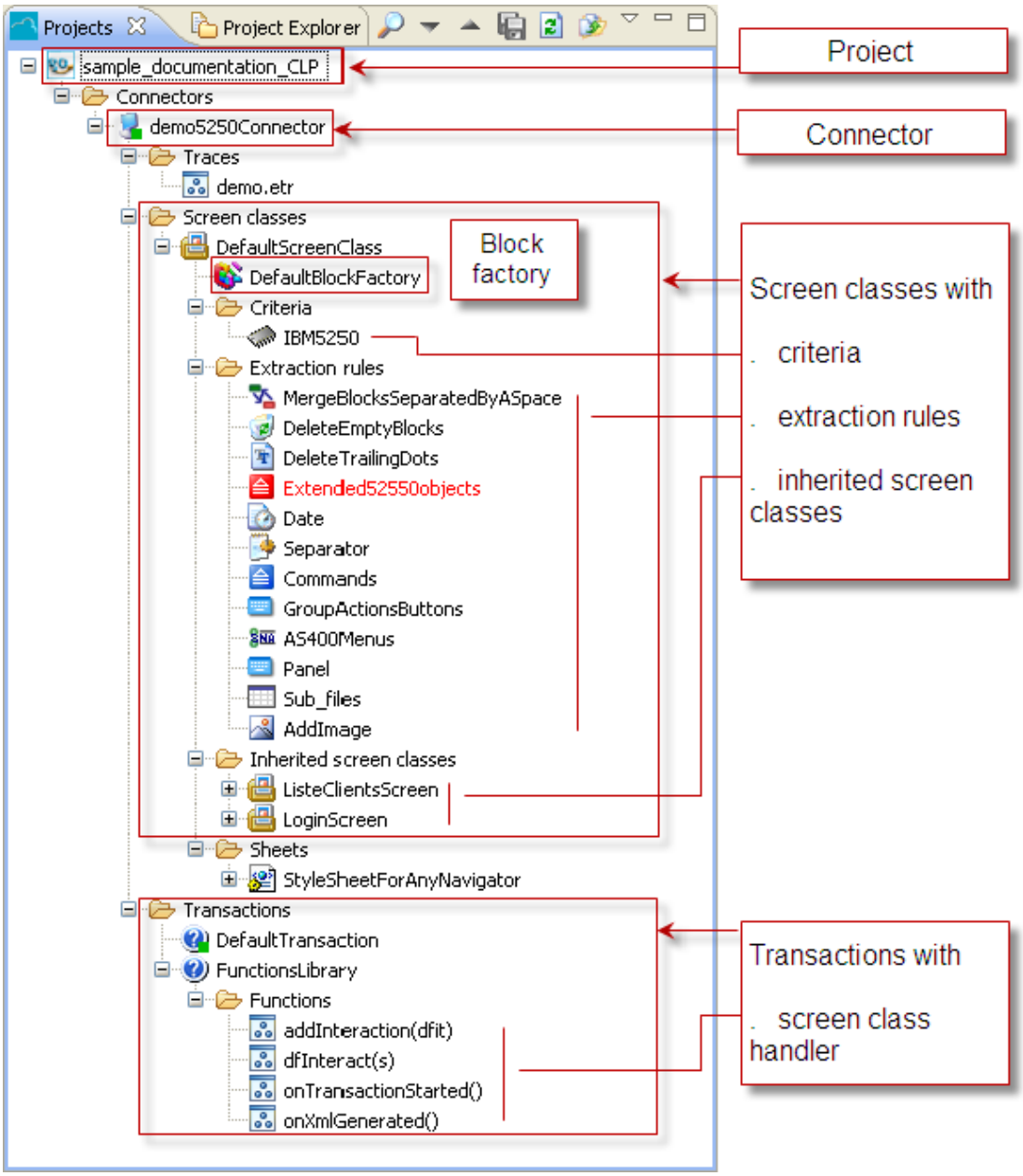


Figure 1 - 3: CLP objects

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

1.2.3 Definitions

CONNECTOR

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

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

Connector parameters and types are set in the **New Project** wizard which opens when

creating a Legacy publishing project (see *"Creating a Project and Setting Connector Parameters"* on page 2-3).

Once a new project has been created, the corresponding connector appears in the *Connectors* folder of the project, in the **Projects View** of the CLP Studio (*"CLP Studio"* on page 1 - 14):

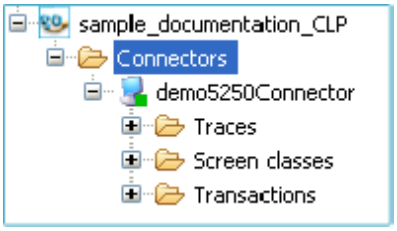


Figure 1 - 4: Connectors folder

Screen Class

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

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

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

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

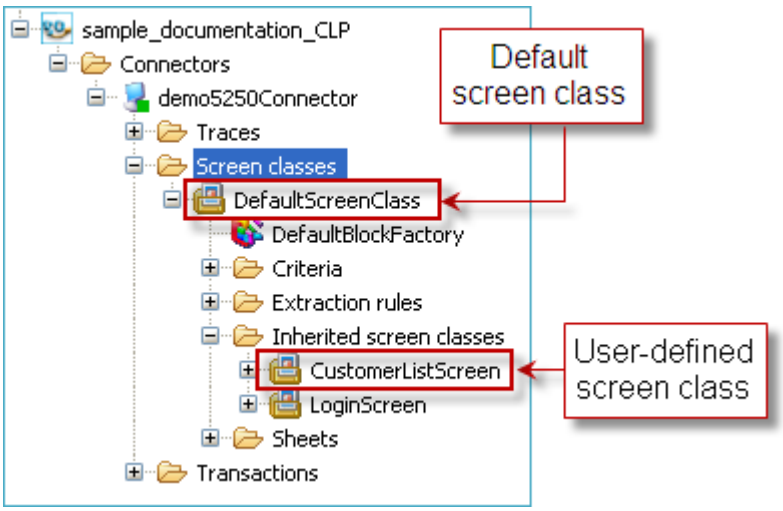


Figure 1 - 5: Screen classes folder

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

Inherited objects appear greyed (see Figure 1 - 20) in the **Projects View**:

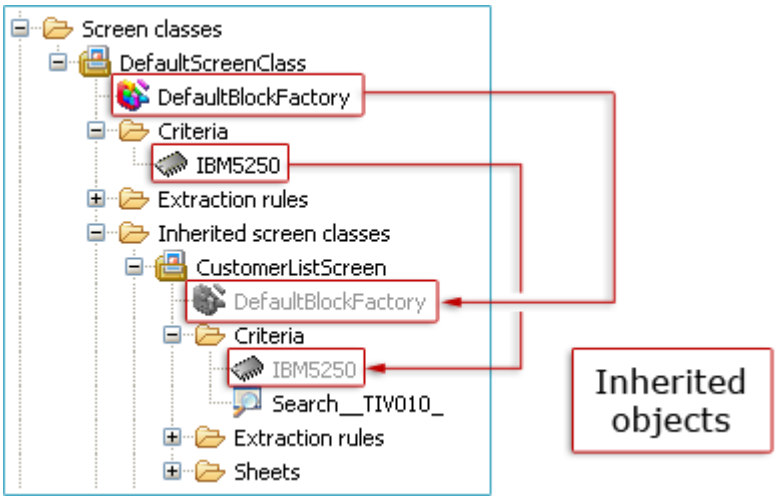


Figure 1 - 6: Objects inherited from the Default screen class

Inherited screen classes appear in subfolders named *Inherited screen classes* in the **Projects View** of the CLP Studio (*"CLP Studio"* on page 1 - 14):

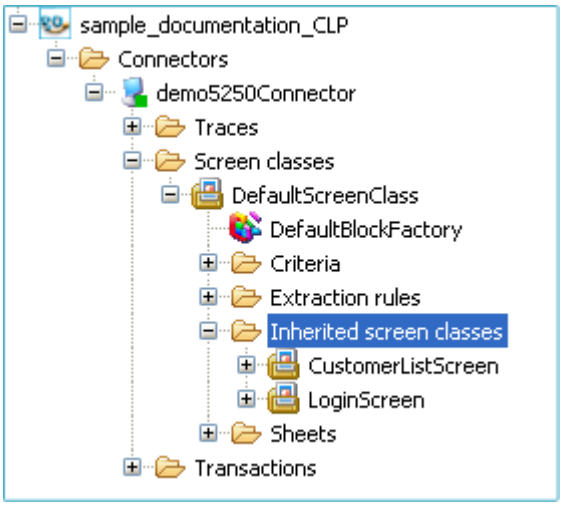


Figure 1 - 7: Screen classes and Inherited screen classes folders

CRITERION

A legacy application screen is considered belonging to a screen class when it matches all criteria defined by the user for this screen class. If no criterion matches, the screen is considered as belonging to the default screen class.

When creating a project (and defining a connector), a criterion is automatically created in the **Criteria** folder of the `DefaultScreenClass` screen class. This default criterion corresponds to the *emulator technology*. For example, if the connector set for the project emulates a connection to an IBM5250 terminal, an `IBM5250` criterion is created by default (see Figure 1 - 6).

Since the default screen class must match all unidentified screens, it is highly recommended not to add any extra criterion to this screen class.

Three other types of criteria can be defined in CLP:

Table 1 - 2: Criterion types

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

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

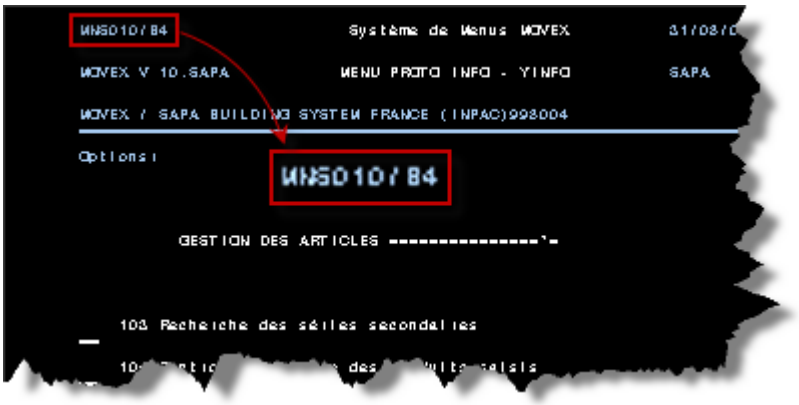


Figure 1 - 8: Legacy application screen identifier

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

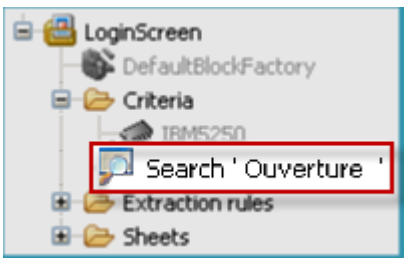


Figure 1 - 9: "Find string" criterion in screen class Criteria folder

EXTRACTION RULE

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

During a *webization* or *transaction* process, when a screen class is detected (i.e. all criteria defined for this screen class match), all of its extraction rules - including those inherited from its parent screen classes - are executed to extract data.

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

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

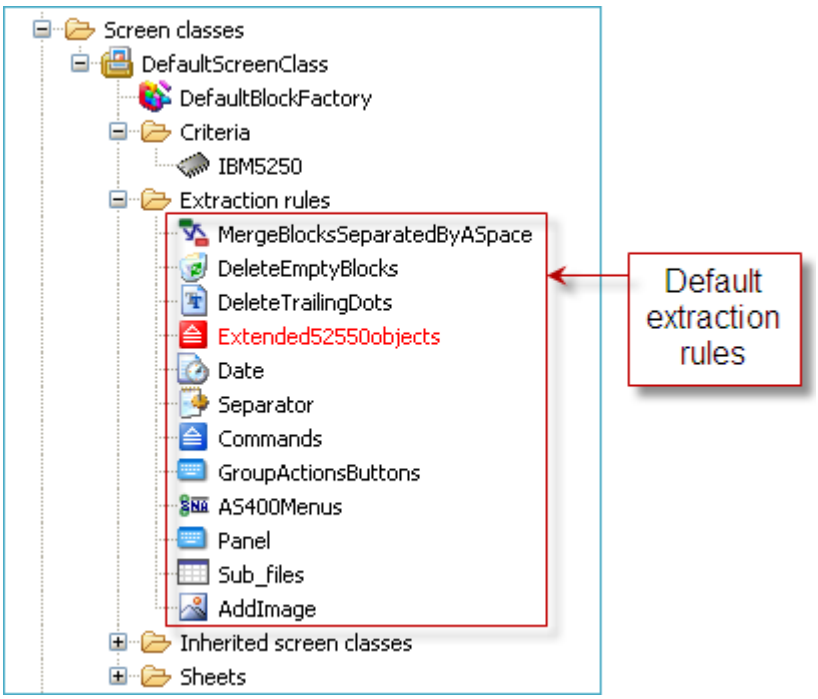


Figure 1 - 10: Default extraction rules

Their purpose is to :

- *extract* from any legacy screen a set of frequently used elements such as commands, separators, dates, AS/400 menus, etc. and present them in HTML format using already defined style sheets (see *"XSL and Cascading Style Sheets"* on page 1-10);
- *perform* a number of actions on the block structure of the legacy screen (for example merge blocks separated by a space, delete empty blocks, group trailing dots, etc.).

Extraction rules share three common properties:

Table 1 - 3: Extraction rule common properties

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

Table 1 - 3: Extraction rule common properties (...)

Property	Description
Type	Specifies which types of block are affected by the rule using a regular expression. There are three types of block: . *static*:non editable static text area; . *field*: editable text area; . *filler*: filler area.

When extraction rules are executed, blocks are extracted in XML format and an XML document is generated (XML representation of the legacy application screen). Blocks are described in XML format using ***XML standard attributes***:

Table 1 - 4: Block standard XML attributes

Attribute	Description
background	Defines the name of the block background color.
blink	Defines if the block has the « blinking » presentation attribute (`false` if the XML attribute is not specified).
column	Defines the block starting column on emulation screen.
foreground	Defines the name of the block text color.
intense	Defines if the block has the « intense » presentation attribute (`false` if the XML attribute is not specified)
line	Defines the block starting line on the emulation screen.
name	Defines the block name.
reverse	Defines if the block has the « inverse video » presentation attribute (`false` if the XML attribute is not specified)
type	Defines the block type (several block families can be created). Used as `match` value in XSL style sheet template rules to process XSL transformation of extracted blocks (see *"XSL and Cascading Style Sheets"* on page 1-10).
underlined	Defines if the block has the « underlined » presentation attribute (false if the XML attribute is not specified).

For example, if we consider the following word sequence divided into blocks:

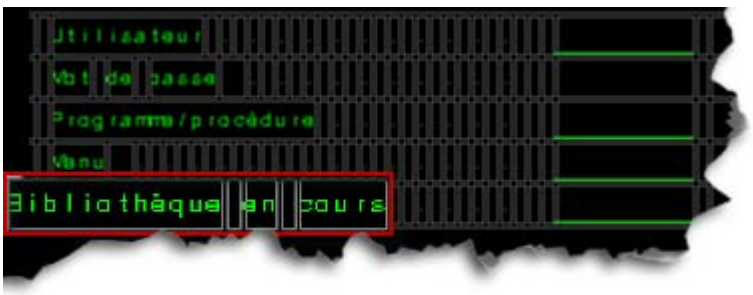


Figure 1 - 11: Example of extracted blocks

After they have been merged into one single block through the `MergeBlocksSeparatedByASpace` default extraction rule, these blocks are extracted according to extraction rules. Their XML format is the following:

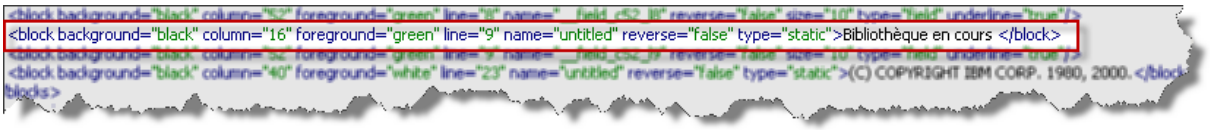


Figure 1 - 12: Extracted data in XML

The XML data are then transformed into HTML using CSS and XSL style sheets.

XSL and Cascading Style Sheets

After data has been extracted in XML format, it is turned into HTML for display purpose through ***XSL transformation***. The whole process - extracting data into XML then transforming it into HTML for display on Web screens - is known as *Webization*.

XSL transformation is achieved using XSL style sheets. CLP includes a number of style sheet templates. They are created by default when creating a project.

Style sheet templates are stored in the **Sheets** folder of the `DefaultScreenClass` screen class:

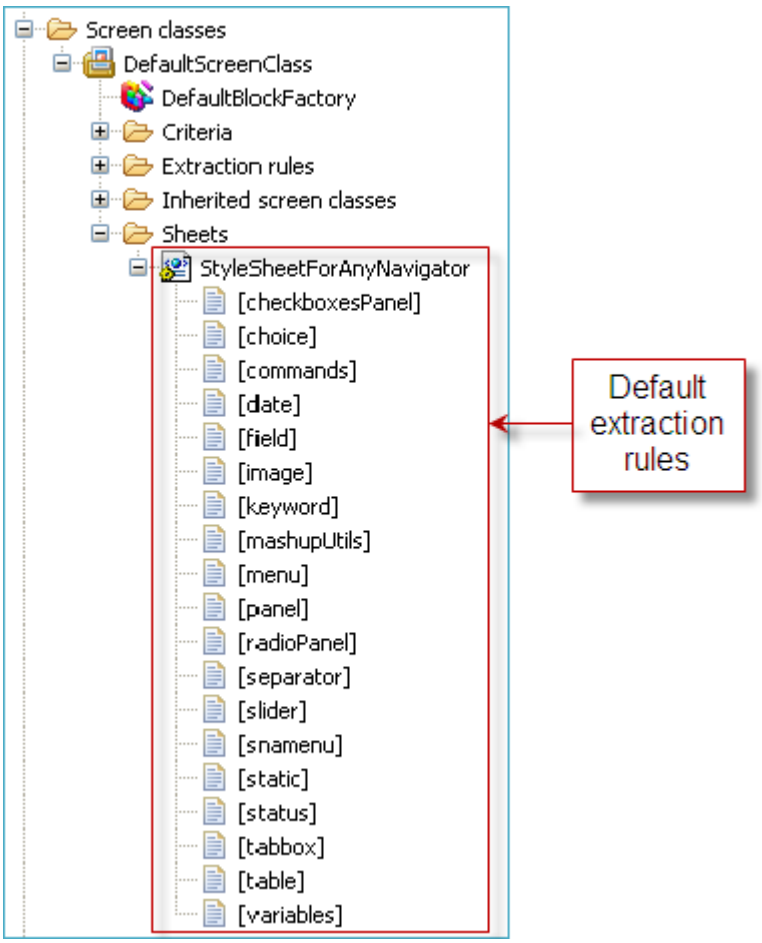


Figure 1 - 13: Default style sheets

The purpose of these *ready-to-use* style sheets is to process the XSL transformation of known XML data - XML data is generated by default extraction rules - such as dates, menus, fields, tables, etc. This process is called *dynamic Webization*.

Style sheet templates match on extracted data *types*. For example, a date extracted on

execution of the `Date` extraction rule is given the "date" XML type attribute (see Table 1 - 4 on page 1-9). The data is then transformed into HTML using the `date.xsl` XSL style sheet, which matches on "date" type elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <!-- TEMPLATE DATE -->
    <xsl:template match="*[@type='date']">
        <xsl:param name="fieldSize">
            <xsl:variable name="sWidth" select="/documen
            <xsl:if test="(@column + @size) &lt; $sWid
```

Figure 1 - 14: Date style sheet - <xsl:template> element

For an example of dynamic Webization, see "Connector View / Document Editor" on page 1-21.

The generic Webization XSL style sheet is `sna.xsl`. It includes all default XSL style sheet templates.

In CLP, data formatting is also achieved through the use of ***cascading style sheets*** (CSS) as regards *presentation attributes*. The generic CSS is `sna.css`. It is stored in the **css** folder of the project files tree structure (**Project Explorer**).

For more information on how to edit the generic Webization CSS, see "Editing the Generic Webization CSS" on page 2-28.

Generic style sheets (`sna.xsl` and `sna.css`) can be edited in **Document Editors** (see *"Document Editor"* on page 1-27) from the **Project Explorer View** to meet user's specific data formatting and transformation needs:

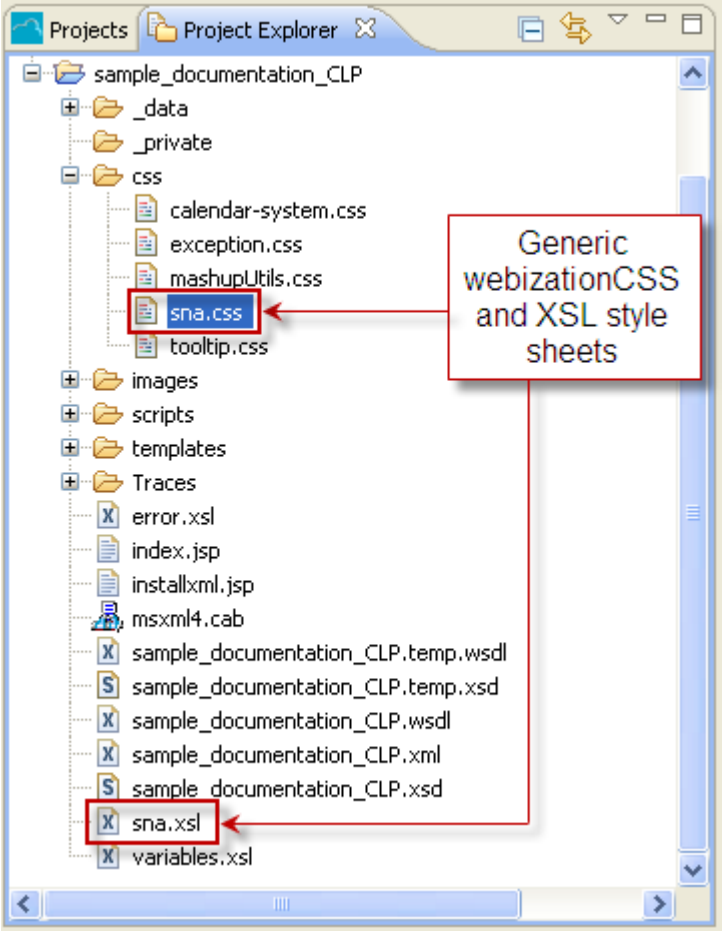


Figure 1 - 15: Generic Webization CSS and XSL style sheet

For more information on CSS and XSL, see the CSS and XSL tutorials from W3Schools at http://www.w3schools.com.

TRANSACTION

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

Once a transaction has been defined, it appears in the *Transactions* folder of the connector, in the **Projects View** of the CLP Studio (*"CLP Studio"* on page 1 - 14):

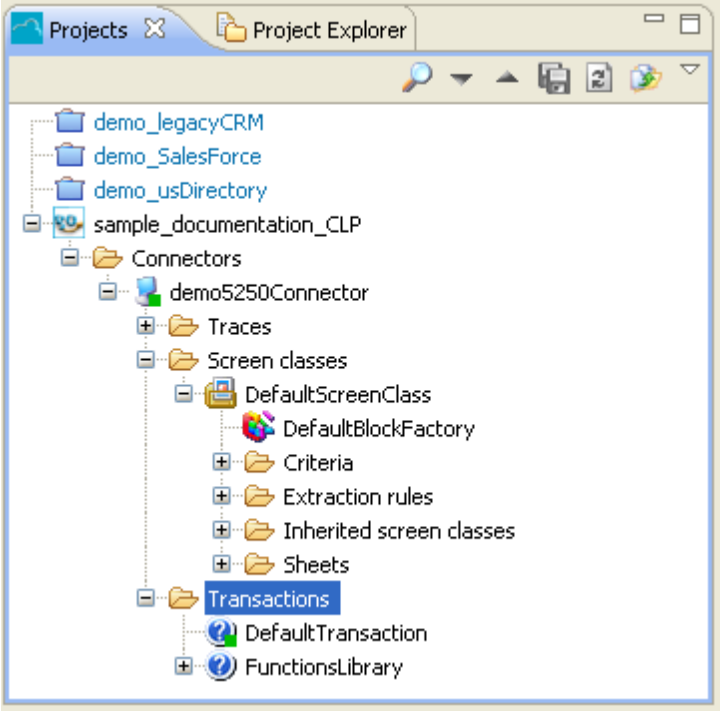


Figure 1 - 16: Transactions folder

*Notions of **transaction** and **screen class handler** are defined here for information only. No transaction or screen class handler is set as part of this Quick Guide project. For examples of such settings, refer to the "Starting with Convertigo Legacy Integrator" Quick Guide.*

SCREEN CLASS HANDLER

Basically, a screen class handler is meant to answer the following question : *"Now that this screen has been accessed and detected as belonging to given screen class, what am I supposed to do?".*

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

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

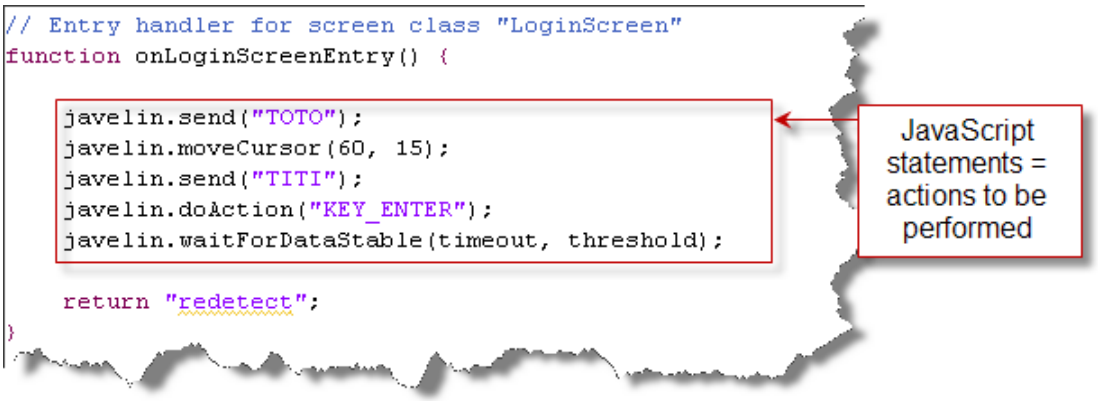


Figure 1 - 17: Screen class handler triggered on detection of the LoginScreen screen class

Once a screen class handler has been created, it appears in the *Functions* folder of the transaction, in the **Projects View** of the CLP Studio (*"CLP Studio"* on page 1 - 14):

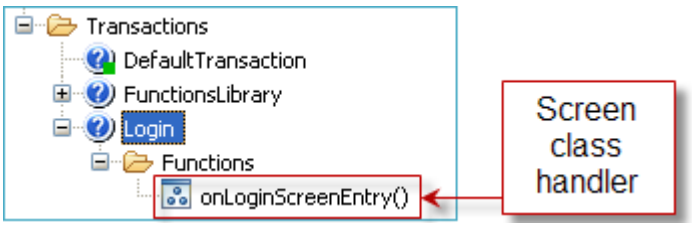


Figure 1 - 18: Screen class handler in transaction Functions folder

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

1.3 CLP Studio

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

- General Presentation
- View Description

1.3.1 General Presentation

Based on an Eclipse platform, the CLP 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:

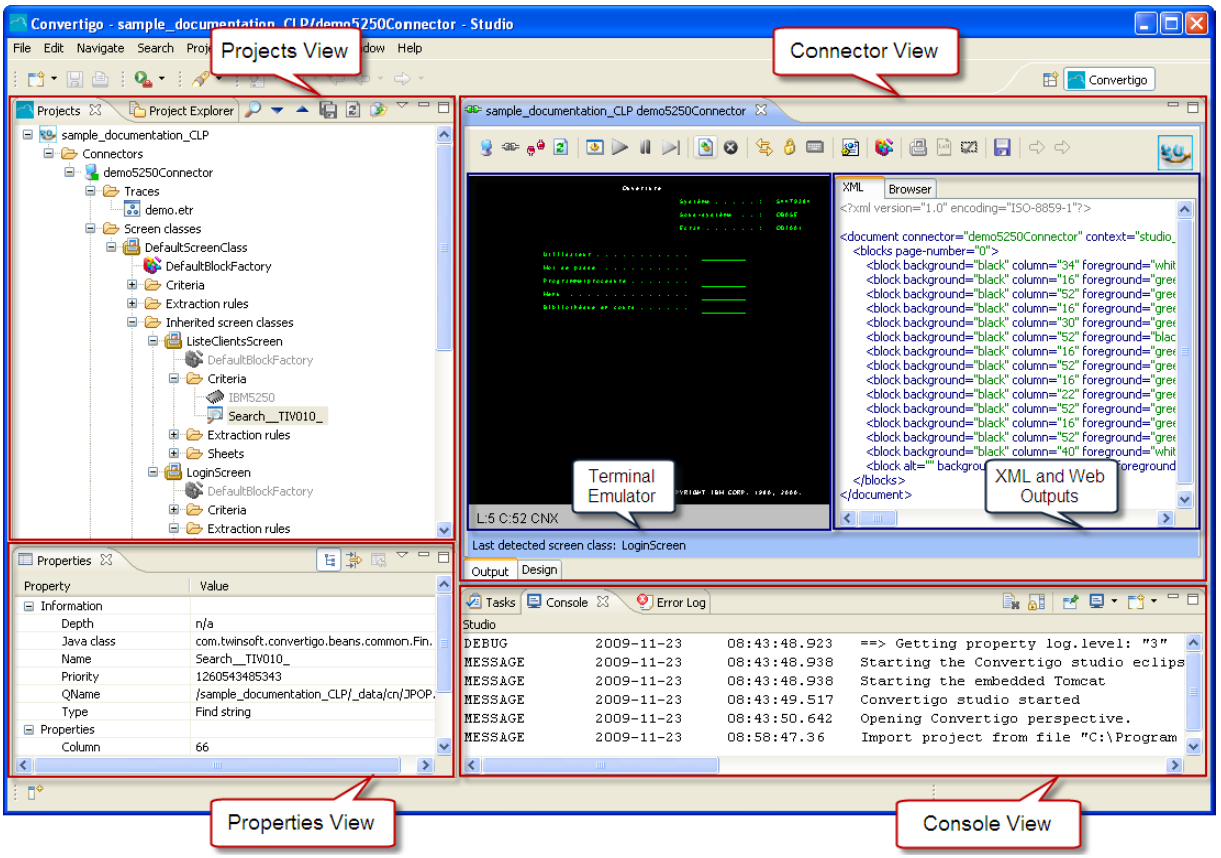


Figure 1 - 19: CLP Studio

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

Table 1 - 5: CLP views

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

1.3.2 View Description

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

Table 1 - 6: Standard Eclipse buttons

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

The following section describes the four panes of the CLP 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 - 6 on page 1-16), the **Projects View** includes tabs and icons:

Table 1 - 7: Projects View tabs and icons

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

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

- plain text - unchanged object;

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

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

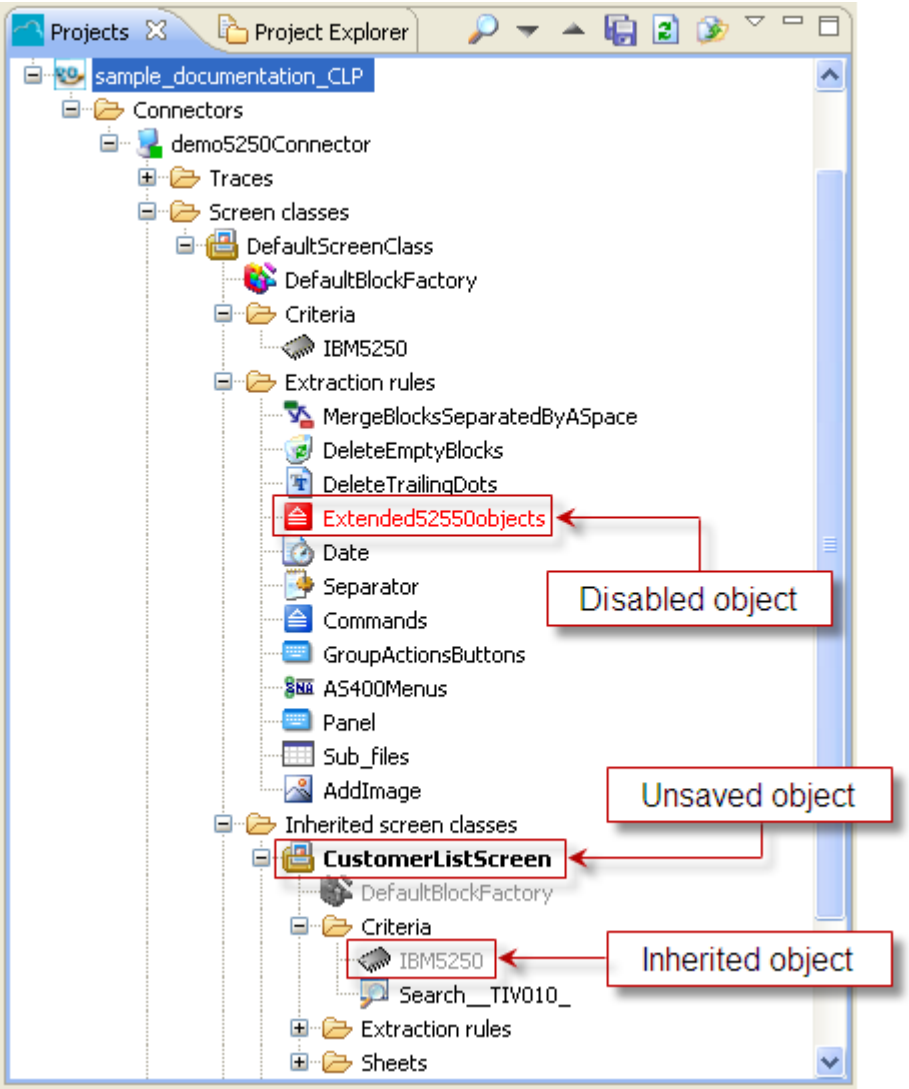


Figure 1 - 20: Formatting of unsaved, inherited and disabled objects in the Projects View

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 .

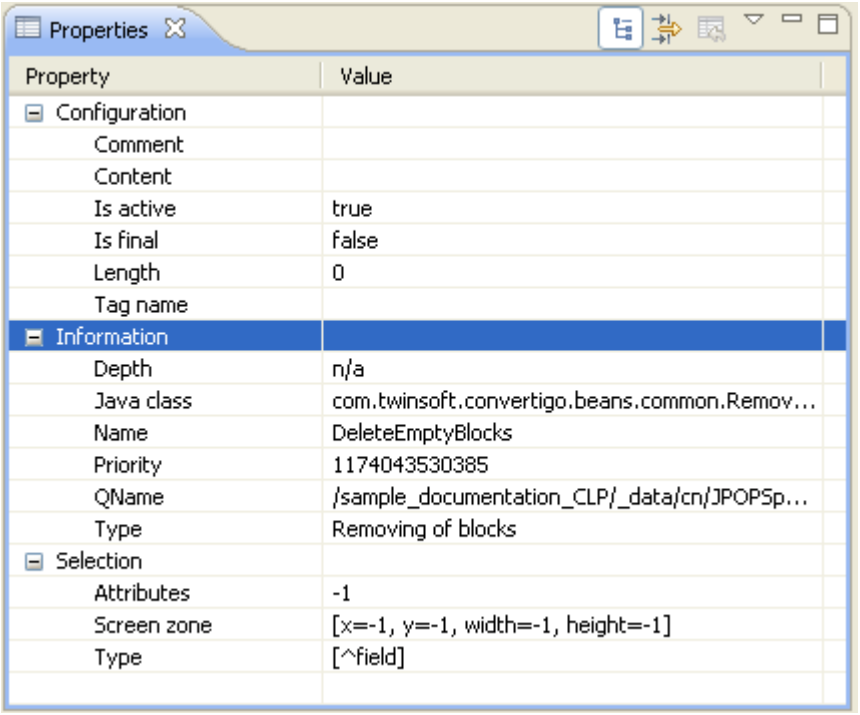


Figure 1 - 21: Properties View

In addition to object properties and standard Eclipse buttons (see Table 1 - 6 on page 1-16), the **Properties View** includes one tab (*Properties*) and icons:

Table 1 - 8: Properties View icons

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

To edit properties

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

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

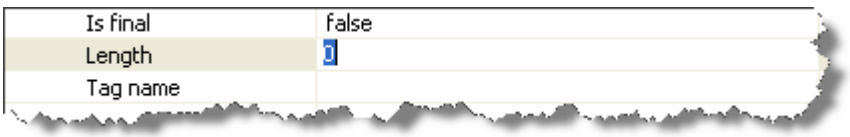


Figure 1 - 22: Text property field edition

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

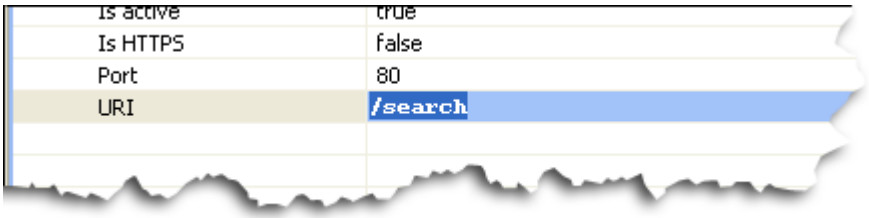


Figure 1 - 23: Javascript expression field edition

- the value is not highlighted but a symbol appears on the right of the field:

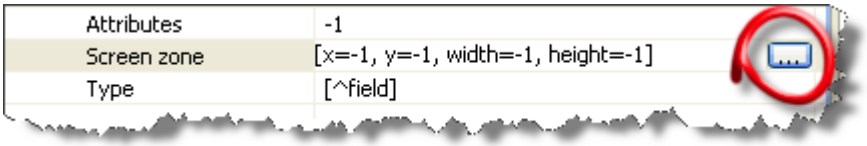


Figure 1 - 24: Text property edition via edition window

Click on . Depending on the property type:

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

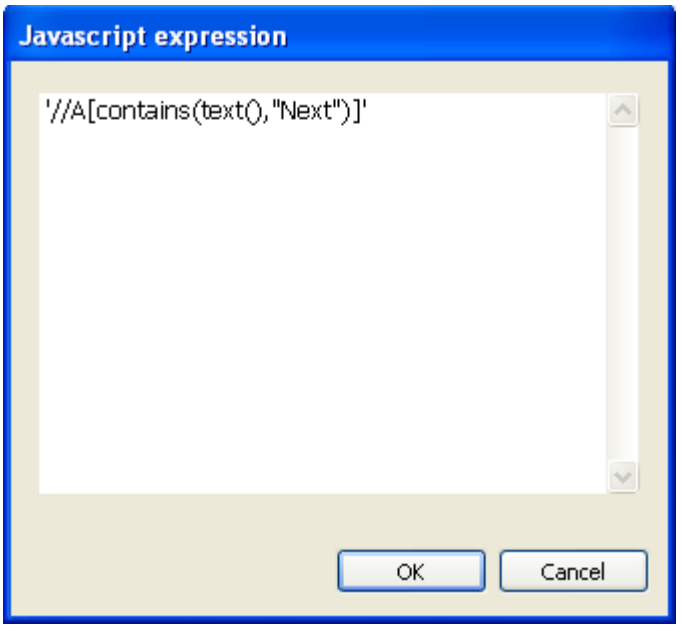


Figure 1 - 25: Javascript expression window

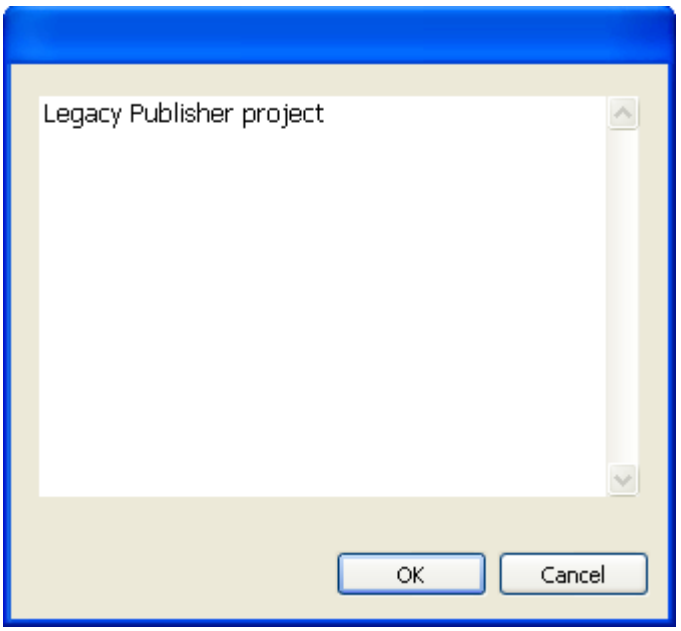


Figure 1 - 26: Edition window

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

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

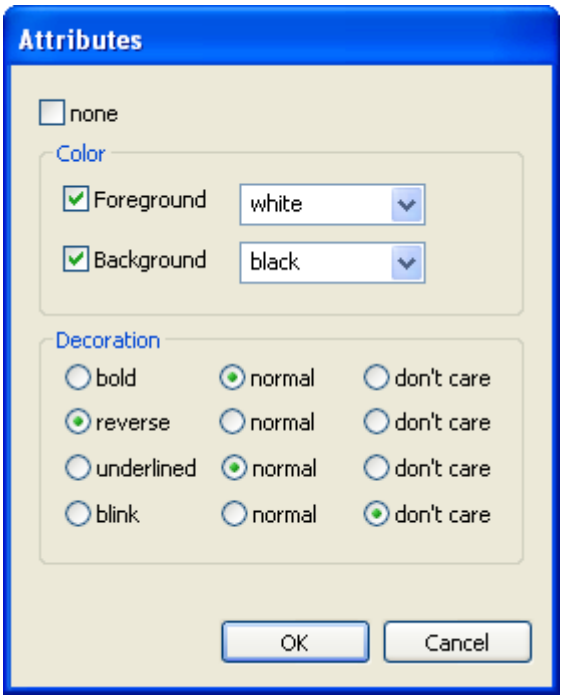


Figure 1 - 27: Configuration window

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

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

Figure 1 - 28: Text property edition via drop-down menu

1 Click on . A drop-down menu appears:

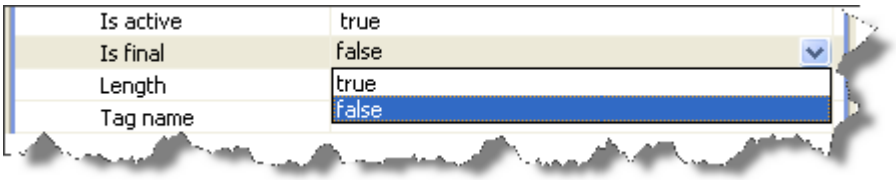


Figure 1 - 29: Selecting a value in the drop-down menu

2 Select a value in the drop-down menu.

3 Press `Enter`.

Connector View / Document Editor

Connectors open in a specific view called **Connector View**, whereas text documents open in standard Eclipse text editors called **Document Editors**.

Although the **Connector View** features two tabs (**Design** and **Output**), the **Design** tab is not used with Legacy connectors.

The **Output** tab of the **Connector View** is divided into three parts:

Table 1 - 9: Connector View parts

Name	Description
Emulator	Connects to the legacy application. Displays legacy application screens, which are divided into blocks for extraction purpose (see *"Block Factory"* on page 1-2).
XML output	Displays XML code resulting from extraction, after all extraction rules have been executed.
Browser output	Displays the HTML document resulting from XSL transformation of the XML code displayed in the XML output.

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

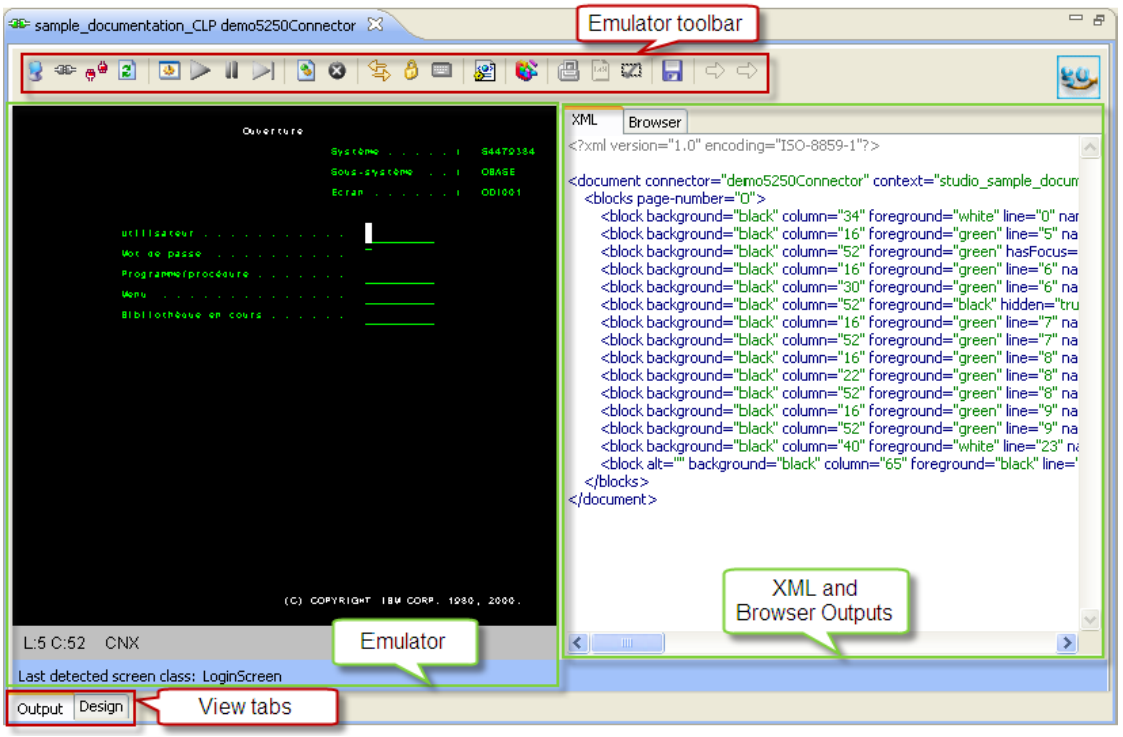


Figure 1 - 30: Connector View (Output tab selected by default)

The **Output** tab is selected by default. The **Design** tab is useless for this module of the Convertigo Enterprise Mashup Studio.

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

Table 1 - 10: Connector View icons

Icon	Description
	Renew the connector icon. Renews the connection according to the connector parameters set for the project.
	Connect the connector icon. Connects the emulator to the application defined in the connector of the project.
	Disconnect the connector icon. Disconnects the emulator from the application defined in the connector of the project.
	Refresh connector content icon. Refreshes the content of current legacy application screen.
	Debug mode icon. Activates debug mode. In Debug mode, the XML generation process (mainly through execution of extraction rules) pauses after each step.
	Continuous debug mode icon. Activates continuous debug mode. When in debug mode, CLP proceeds with XML extraction by executing remaining steps without pause until XML is generated or process is manually paused again.
	Pause the debug process icon. Pauses XML extraction. Available only when continuous debug mode is activated.

Table 1 - 10: Connector View icons (...)

Icon	Description
	Step-by-step debug mode icon. Activates step-by-step debug mode.
	Generate XML icon. Generates the XML code of the legacy screen currently displayed in the **Emulator** by executing the related extraction rules.
	Stop transaction icon. Stops the current transaction (while a transaction is in process).
	Show current screen class icon. Highlights in light grey in the **Projects View** the screen class detected for the legacy application screen currently displayed in the **Emulator**.
	Link current selection to highlighted property in Properties panel icon. Shortcut command for automatically setting screen zone property (**Screen zone**, **Image layout**, etc.) coordinates.
	Add element from current selection to highlighted property in Properties panel icon. Shortcut command for automatically adding a column to a table when setting a *Table* extraction rule.
	Generate XSL icon. Generates an XSL style sheet for a given screen class. The XSL style sheet is based on the XML output generated for this screen class, and ready for edition using any editor.
	Show blocks icon. Show the block structure of currently displayed legacy screen.
	New ScreenClass icon. Creates a new screen class based on detection criterion selected on screen.
	New TagName icon. Creates a new tag name extraction rule based on selected screen element under the screen class that Convertigo detects for the screen currently displayed in the emulator.
	New WaitAt icon. Available for Videotex, VT connectors only. Shortcut command for generating the code of a synchronisation based on appearance of an element on screen.
	Record trace icon. Only for IBM TN3270 and TN5250 streams. When activated, CLP starts recording a trace of the emulator screens. Stops when icon is clicked on a second time. The trace is stored in the **Traces** folder of the connector in the following format: `YYYY-MM-DD_trace.etr`.
	Learn icon. Available only when a transaction is selected. Activates the *Learn* mode - any action performed in the emulator will be recorded and screen classes entry handlers generated. For advanced users only.
	Accumulate learning mode icon. Available only when *Learn* mode is on. Accumulate data on visited legacy application screens. Exit screen class handlers will automatically be created in the transactions. For advanced users only.

EMULATOR

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

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

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

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

As mentioned in Table *"Connector View icons"* on page 1-22, specific screen-related properties of objects selected in the **Projects View** can be directly set from the **Emulator** using the **Link current selection to highlighted property in Properties panel** icon .

For example, to set the **Screen zone** property of an extraction rule (see Table 1 - 3 on page 1-8): select the property in the **Properties View**, select the screen zone in the **Emulator,** then click on . The **Screen zone** property value is automatically set.

When the terminal emulation session is active, all keys associated with the emulated technology are available from the CLP application. For example, if the emulated terminal F3 function key is associated with a "display general menu" command, pressing F3 on the keyboard from CLP will display the legacy application general menu in the **Emulator** (see *"Connector View / Document Editor"* on page 1-21).

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

XML and Browser outputs

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

The **Browser Output** displays the result of the dynamic Webization, or XSL transformation (using default or user-defined XSL and cascading style sheets) of XML document displayed in the **XML Output**.

For example, if we consider the following legacy application screen:

Figure 1 - 31: Example legacy application screen

In terms of block structure, this example legacy application screen can be represented as follows:

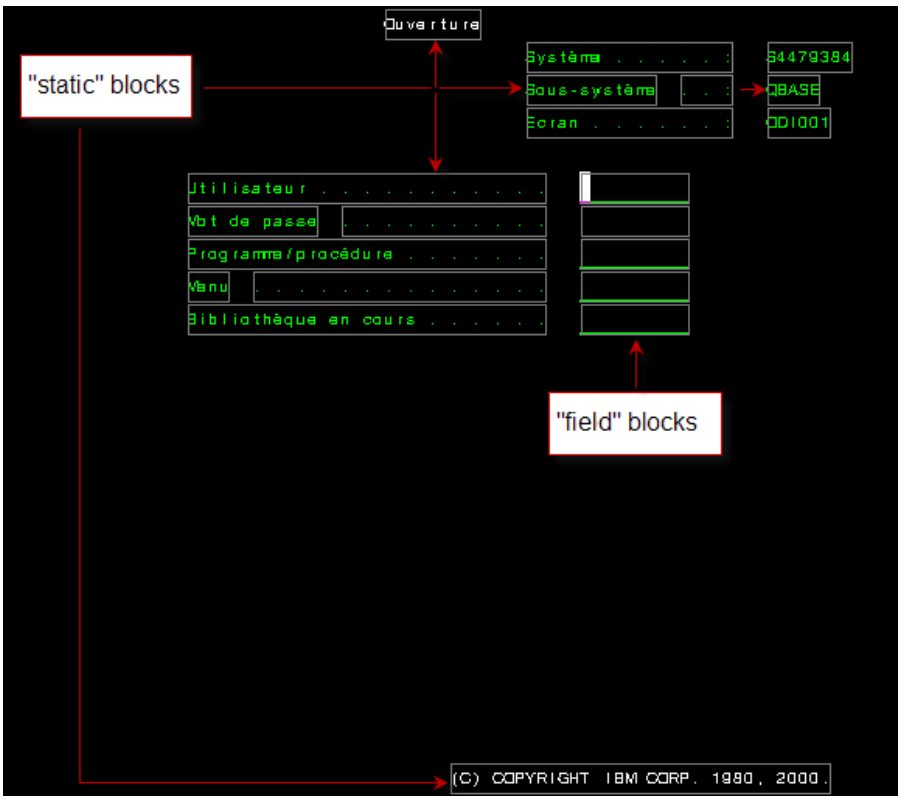


Figure 1 - 32: Block structure of the example legacy application screen

The resulting HTML document (*Webized* legacy application screen), based on XSL transformation and CSS, can be seen in the **Browser** tab of the **Connector View**:

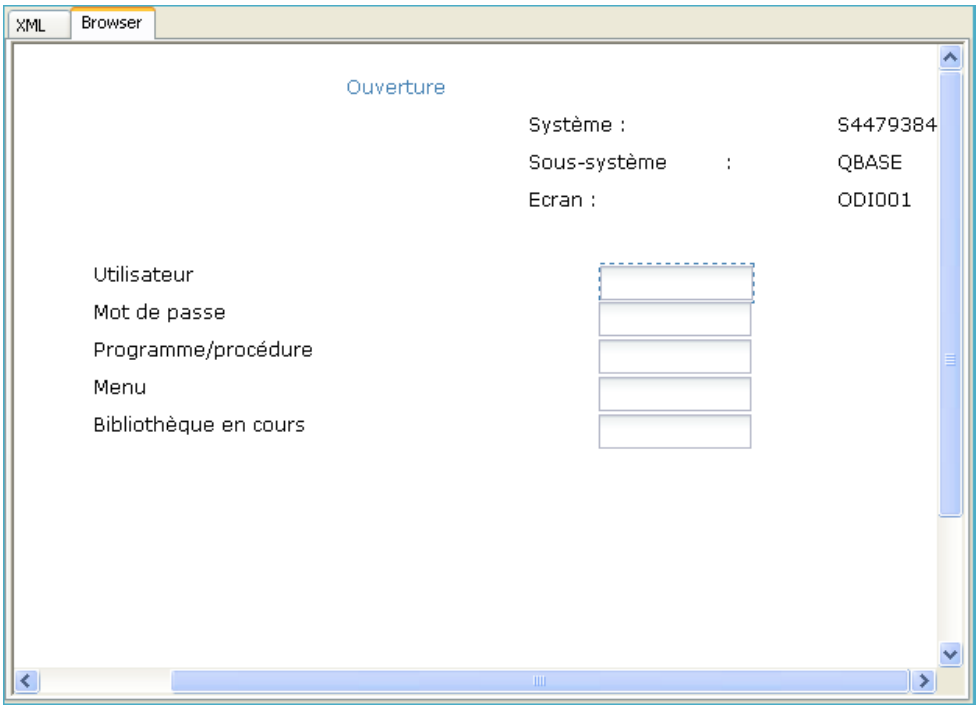


Figure 1 - 33: Dynamic Webization of legacy application screen

After default extraction rules have been executed (for example after the **Generate XML** command has been launched via the icon), an XML document is generated and displayed in the **XML** tab:

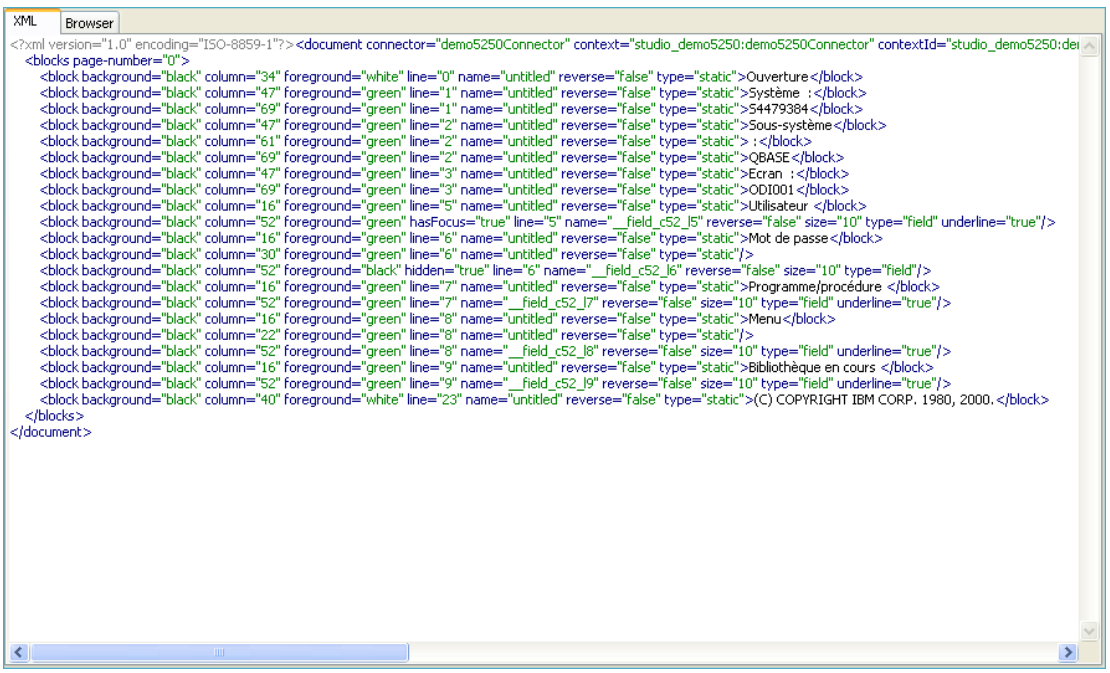


Figure 1 - 34: XML output after execution of extraction rules

The **XML** output contains all extracted blocks in XML format. These blocks are *typed* either as `static` or as `field` (that is, the `type` XML attribute of all extracted blocks has either the `static` or the `field` value):

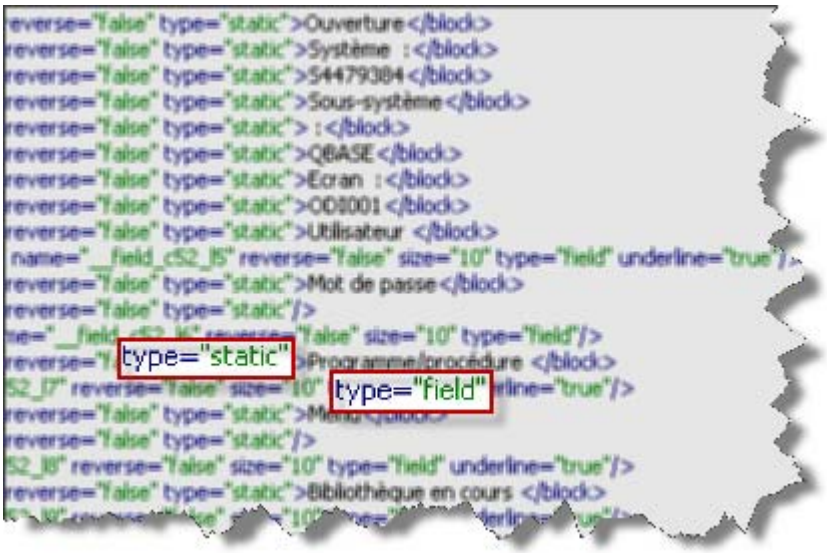


Figure 1 - 35: Static or field XML type attribute of extracted blocks

Document Editor

The **Document Editor** is used to display the content of various types of documents, which can be accessed for different purposes in the context of a given project.

For example, as explained in *"XSL and Cascading Style Sheets"* on page 1 - 10, blocks of `static` and `field` types (see Figure 1 - 35) are dynamically Webized (dynamically transformed into HTML objects after XSL transformation) using:

- template XSL style sheets (in this example, the `static.xsl` and `field.xsl` style sheets);
- style definitions contained in `sna.css`.

In the example legacy screen, white and green (as foreground color) static text as well as fields appear on black background. These styles are defined in the generic Webization CSS (`sna.css`), which opens in the **Document Editor** when double-clicking on it in the **Project Explorer**:

```
/* Base style of static text outside of tables*/
span.staticText {
    font-size:   12px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
}

/* Foreground color styles for static text */
span.colorwhite {
    color: SteelBlue;
}

span.colorgreen {
    color: black;
}

/* Background color styles for static text */

span.bgcolorblack {

}
/* New ext style for input fields */
```

Figure 1 - 36: Style definitions applying to "static" blocks of example legacy screen

```
/* New ext style for input fields */
input.extField {
    padding-bottom: 0;
    padding-top: 2px;
    height: 18px;
    line-height: 18px;
    vertical-align: middle;
    background: #FFFFFF url(../images/text-bg.png) repeat-x scroll 0 0;
    border: 1px solid #B5B8C8;
    padding: 1px 3px;
    -x-system-font: none;
    font-family: tahoma,arial,helvetica,sans-serif;
    font-size: 12px;
    font-size-adjust: none;
    font-stretch: normal;
    font-style: normal;
    font-variant: normal;
    font-weight: normal;
    margin: 0;
}
```

Figure 1 - 37: Style definitions applying to "field" blocks of example legacy screen

Style definitions can then be edited to meet user's needs and saved using the **Document Editor**.

2 My First Convertigo Legacy Publisher Project

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

- Legacy Publishing Project
- Setting up the project
- Customizing Webization

2.1 Legacy Publishing Project

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

- Project Description
- Opening the Sample Project from the Studio

2.1.1 Project Description

This sample Legacy Publishing project (called `sample_doc_CLP`) will teach you how to:

- *Create* a new CLP project (and set its connector) - see *"Creating a Project and Setting Connector Parameters"* on page 2-3
- Dynamically *XMLize (Webize*) legacy application screens using the default screen class and extraction rules - see *"Dynamically webizing legacy application screens"* on page 2-7.
- *Customize* Webized screens by:
  - editing the webization page template - *see "Customizing the Webization Page Template"* on page 2-12*;*
  - creating extraction rules on the default screen class (in this project, an *Image* extraction rule) - see *"Creating Extraction Rules on the Default Screen Class"* on page 2-22;
  - editing the generic Webization CSS - see *"Editing the Generic Webization CSS"* on page 2-28;
  - creating screen classes and screen class-specific extraction rules (in this project, *Removing of text* and *Table* extraction rules) - see *"Defining Screen Classes and Screen Class-Specific Extraction Rules"* on page 2-32.

This CLP project is based on an IBM 5250 legacy application.

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***:

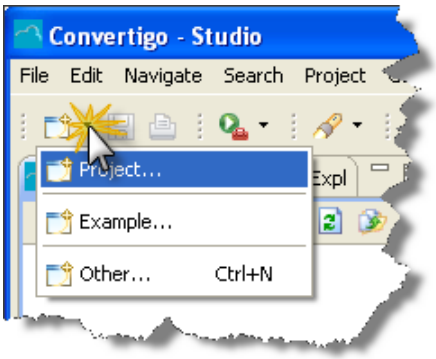


Figure 2 - 1: Launching the New Project wizard

A **New Project** wizard opens.

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

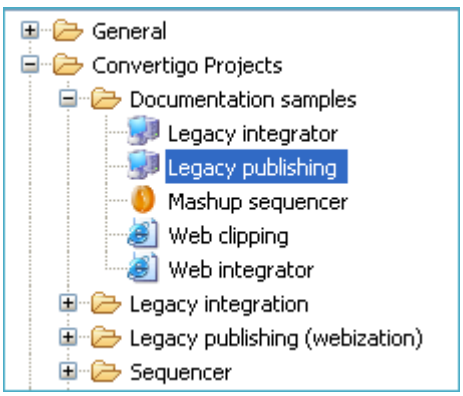


Figure 2 - 2: Selecting the CLP sample project

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

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

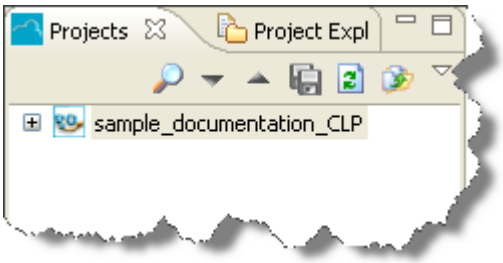


Figure 2 - 3: Sample project in Projects View

You can now start setting up your own CLP project, `sample_doc_CLP`.

2.2 Setting up the project

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

- Creating a Project and Setting Connector Parameters
- Dynamically webizing legacy application screens

2.2.1 Creating a Project and Setting Connector Parameters

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

To create a project and set a connector

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

   A **New Project** wizard opens:

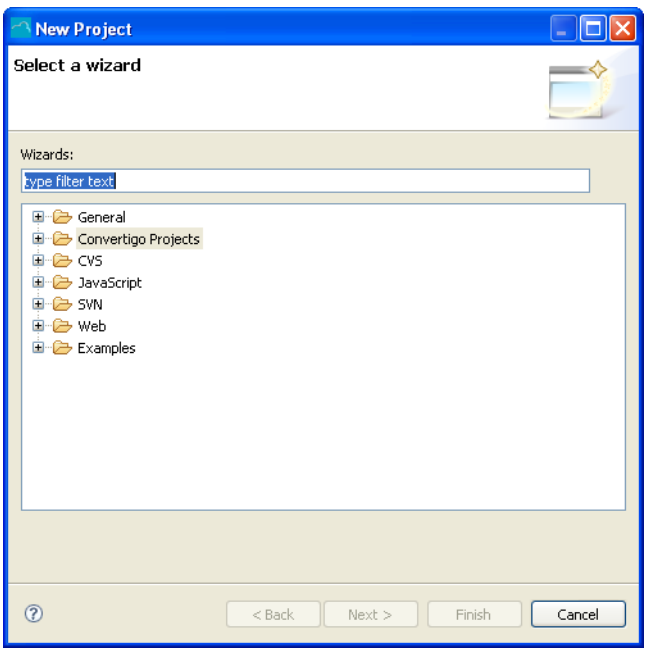


Figure 2 - 4: New Project Wizard

3. Expand **Convertigo Projects**, then **Legacy Publishing (Webization)**, and select **5250 Web Style**:

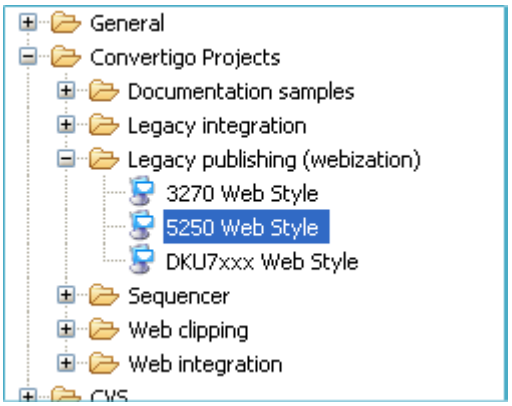


Figure 2 - 5: Selecting a project type

4 Click on **Next**:

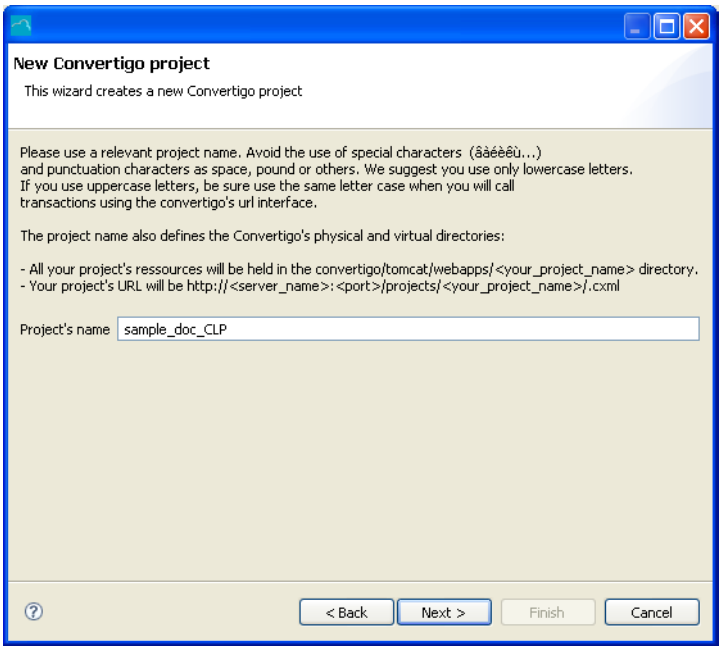


Figure 2 - 6: Entering a project name

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

Project names are part of the URL adress used for calling a project from a Web browser (see "To call webized legacy application screens from a Web browser" on page 2-10). It is therefore recommended to avoid project names with blank spaces.

6 Click on **Next**:

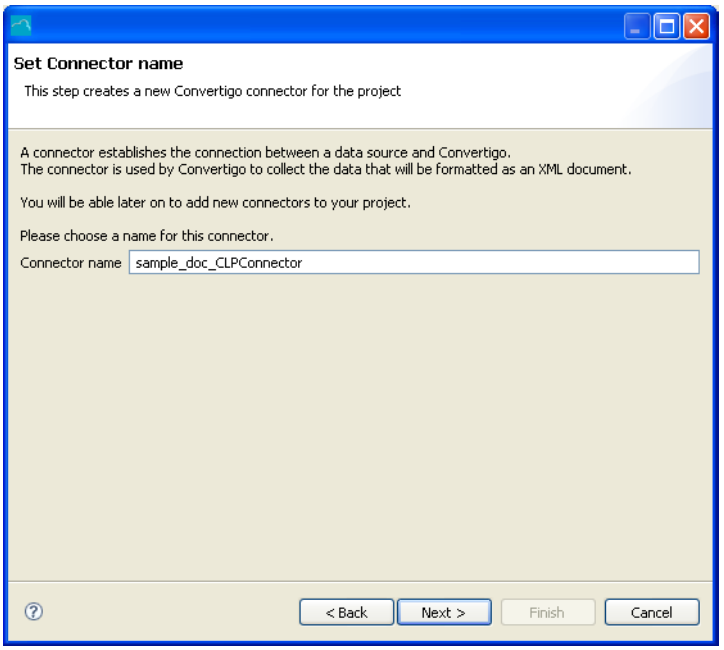


Figure 2 - 7: Entering a connector name

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

7 Click on **Next**:

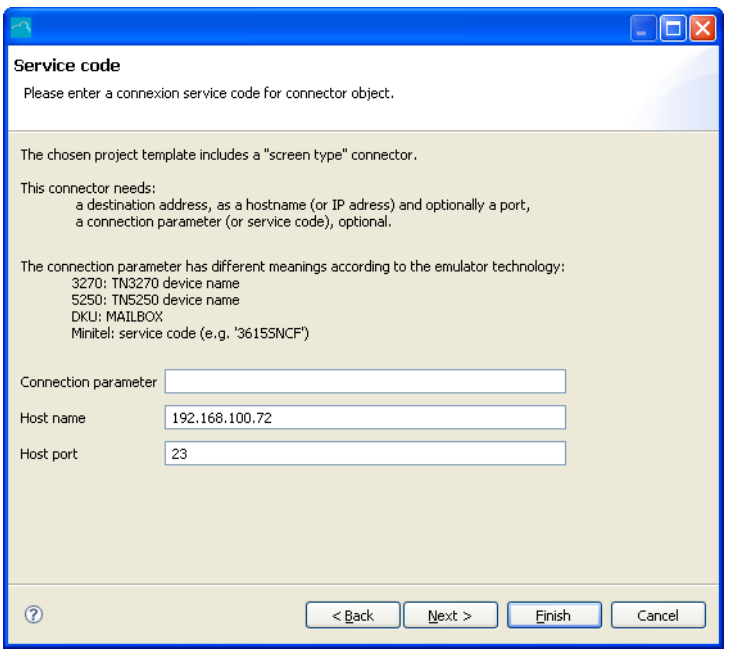


Figure 2 - 8: Setting connection parameters

8 Enter required information:

- in the **Connection parameter** field, type in the name of the device, mailbox or service code to which CLP must connect (optional parameter).
- in the **Host name** field, type in the host name or IP address of the server hosting the legacy application.

- in the **Host port**, type in the port of the server hosting the legacy application.

9 Click on **Next**.

The **New project summary** window opens:

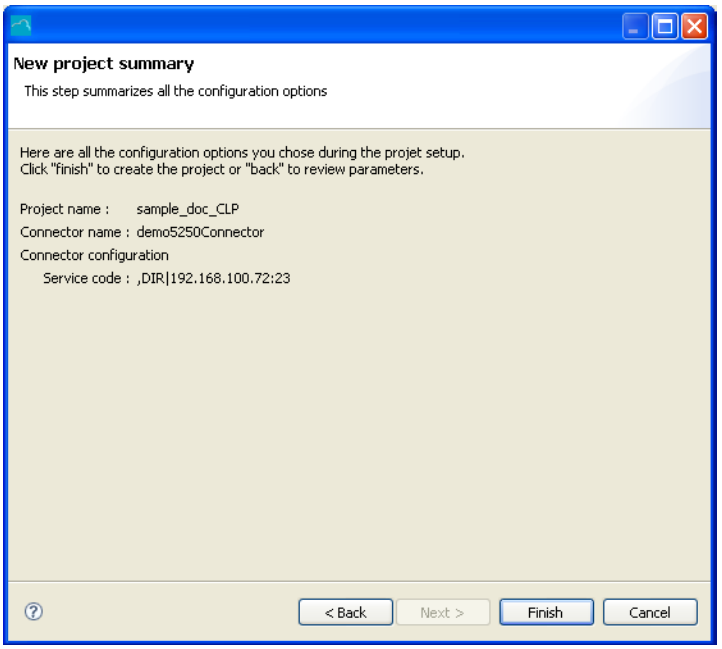


Figure 2 - 9: New project summary window

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

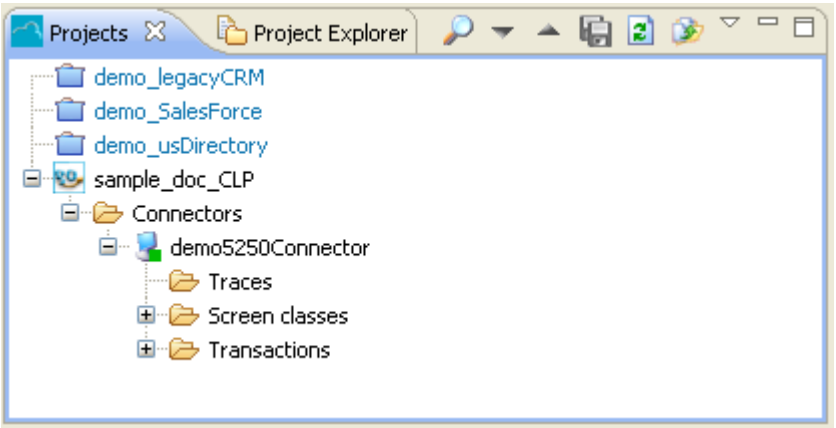


Figure 2 - 10: sample_doc_CLP project appearing in the Projects View

11 Connect the emulator by clicking the **Connect the connector** icon .

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

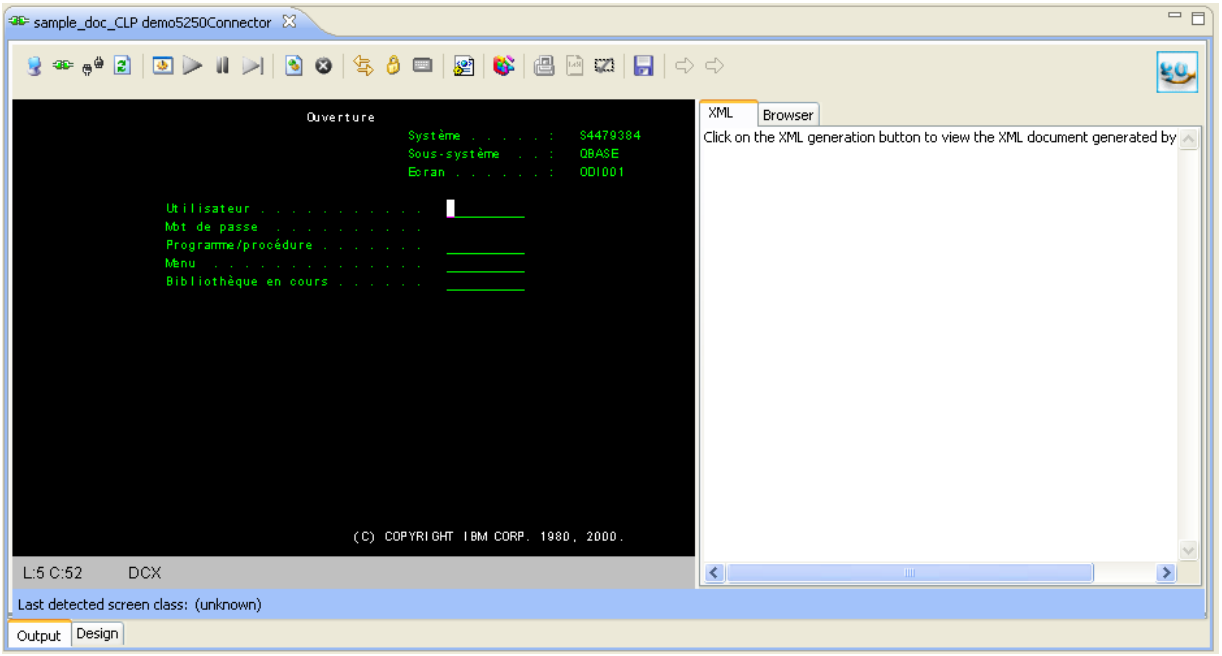


Figure 2 - 11: Emulator displaying legacy application screen

This is the starting point of the project.

WHAT COMES NEXT?

At this point in the project, legacy application screens can be dynamically webized using default extraction rules and style sheet templates (see *"Extraction rule"* on page 1-7 and *"XSL and Cascading Style Sheets"*).

The default dynamic webization, that is to say, dynamic webization based on default extraction rules and style sheet templates:

- applies mostly to legacy screens including exclusively "usual" screen elements such as tables, menus, text, etc.
- extracts and webizes legacy elements "as are" - no customization is performed yet and the webized screen is perfectly similar, in terms of content, to the legacy application screen displayed in the **Emulator**.

For webizing more specific screens or creating specific extraction rules (for example to remove parts of legacy screens from webization, add images, etc.), see "Customizing Webization" on page 2-12.

The following section describes how to dynamically webize legacy application screens using default predefined objects (extraction rules and style sheet templates).

2.2.2 Dynamically webizing legacy application screens

This section describes how to dynamically webize legacy application screens using default extraction rules and style sheet templates and how to call webized screens from a Web browser.

To XMLize and dynamically Webize a legacy application screen

Assuming that the **Emulator** displays the login screen of the legacy application, click on the **Generate XML** icon in the **Emulator** toolbar:

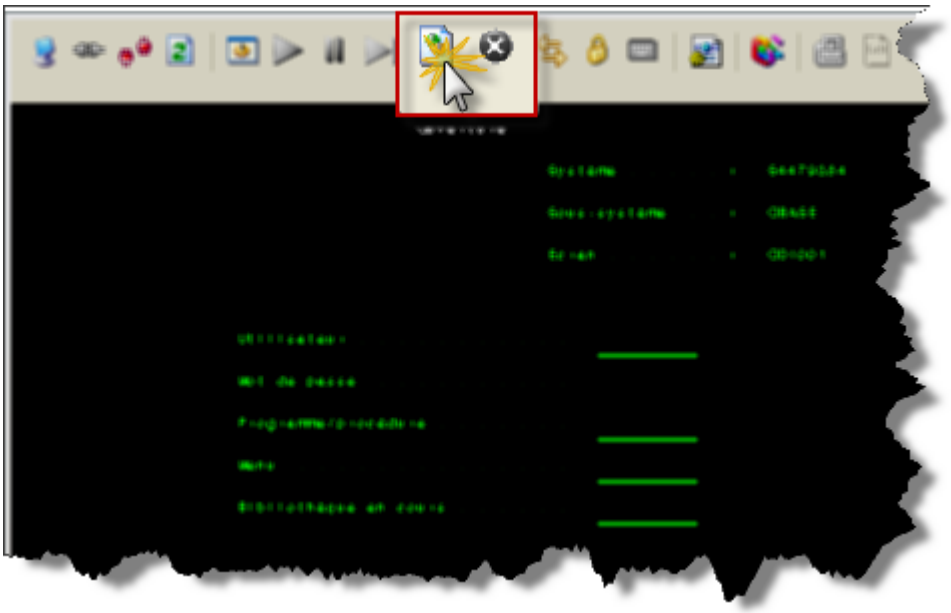


Figure 2 - 12: Generating XML of legacy application screen

The screen is:

- detected as belonging to the `DefaultScreenClass` screen class - this means that all extraction rules associated with this screen class (that is to say, default extraction rules) are executed on this screen when clicking on the icon.
- *XMLized* - following detection of the screen class and execution of associated extraction rules, the XML encoding (or representation) of the legacy application screen is generated. It appears in the **XML** tab of the **Connector View**:

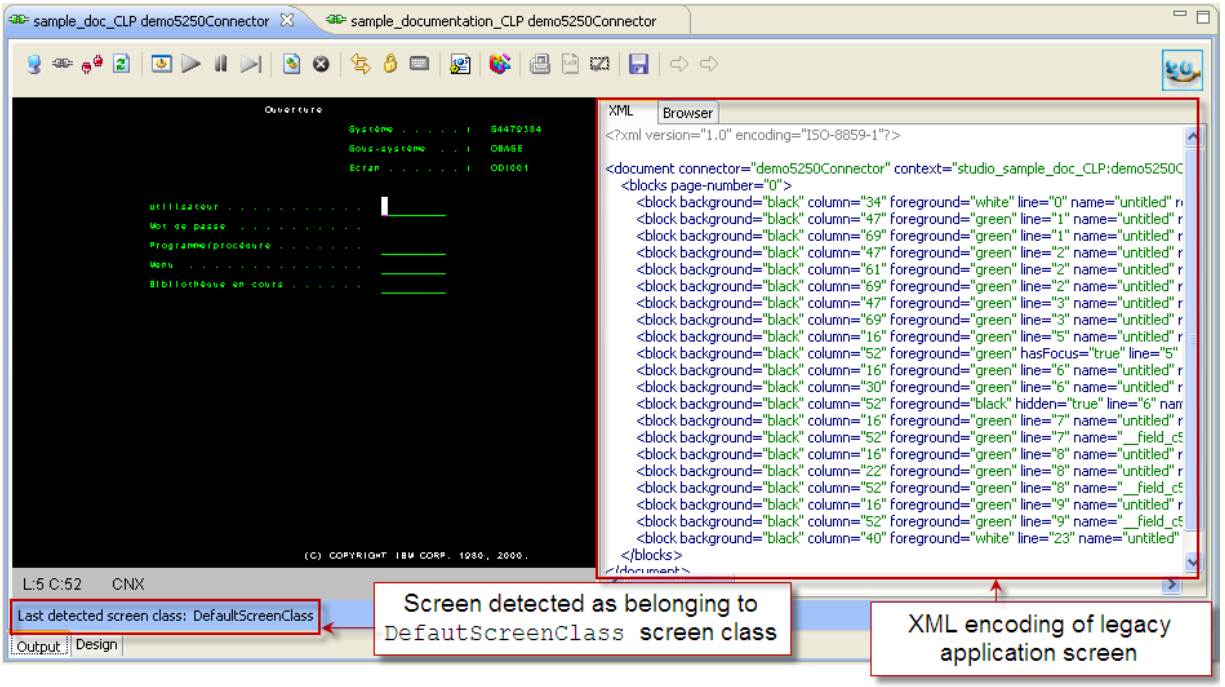


Figure 2 - 13: Generated XML and Last detected screen class

- dynamically *Webized* - in the XML output, legacy screen blocks (see *"Block Factory"* on page 1-2) are typed as `static`, `field`, etc. Style sheets matching on these types are then used to achieve the screen dynamic webization of corresponding elements. The webized legacy application screen appears in the **Browser** tab of the **Connector View**:

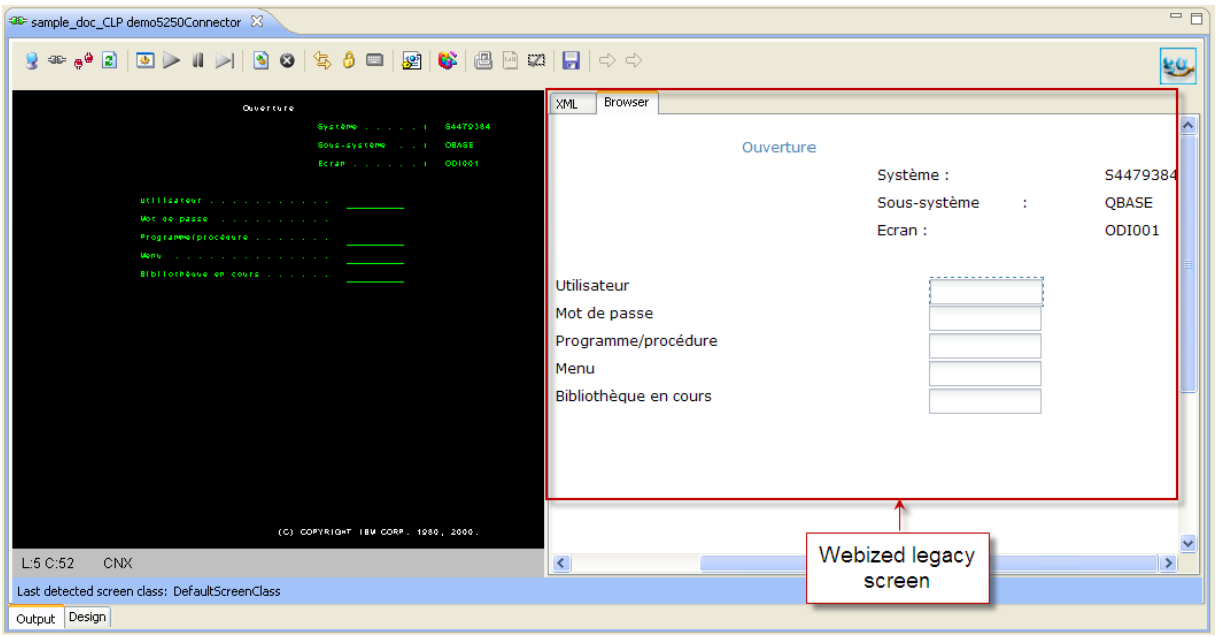


Figure 2 - 14: Webized legacy application screen

This operation can be repeated anytime on any legacy screen.

For example, if we pass on to a next screen, keep the **Browser** tab of the **Connector View** open and generate the XML again via the icon, the webized screen displayed in the **Browser** tab is dynamically webized to match the screen displayed in the **Emulator**:

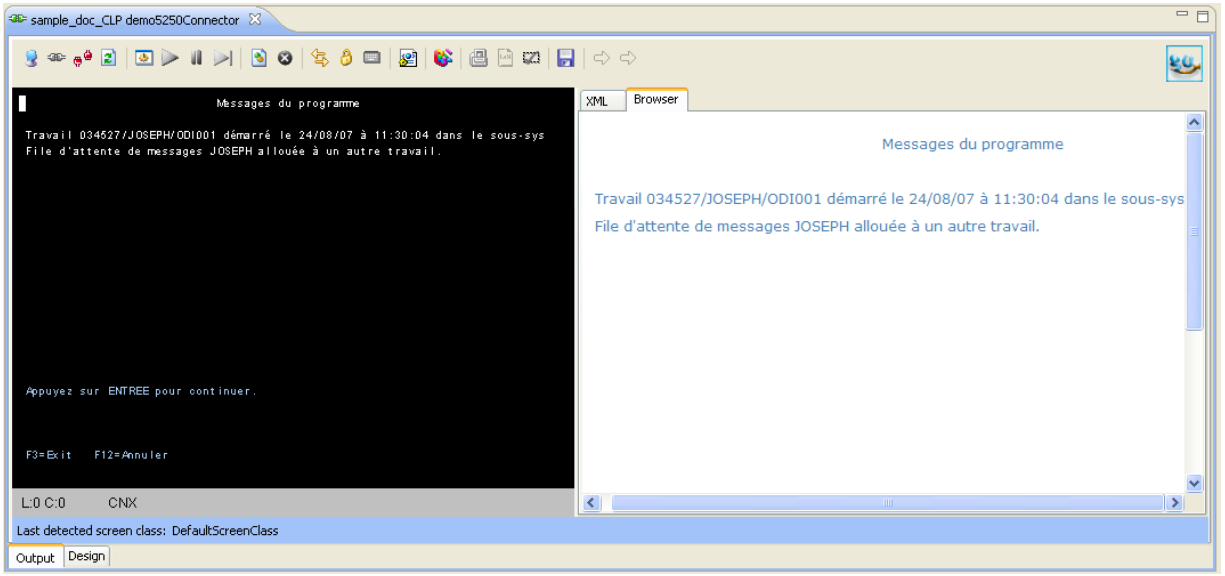


Figure 2 - 15: Second webized legacy application screen

In these examples, webized screens appear in a built-in browser. You can also choose to call webized screens from any external browser, via a properly formatted URL.

In this Quick Guide project, webized screens are called either from the internal or from an external Web browser.

To call webized legacy application screens from a Web browser

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

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

For example, in our project:

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

The legacy screen currently appearing in the **Emulator** in the studio is displayed in the browser (here, the login screen class):

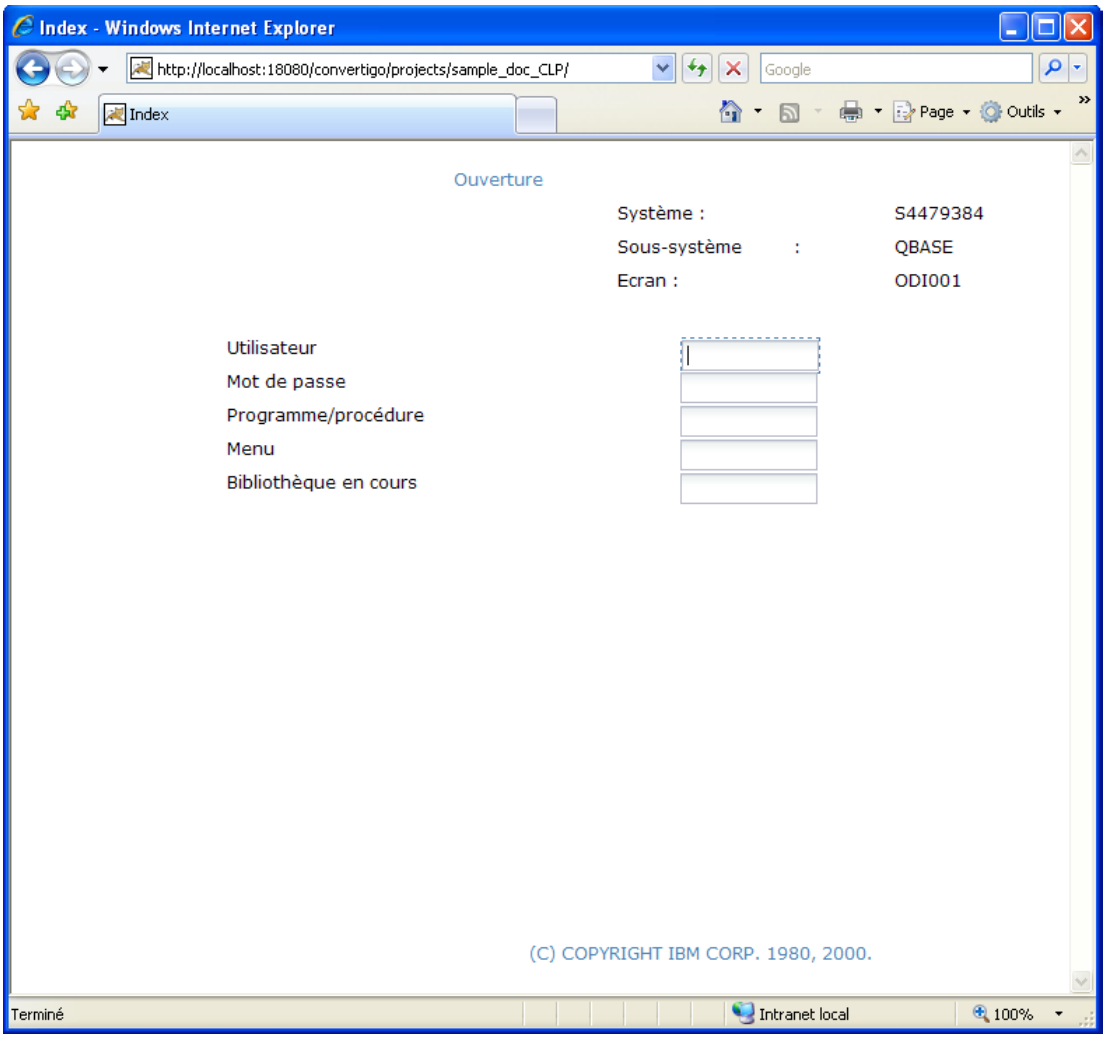


Figure 2 - 16: Legacy screen webized using default extraction rules and templates

If changes do not appear, empty the browser cache.

WHAT COMES NEXT?

As previously mentioned, so far in the project legacy screens are dynamically webized "as are".

The following steps consist in customizing the webization process.

In CLP, several methods can be applied to meet customization needs:

Table 2 - 1: Webization customization methods

Method	Comment
Modify the webization page template	For customizing the webization page template, for example to add a logo, group buttons, etc. on all webized pages
Create specific extraction rules on the default screen class	As long as no specific screen class is defined, these extraction rules, which are added to default extraction rules, are executed for each accessed screen

Table 2 - 1: Webization customization methods (...)

Method	Comment
Edit the generic Webization CSS (`sna.css`)	For customizing the formatting style of webized elements
Define screen classes and create screen class-specific extraction rules	For: • customizing the webization of "regular" legacy screens with specific extraction rules (other than default extraction rules); • webizing screens including elements not processed by default extraction rules/style sheet templates.

Customizing methods are described in further details in the the following section.

2.3 Customizing Webization

This section describes how to customize the webization process using different methods:

- Customizing the Webization Page Template
- Creating Extraction Rules on the Default Screen Class
- Editing the Generic Webization CSS
- Defining Screen Classes and Screen Class-Specific Extraction Rules

2.3.1 Customizing the Webization Page Template

A webization page is structured as:

- an HTML webization page template (empty by default), defined in an `index.jsp` file included in each CLP project:

```
index.jsp
<%@ page
    sample_documentation_CLP/index.jsp
    charset=iso-8859-1"
    language="java"
    import="java.util.*, java.io.*"
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-tr
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Index</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <script type="text/javascript" src="scripts/scriptlib.js"><!--script--></script>
    <script type="text/javascript" src="scripts/scriptlib-param.js"><!--script--></script>
    <script type="text/javascript" src="scripts/contextMenu.js"><!--script--></script>
    <script type="text/javascript" src="scripts/keymapsna.js"><!--script--></script>
    <script type="text/javascript" src="scripts/tooltip.js"><!--script--></script>
    <script type="text/javascript" src="scripts/position.js"><!--script--></script>
    <script type="text/javascript" src="scripts/calendar.js"><!--script--></script>
    <script type="text/javascript" src="scripts/calendar-en.js"><!--script--></script>
    <script type="text/javascript" src="scripts/calendar-setup.js"><!--script--></script>
    <script type="text/javascript" src="scripts/selectMenu.js"><!--script--></script>
    <script type="text/javascript" src="scripts/xmlparse.js"><!--script--></script>
    <script type="text/javascript" src="scripts/mashupUtils.js"><!--script--></script>
    <script type="text/javascript" src="../../scripts/button.js"><!--script--></script>
    <script type="text/javascript" src="../../scripts/cookies.js"><!--script--></script>

<script type="text/javascript" defer="false">
    getOriginalFontSize();
    ajaxXmlPostData(xmlhttp, "<%=data%>");
</script>

<!-- TEMPLATE HTML -->
<!-- Include your html template here -->

</body>

</html>
```

Webization page template

Figure 2 - 17: index.jsp file

- plus, a *viewport*, that is to say an area where webized elements are displayed. The position of the viewport depends on positioning variables defined in the `variables.xsl` file included in each CLP project:

```
index.jsp   variables.xsl
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/19
    <!--- WARNING : if you modify these constants, display may be wrong -->
    <!-- Set this to false if you want to disable dynamic screen content resizing -->
    <xsl:variable name="resize">false</xsl:variable>

    <xsl:variable name="coefx">9</xsl:variable>
    <xsl:variable name="coefy">24</xsl:variable>

    <!-- Viewport left margin -->
    <xsl:param name="offsetx">10</xsl:param>
    <!-- Viewport right margin -->
    <xsl:param name="offsetr">22</xsl:param>
    <!-- Viewport top margin -->
    <xsl:param name="offsety">20</xsl:param>

    <!-- Width of a scroll bar -->
    <xsl:param name="scrollwidth">17</xsl:param>

    <!-- Width of icons (like  page up/down incons in table XSL template) -->
    <xsl:variable name="iconSize" select="16" />

    <!-- Panel width enlargement -->
    <xsl:param name="offsetw">10</xsl:param>
    <!-- Panel height enlargement -->
    <xsl:param name="offseth">10</xsl:param>

    <!--Set this value to true if you want to display the action key in action buttons-->
    <xsl:variable name="DisplayActionKey">false</xsl:variable>

    <!-- Set this value to true if you want to display action buttons outside of panels -->
    <xsl:variable name="DisplayDisabledButtons">false</xsl:variable>

    <!-- Set this to true if you want to gather action buttons in a panel on the left of the screen -->
    <!-- Note : requires offsetx (left margin) to be at least 110px  -->
    <xsl:variable name="GroupActionButtons">false</xsl:variable>

    <!-- Help keyword string -->
    <xsl:variable name="helpKeywordString">(F4)</xsl:variable>
```

Figure 2 - 18: variables.xsl file

Customizing the webization page template therefore amounts to:

- adding HTML elements in the `index.jsp` HTML template section,
- and/or modifying *viewport* variables in the `variables.xsl` file.

The following procedures describes how to edit the `index.jsp` and `variables.xsl` files to:

1 Add a logo as a banner - see *"To edit the webization page template to add an image"* on page 2-14.

2 Group action buttons - see *"To edit the webization page template to group action buttons"* on page 2-18.

Both customized elements will be added in the upper left corner of webized pages.

To edit the webization page template to add an image

1 Click on the **Project Explorer** tab in the **Projects View**.

The `demo5250` project files tree structure is displayed:

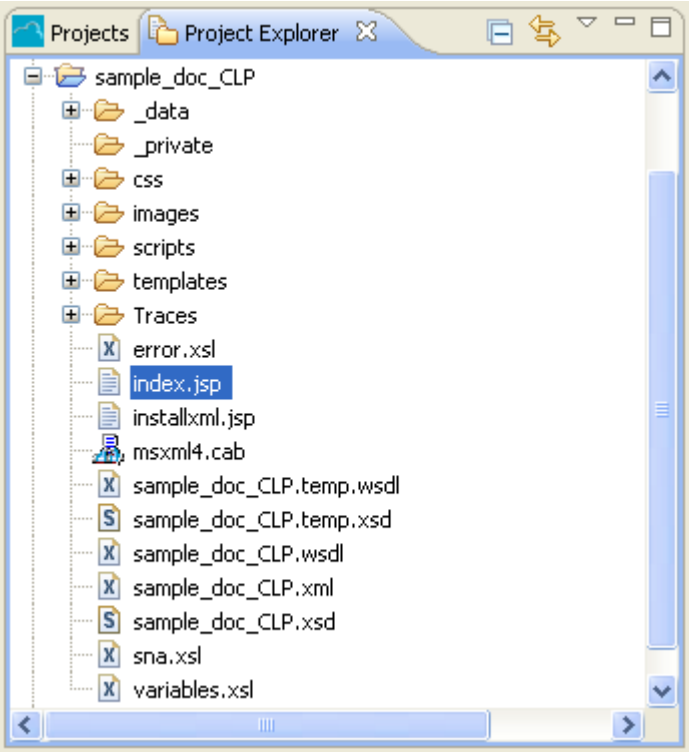


Figure 2 - 19: sample_doc_CLP project files tree structure

2 Open the `index.jsp` and `variables.xsl` files by double-clicking on each file:

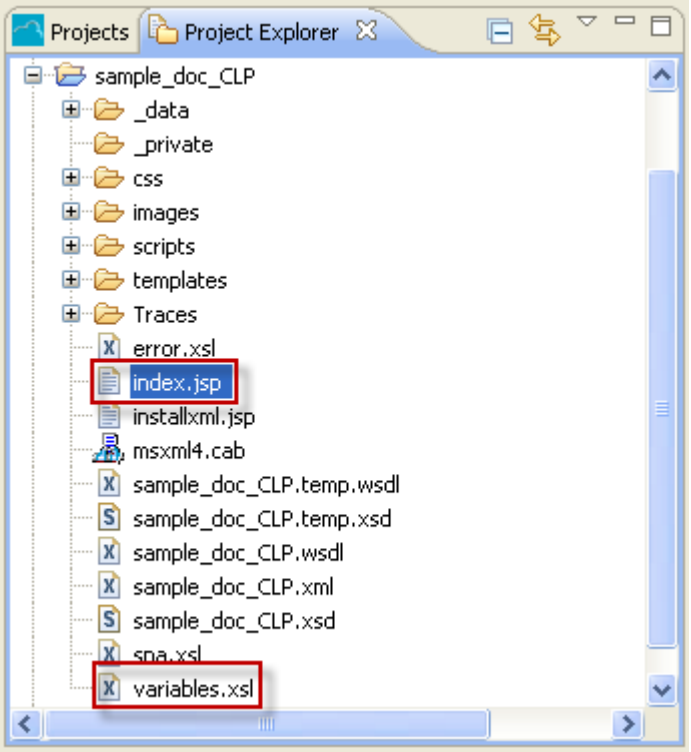


Figure 2 - 20: Webization template files in Project Explorer

Both files open in respective **Document Editors** in the **Connector View** (see Figure 2 - 17 and 2 - 18).

*Depending on your IT environment, `.xsl` files open either in the **Source** or in the **Design** tab of **Document Editors** (see "Connector View / Document Editor" on page 1-21). Should it open in the **Design** tab, please switch to the **Source** tab by clicking on it before continuing.*

We will first edit the `index.jsp` HTML template section so that an image is added in the webization page template as an upper-left positioned banner.

3 In the `index.jsp` file, locate the HTML template section (see Figure 2 - 17).

4 Type in the required HTML `img` tag in the HTML template section using:

- either a relative or an absolute URL as `src` attribute,
- required `style` attribute values for the image to be positioned in the upper left corner of webized screens.

For example: `<img src="../../images/logo-ems-datasheet-300pix.png" style="position: absolute; top: 0px; left: 0px;"/>`

*Use a relative path for the URL if the image is stored in the project **images** folder.*

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

.

5 Save by clicking on the **Save** icon of the CLP application or by pressing the `Ctrl+S` key combination:

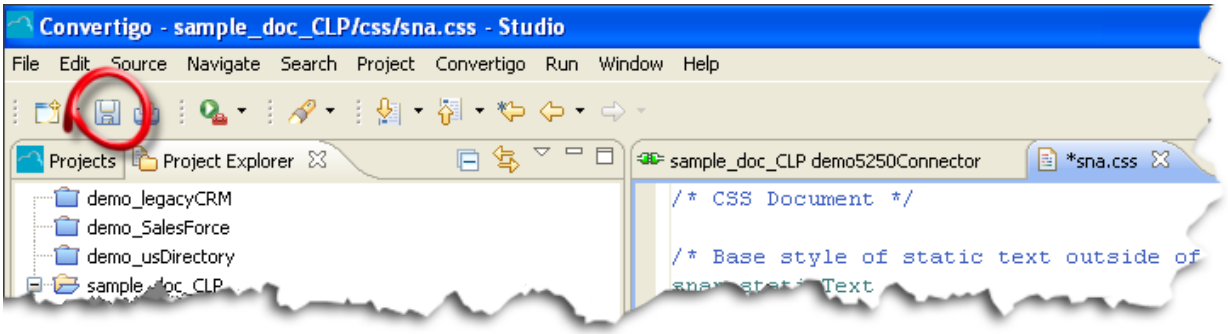


Figure 2 - 21: Studio Save icon

The asterisk disappears, meaning that the file has been saved.

We will now check that the image appears in webized pages.

6 Call the webized screen from a Web browser (see *"To call webized legacy application screens from a Web browser"* on page 2-10):

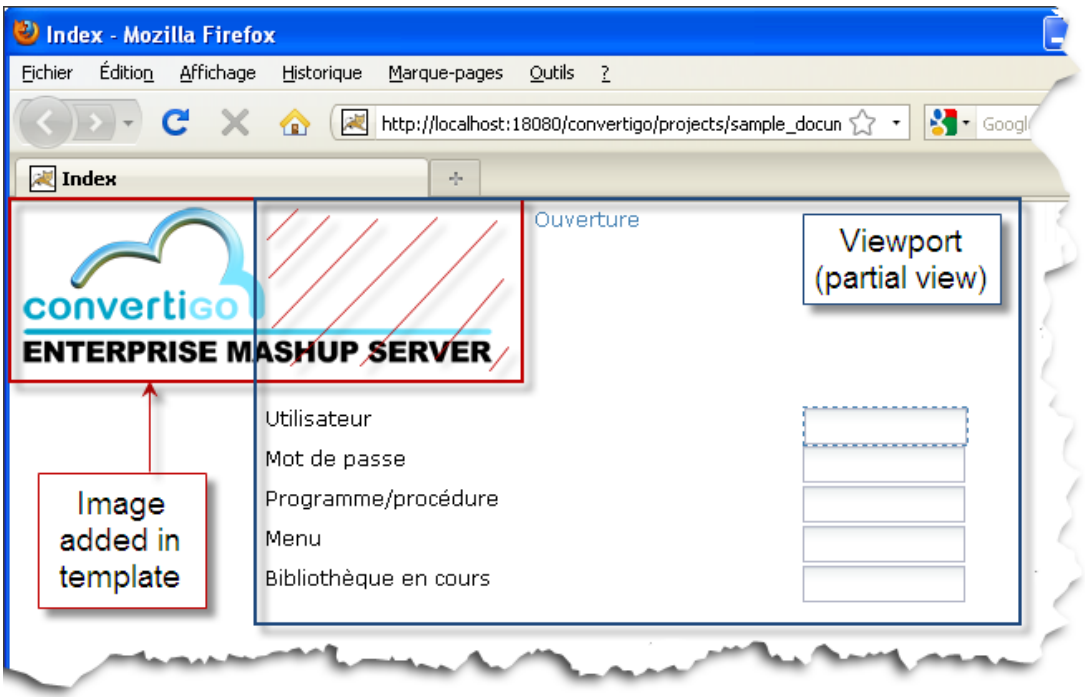


Figure 2 - 22: Image added in webization page template

The specified image appears in the upper-left corner of the webized screen, so the test is conclusive.

The problem is that the added image and viewport ***overlap***.

To make sure that the logo never overlaps with webized elements in the viewport, we will now change the viewport positioning variables.

7 In the `variables.xsl` file, locate the viewport positioning variables:

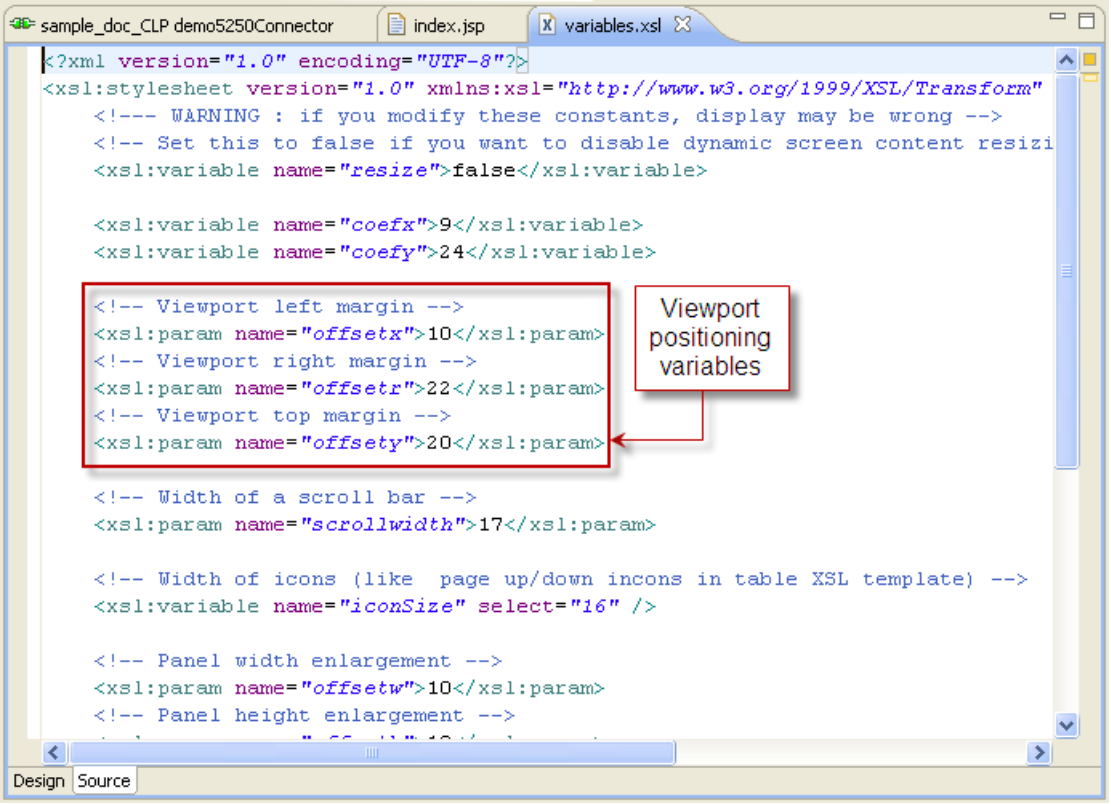


Figure 2 - 23: Viewport positioning variables

8 Increase the `offsety` top margin variable to the required value (for example `120` - with this value, a top margin of 120 pixels is left on top of webized screens for the logo).

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

We will now check that the position of webized elements changes accordingly when webizing screens.

10 In the Web browser, refresh the current page (not pressing the F5 key not to launch the corresponding action in the emulator):

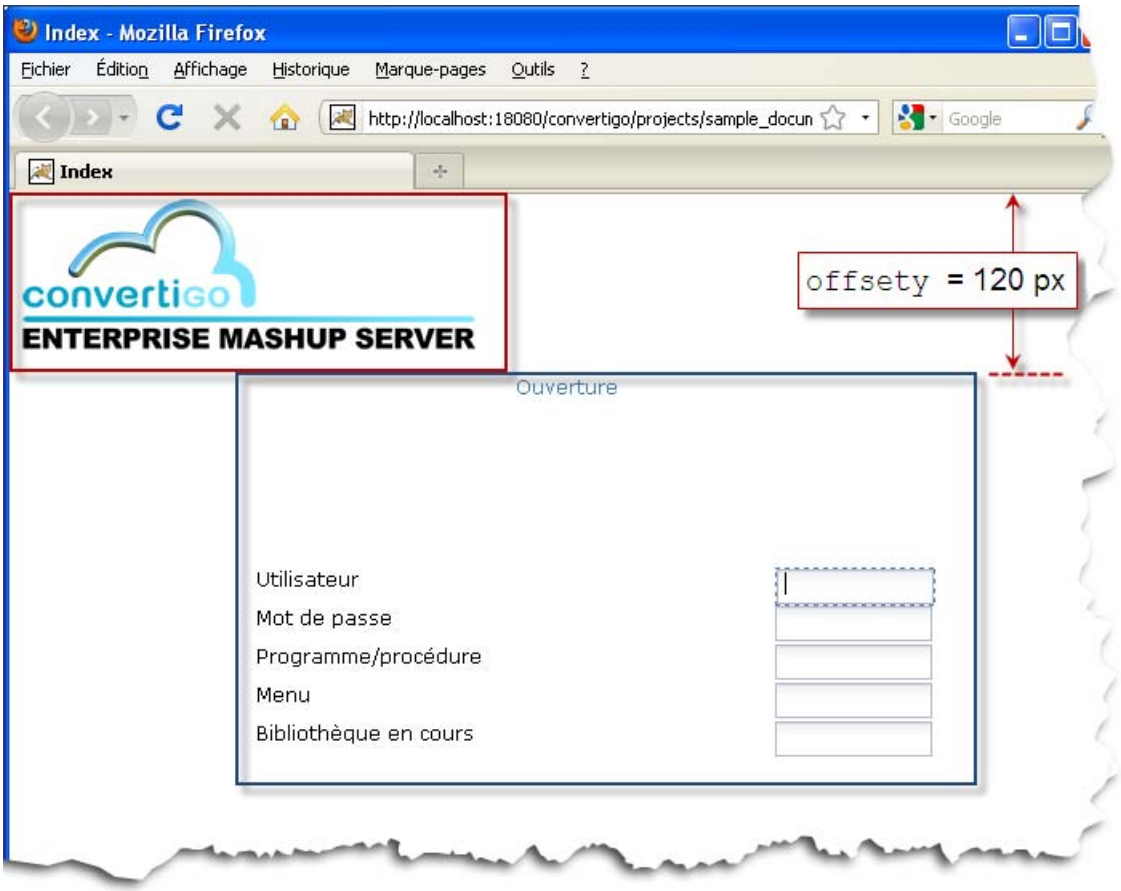


Figure 2 - 24: Webized screen with changed viewport position

> *If changes do not appear, empty the browser cache.*

The logo and webized elements do not overlap anymore.

WHAT COMES NEXT?

Now that a logo has been added to the template, we would like to group action buttons appearing on each legacy screen (see Figure 2 - 27) to the left of the added logo.

To do so, we will:

1 *Set* the `GroupActionButtons` variable (included in the `variables.xsl` file) value determining whether action buttons are grouped to `true`.

2 *Change* the viewport and logo positions so that all elements (logo, action buttons and webized elements) are displayed in a neat layout and do not overlap.

To edit the webization page template to group action buttons

1 In the `variables.xsl` file, locate the `GroupActionButtons` variable definition:

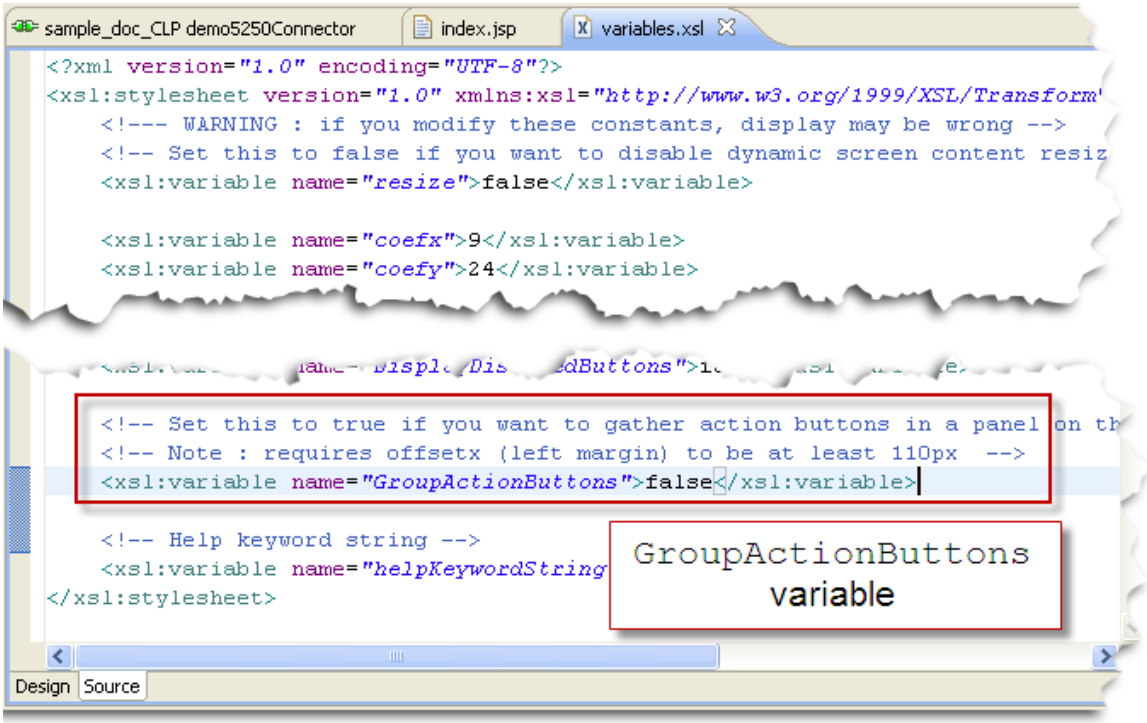


Figure 2 - 25: GroupActionButton variable

2 Set the variable value to `true`.

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

We will now check if action buttons are correctly grouped in the upper left corner of webized screens.

4 Switch back to the **Emulator**:

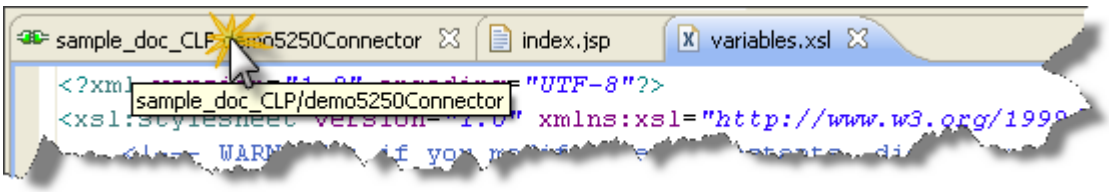


Figure 2 - 26: Selecting connector tab

5 Browse to a legacy screen including action buttons:

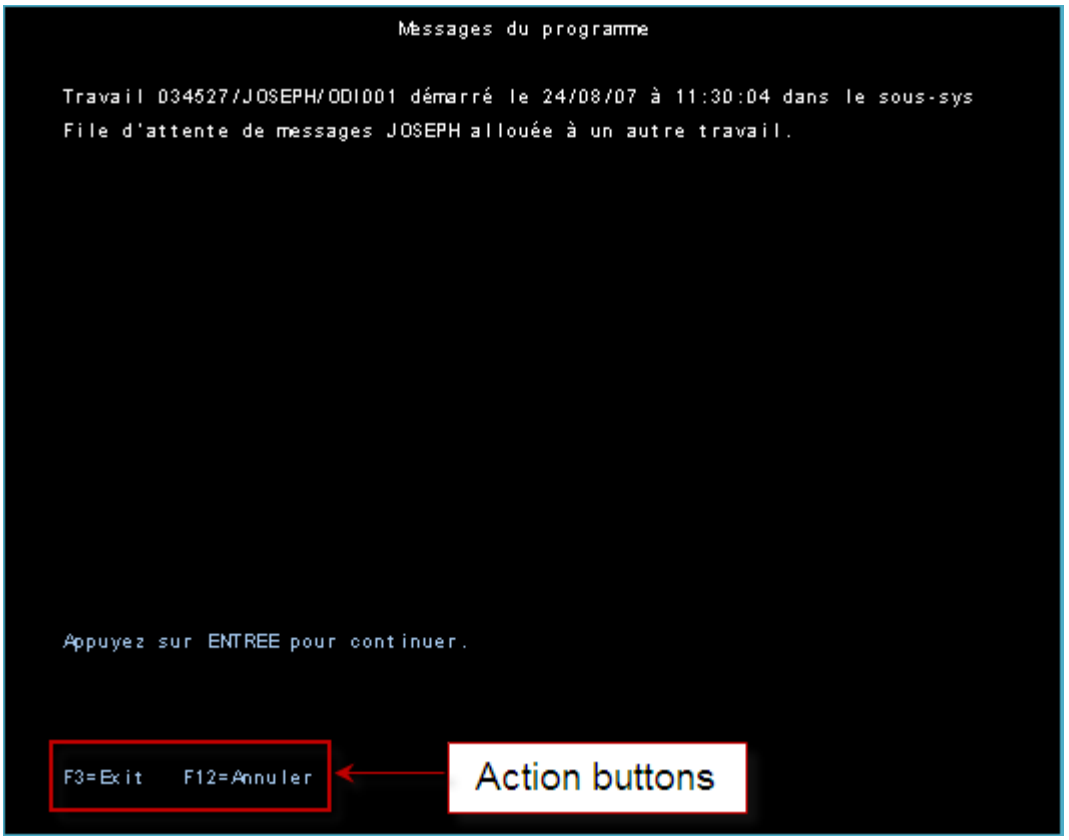


Figure 2 - 27: Legacy screen with action buttons

6 In the Web browser, refresh the current page:

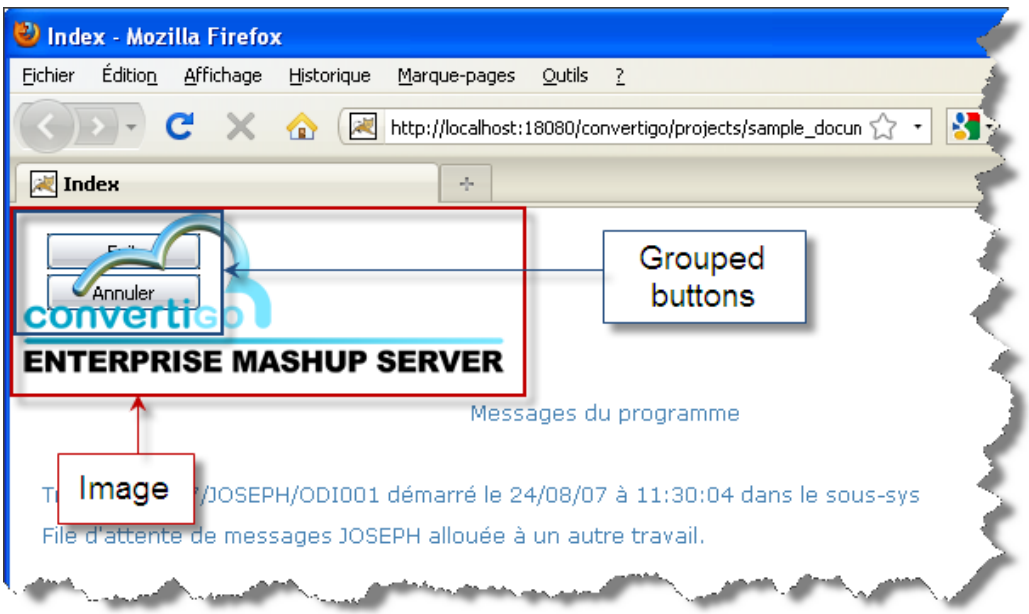


Figure 2 - 28: Webized screen with grouped action buttons

Action buttons appear grouped but they overlap with the logo. In addition, should the webized screen include a high number of buttons, these would also overlap with the viewport

In the `variables.xsl` file, a note is joined to the `GroupActionButtons` variable definition: `"Note : requires offsetx (left margin) to be at least`

`110px"`.

So, for the logo, grouped buttons not to overlap anymore, we will now edit:

- the `offsetx` variable value in the `variables.xsl` so that webized elements appear to the right of the grouped buttons dedicated column,
- the `img` tag definition in the `index.jsp` file HTML template section so that the logo appears further to the right.

7 In the `variables.xsl` file, locate the `offsetx` variable definition.

8 Set the variable value to `110`.

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

10 In the `index.jsp` file HTML template section, set the `img` style attribute `left` value to `120px`.

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

We will now check that action buttons and logo do not overlap anymore.

12 In the Web browser, refresh the current page:

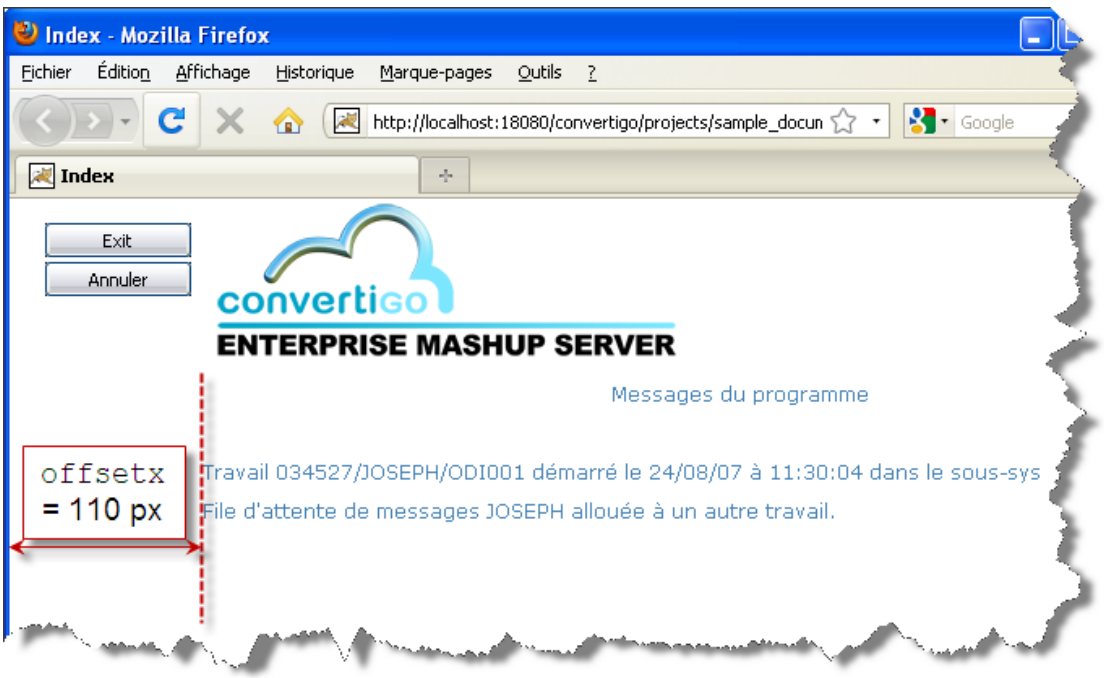


Figure 2 - 29: Webized screen with properly layed out grouped buttons and logo

Action buttons appear grouped in the upper left corner of webized screens together with a logo to the right and the viewport is correctly positioned below these template elements - the webization page template is now properly customized.

WHAT COMES NEXT?

Customizing the webization page template is the best way to obtain a fixed customized template for all webized legacy screens.

You might also want, for all accessed legacy screens, to have specific extraction rules executed. In this case, we recommend that you create specific extraction rules on the default screen class.

This is described in the next section.

2.3.2 Creating Extraction Rules on the Default Screen Class

This second customization method applies when default extraction rules fail to address specific webization needs.

This is the case for example if you need to add an image to the webized screens, since this is not automatically done by default extraction rules.

To do so, we will create a new *Image* extraction rule called `AddImage`.

To create an Image extraction rule

1. Right-click on the default screen class, `DefaultScreenClass` or on the **Extraction rules** folder of `DefaultScreenClass` screen class.
2. Select ***New > Extraction rule***:

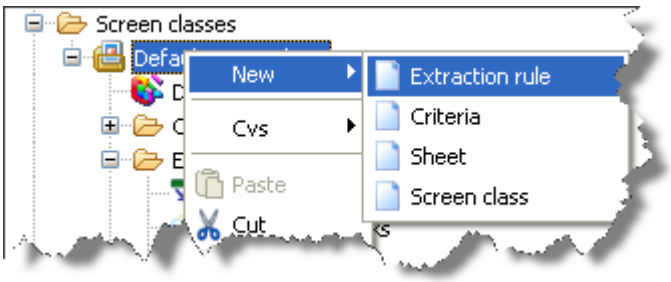


Figure 2 - 30: Creating a new extraction rule

A **New Extraction Rule** wizard opens.

3. Click on **Image**, which is the extraction rule to be used for adding images:

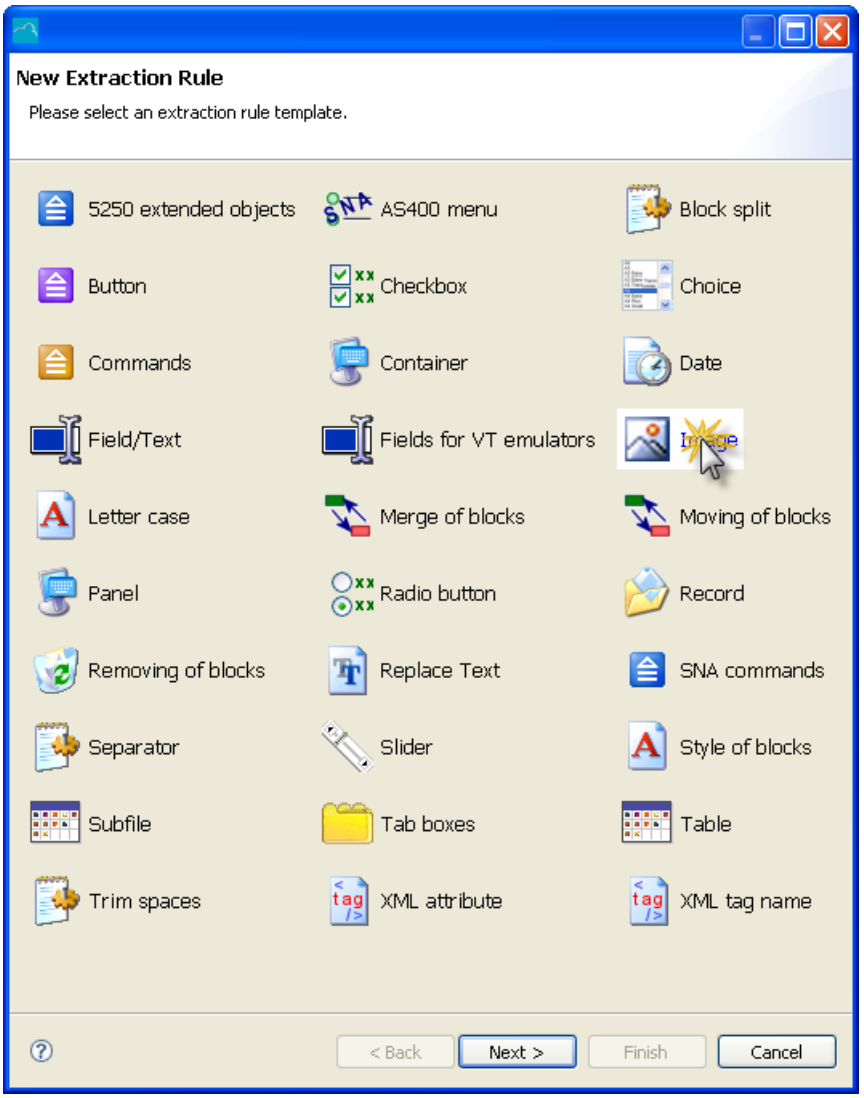


Figure 2 - 31: New Extraction Rule wizard - Image

4 Click **Next**.

5 Type in the name of the new extraction rule (for example `AddImage`):

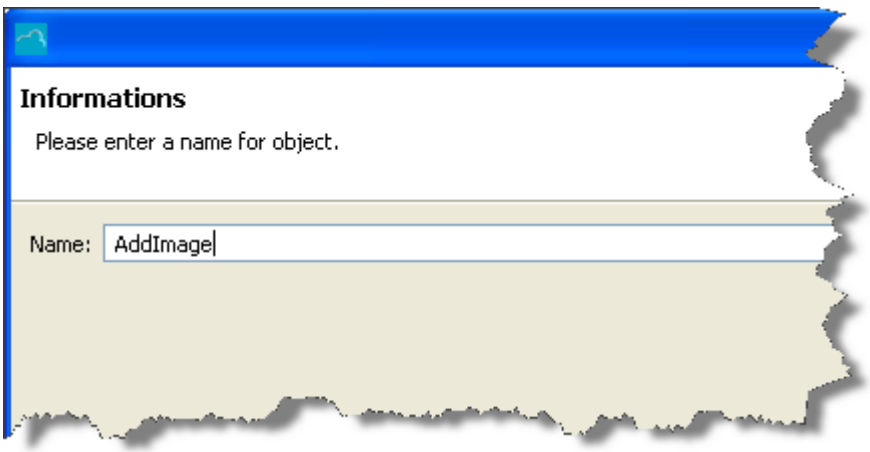


Figure 2 - 32: New extraction rule name - AddImage

6 Click on **Finish**.

The new extraction rule appears in the **Extraction rules** folder of the `DefaultScreenClass` screen class, at the bottom of the extraction rules list:

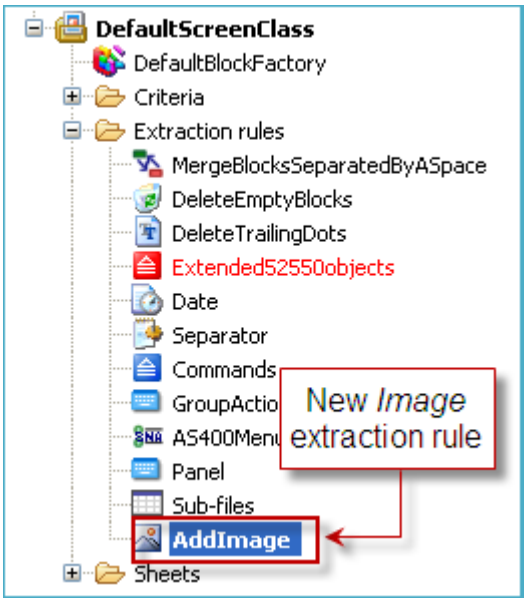
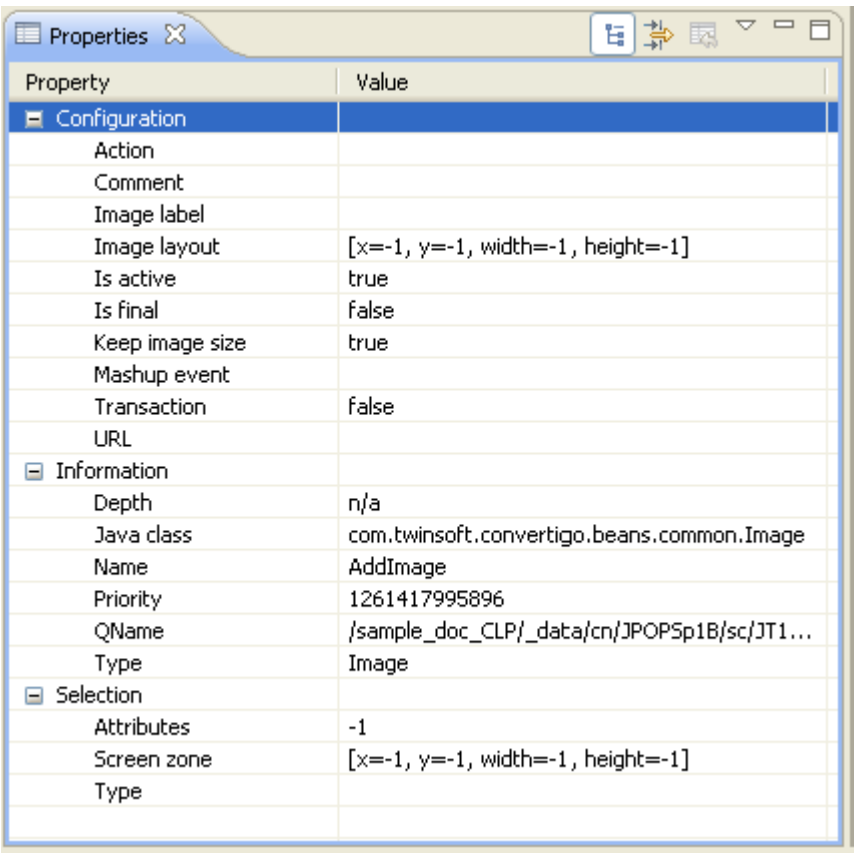


Figure 2 - 33: New extraction rule (AddImage) in default screen class Extraction rules folder

The `AddImage` extraction rule object is now created. We will now set its properties.

7 In the **Projects View**, select the `AddImage` extraction rule.

The **Properties View** is automatically updated:



Properties

Property	Value
Configuration	
Action	
Comment	
Image label	
Image layout	[x=-1, y=-1, width=-1, height=-1]
Is active	true
Is final	false
Keep image size	true
Mashup event	
Transaction	false
URL	
Information	
Depth	n/a
Java class	com.twinsoft.convertigo.beans.common.Image
Name	AddImage
Priority	1261417995896
QName	/sample_doc_CLP/_data/cn/JPOPSp1B/sc/JT1...
Type	Image
Selection	
Attributes	-1
Screen zone	[x=-1, y=-1, width=-1, height=-1]
Type	

Figure 2 - 34: AddImage extraction rule properties

The following table describes these properties:

Table 2 - 2: Image extraction rule properties

Property	Description
Action	If a mainframe action such as `KEY_ENTER`, `KEY_PF3` is specified as value of this property, clicking on the image in the webized application triggers the specified action in the emulator.
Image label	If no image is found, the label specified as value of this property is displayed
Image layout	Position and size of image in screen. Can be set using the icon.
Keep image size	If set to `true`, the image is displayed in its original size. If set to `false`, the image size is as specified in the **Image layout** property.
Transaction	If a transaction is specified, clicking on the image triggers the transaction.
URL	Image URL address. If set as relative URL address, the URL root is the project root. Example: if set as `images/mypic.jpg`, the image is searched in the project **images** sub-folder. If set as `http://www.images.com/mypic.png`, the image is directly loaded from this address.

Assuming that we do not want to display the image in its original size, we will set the **Image layout** property (using the **Link current selection to highlighted property in Properties panel** icon).

We also need to specifiy the image URL address, and will therefore need to set the mandatory **URL** property.

8 Click on the default value of the **Image layout** property. The property name is highlighted and a button appears in the **Value** column:

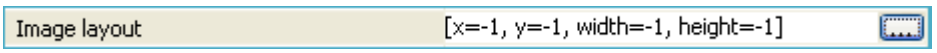


Figure 2 - 35: Image layout property

To set the **Image layout** property, you can either:

A. click on the button, set **Top**, **Width**, **Right**, **Left** values in the **Screen zone** window that appears and click **OK**:

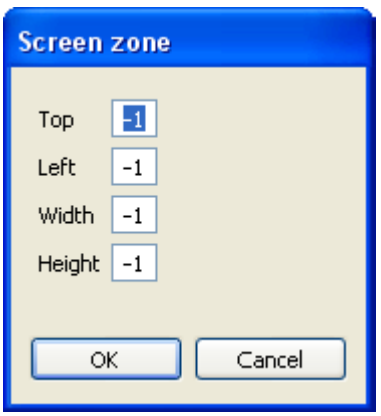


Figure 2 - 36: Screen zone window

B. OR select the screen zone in the **Emulator** then click on the **Link current selection to highlighted property in Properties panel** icon :

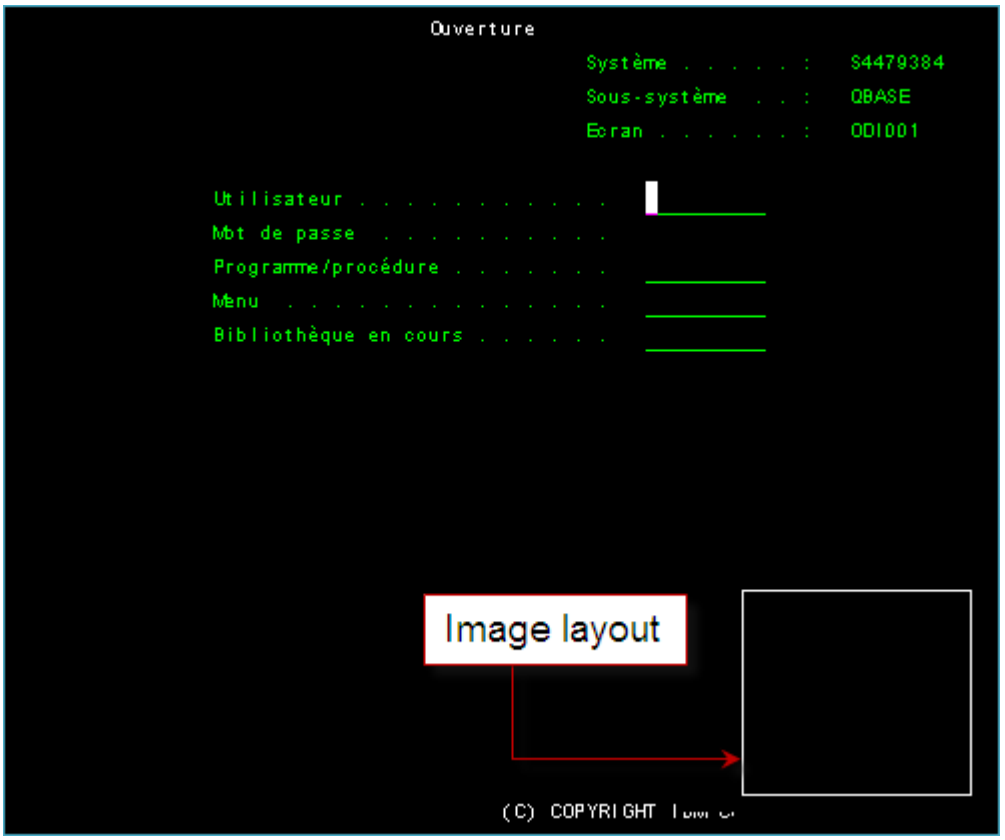


Figure 2 - 37: Selecting the image layout in the Emulator

Figure 2 - 38: Linking selection to Image layout property

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

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

Image layout	[x=65, y=19, width=15, height=4]

Figure 2 - 39: Automatic setting of Image layout property value

The **Image layout** property value is now set. We will now set the URL value.

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

The property is highlighted then a cursor appears.

11 Type in the image URL address.

Images can be addressed using either:

- a relative URL if the image is stored in the **images** folder of the project (for example, `images/convertigo-small.gif`) or in the Convertigo application,

- or an absolute URL if the image is stored elsewhere.

In this project, the specified URL is `images/convertigo-small.gif`.

12 Press `Enter`.

The property is updated.

The `AddImage` extraction rule is now created and completely set.

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

We can now check that the newly created extraction rule returns expected results.

To check that the Image extraction rule adds the image in the XML output / webized HTML

Assuming that the **Emulator** displays the login screen of our sample project, click on the **Generate XML** icon in the **Emulator toolbar**.

Expected result: Default as well as user-defined (in this case, `AddImage`) extraction rules are executed and the **Browser** output displays a Webized legacy application screen including the image specified in the **URL** extraction rule property, stretched in the screen zone set as value of the **Image layout** property.

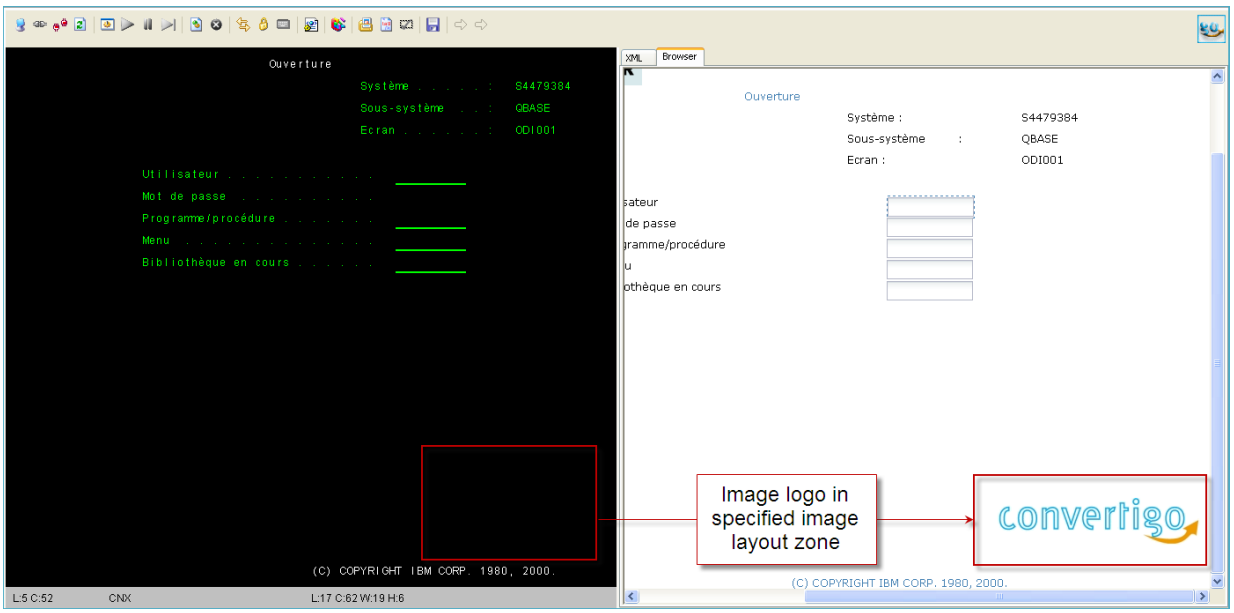


Figure 2 - 40: Browser output after execution of default and AddImage extraction rules

This figure shows that both extraction rules have been executed as expected.

WHAT COMES NEXT?

As mentioned before, creating a specific extraction rule is one way among others (Table *"Webization customization methods"* on page 2-11) to customize webized legacy application screens. Another way is by editing of the `sna.css` style sheet.

The next section deals with such customization.

2.3.3 Editing the Generic Webization CSS

Any CLP project includes a generic Webization cascading style sheet (CSS) called `sna.css`.

The `sna.css` CSS can be edited for webization customization purpose. For example, you can decide whether to change the font size and/or font family of static text (blocks of `static` type), the space between table cells, the style of command panels, etc.

For more information on block types, see Table "Block standard XML attributes" on page 1-9.

After the `sna.css` has been modified, all extracted blocks concerned by modifications (through their `type` attribute) are webized accordingly without distinction.

This following procedure describes how to edit the `sna.css` CSS in order to:

- *Increase* the text size of static text (for example from 12 to 16 pts).
- *Replace* the color of all white text included in legacy application screens with deep pink text.

To edit the sna.css cascading style sheet

1 Click on the **Project Explorer** tab in the **Projects View**.

The `sample_doc_CLP` project files tree structure is displayed:

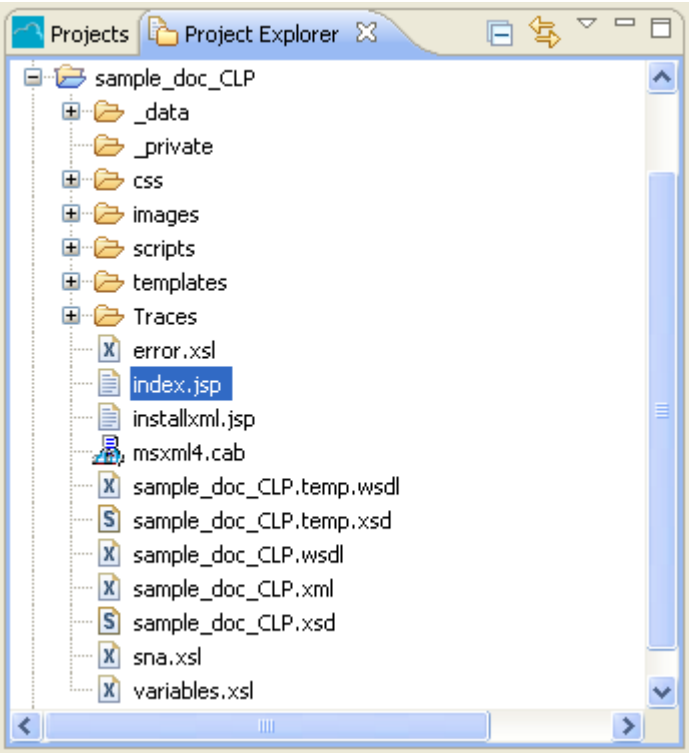


Figure 2 - 41: sample_doc_CLP project files tree structure

2 Expand the **css** folder by clicking on the ⊞ symbol to the left of it:

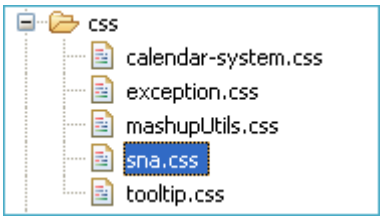


Figure 2 - 42: Expanded css folder

3 Double-click on `sna.css` or right-click on the file and select **Open** in the drop-down menu.

The file opens in an **Editor**:

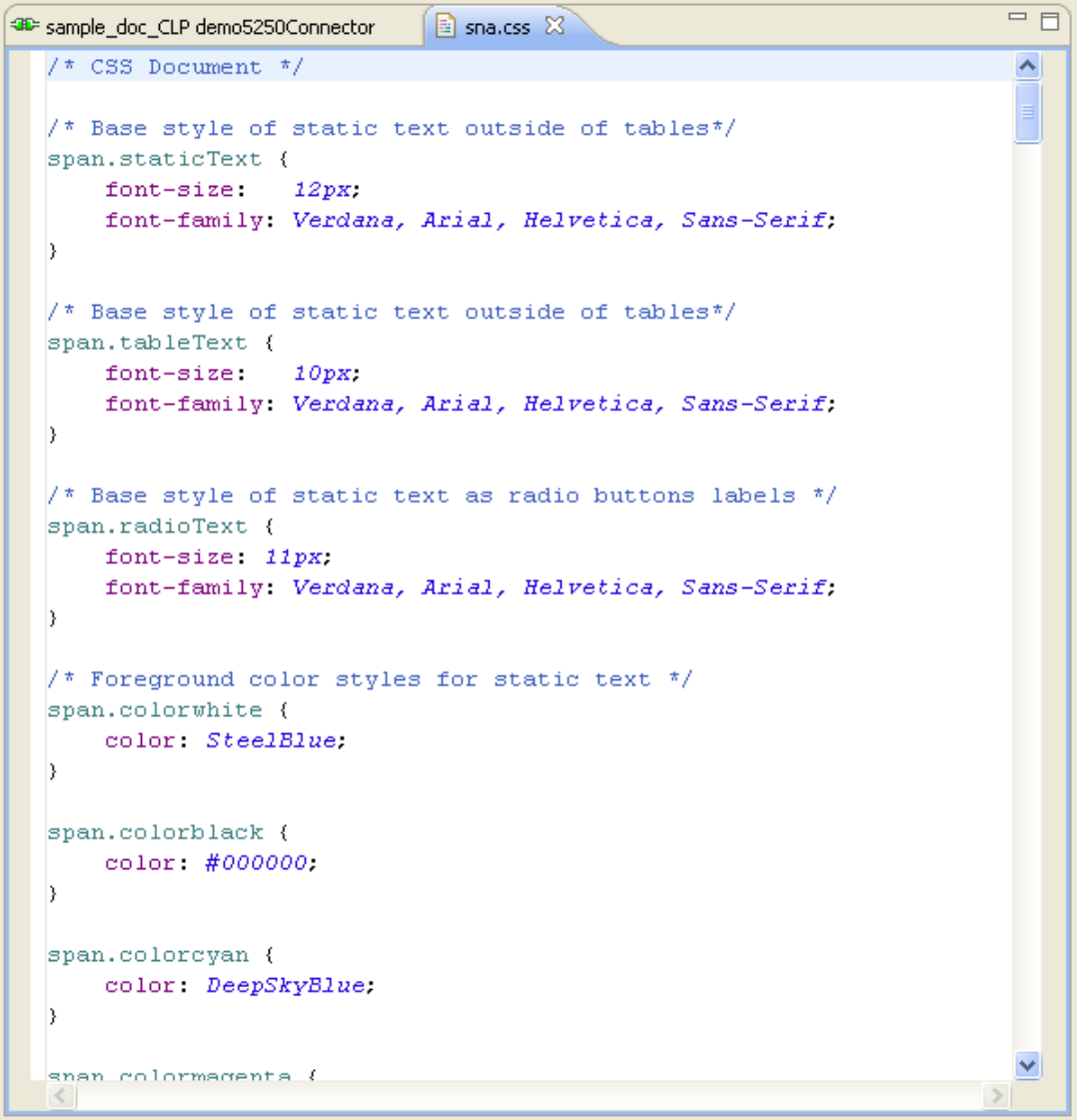


Figure 2 - 43: Generic Webization cascading style sheet (sna.css)

We want first to change the font size of static text, so our editing will concern the `font-size` property of the `span.staticText` selector part of this file, which is commented as "Base style of static text outside of tables" in the CSS:

```
/* Base style of static text outside of tables*/
span.staticText {
    font-size:   12px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
}
```

Figure 2 - 44: Style definition - Static text outside tables

We will edit this style definition and see how it impacts the legacy application screen Webization.

4 Place your cursor at the required position in the style definition:

```
/* Base style of static text outside of tables*/
span.staticText {
    font-size:   12px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
}
```

Figure 2 - 45: Editing font size (1)

5 Replace `12px` with `16px`:

```
/* Base style of static text outside of tables*/
span.staticText {
    font-size:   16px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
}
```

Figure 2 - 46: Editing font size (2)

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

6 Save by clicking on the **Save** icon of the CLP application or by pressing the `Ctrl+S` key combination:

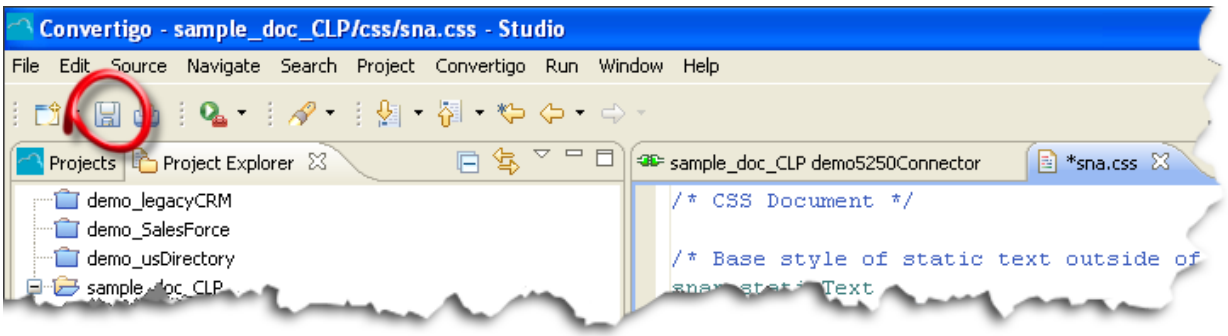


Figure 2 - 47: Studio Save icon

The asterisk disappears, meaning that the file has been saved.

We will now turn all white static text included in a legacy application screen into deep pink text. To this end, we will modify the `span.colorwhite` definition style included in the `sna.css` CSS, which is commented as "Foreground color styles for static text":

```
/* Foreground color styles for static text */
span.colorwhite {
    color: SteelBlue;
}
```

Figure 2 - 48: Style definition - Foreground color styles for static text

7 Place your cursor at the required position in the style definition:

```
/* Foreground color styles for static text */
span.colorwhite {
    color: SteelBlue;
}
```

Figure 2 - 49: Editing foreground color (1)

8 Replace `SteelBlue` with `DeepPink`:

```
/* Foreground color styles for static text */
span.colorwhite {
    color: DeepPink;
}
```

Figure 2 - 50: Editing foreground color (2)

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

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

The asterisk disappears, meaning that the file has been saved.

We wil check that changes have been taken into account by first launching an XML generation then selecting the **Browser** tab of the **Connector View** to see the results of Webization customization.

10 Click on the connector tab (for example `sample_doc_CLP demo5250connector`).

11 Click on the **Generate XML** icon .

12 Select the **Browser** tab of the **Connector View**.

The font size of static text has been increased by 4 pts (to 16 pts), and the white text included in the legacy application screen, which appeared in steel blue before customization, now appears in deep pink:

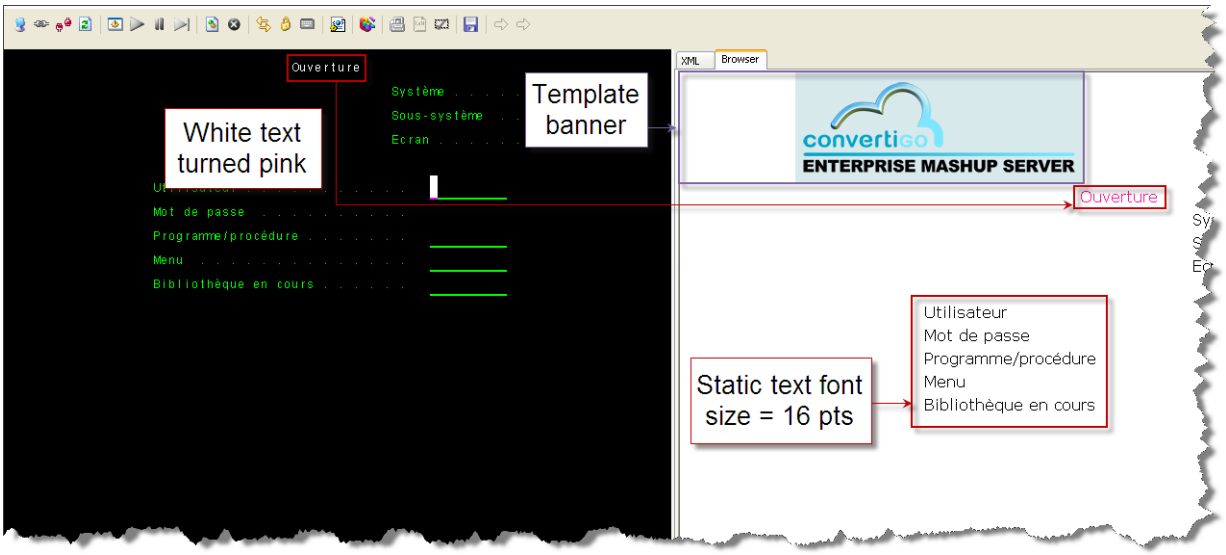


Figure 2 - 51: Webized legacy application screen after page template and CSS customization

The logo added as template banner (see "Customizing the Webization Page Template" on page 2-12) also appears. No webize action button appears because none is included in the legacy screen.

If changes do not appear in the internal browser (because of file cache), call the webized screen from an external Web browser (see "To call webized legacy application screens from a Web browser" on page 2-10).

WHAT COMES NEXT?

You are now familiar with the edition of the generic webization CSS and know how to modify format style definitions before webizing legacy screens.

As mentioned in Table *"Webization customization methods"* on page 2-11, another method for customizing the webization process is to define screen classes.

This is described in the next section.

2.3.4 Defining Screen Classes and Screen Class-Specific Extraction Rules

After the introduction (see *"Introduction"* on page 2-33), this section describes how to define screen classes and associate extraction rules with defined screen classes.

Two situations are considered here:

- Creating Extraction Rules on Properly Webized Screen - a screen is accessed, it is correctly webized "as is" but the user needs a specific processing (for example, delete text blocks);
- Creating Extraction Rules on Improperly Webized Screens - a screen is accessed, it includes screen elements that default extraction rules cannot process and the webization is incomplete.

INTRODUCTION

Defining a screen class means selecting a screen element - for example a text string - defining a screen in a unique manner. The selected element is called *detection criterion*.

When CLP accesses a screen, detection criteria are checked. If all criteria defined for a screen class match the screen elements, then CLP considers that the accessed screen belongs to this screen class. The screen class is said *"detected"*.

Extraction rules associated with the detected screen class as well as default extraction rules are then executed, extracting data in XML format. The legacy application screen is then dynamically *Webized* as is the case with screens detected as belonging to the default screen class (see *"Dynamically webizing legacy application screens"* on page 2-7).

CREATING EXTRACTION RULES ON PROPERLY WEBIZED SCREEN

Let's consider the following case: your legacy application *login screen* is accessed. It includes a text block that you want removed from webization.

This is achievable only if, prior to accessing the login screen, a specific extraction rule is created for the text block to be removed from extracted data.

To this end, a specific screen class (called `LoginScreen`) as well as a *Removing of blocks* extraction rule (called `DeleteTopBlocks`) will be created.

The following table describes the `LoginScreen` screen class:

Table 2 - 3: `LoginScreen` screen class

Name	Description	Detection criterion
`LoginScreen`.	Corresponds to the legacy application login screen accessed in the sample project.	Of **Find string** type. Text string is "`Ouverture`".

This screen class is a child to the `DefaultScreenClass` screen class (see *"Screen Class"* on page 1-5). As such, it inherits its criterion (the *emulator technology*), default extraction rules and style sheet templates.

The following figure shows which element of the legacy application screen has been chosen in this sample project as detection criterion for the `LoginScreen` screen class:

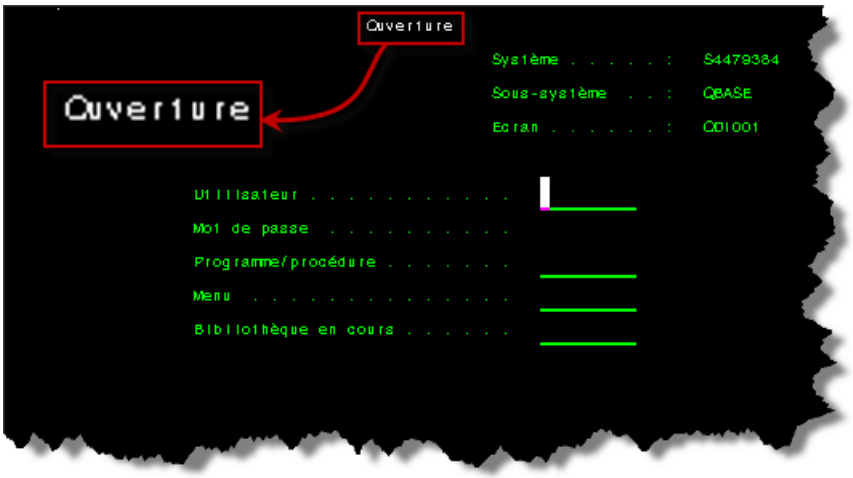


Figure 2 - 52: "Ouverture" text string (detection criterion of LoginScreen screen class)

The text string "Ouverture" ("Opening" in English) is the screen title, this is why it has been selected as detection criterion for the `LoginScreen` screen class.

The following procedure describes how to create the `LoginScreen` screen class.

To create a new screen class

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

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

Figure 2 - 53: Login screen of emulated legacy application

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

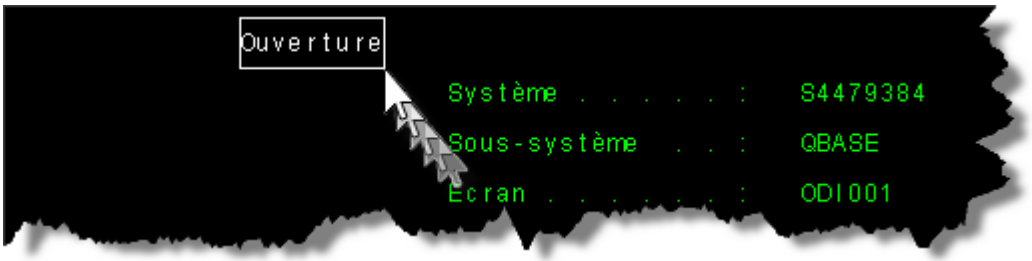


Figure 2 - 54: Selecting a detection criterion on screen

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

3 Click on the **New ScreenClass** icon in the **Emulator toolbar**.

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

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

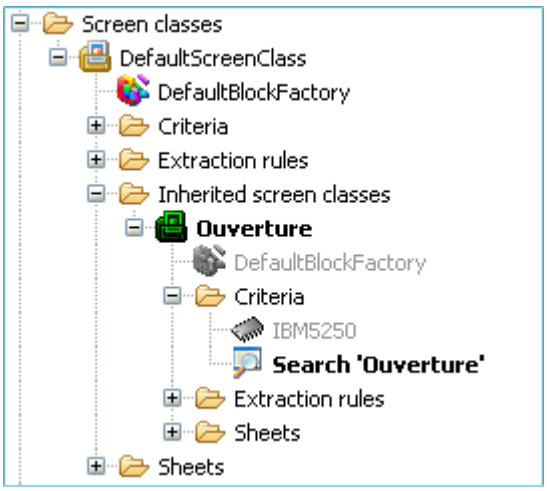


Figure 2 - 55: Newly created screen class and associated detection criterion

Screen class names can be changed to reflect more clearly the screen class that they represent. For example, in our project, we can decide to rename the `Ouverture` screen class, which corresponds to a login screen, into `LoginScreen`.

To rename a screen class

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

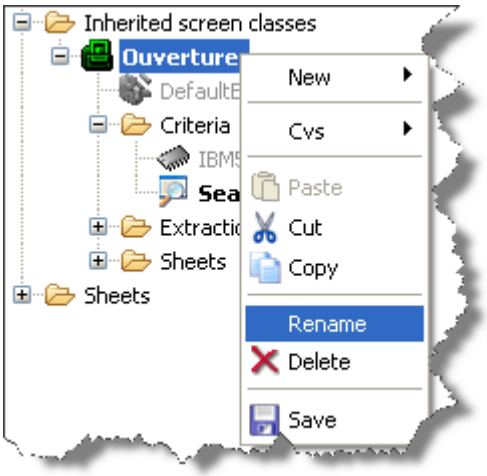


Figure 2 - 56: Renaming a screen class

The screen class name is highlighted:

.

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

3 Press `Enter`.

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

Remember to save your project on a regular basis.

The newly created screen class is now renamed.

It has inherited its parent screen class (`DefaultScreenClass`) criterion, extraction rules and sheets.

These inherited elements appear greyed in the **Criteria**, **Extraction rules** and **Sheets** folder of the screen class:

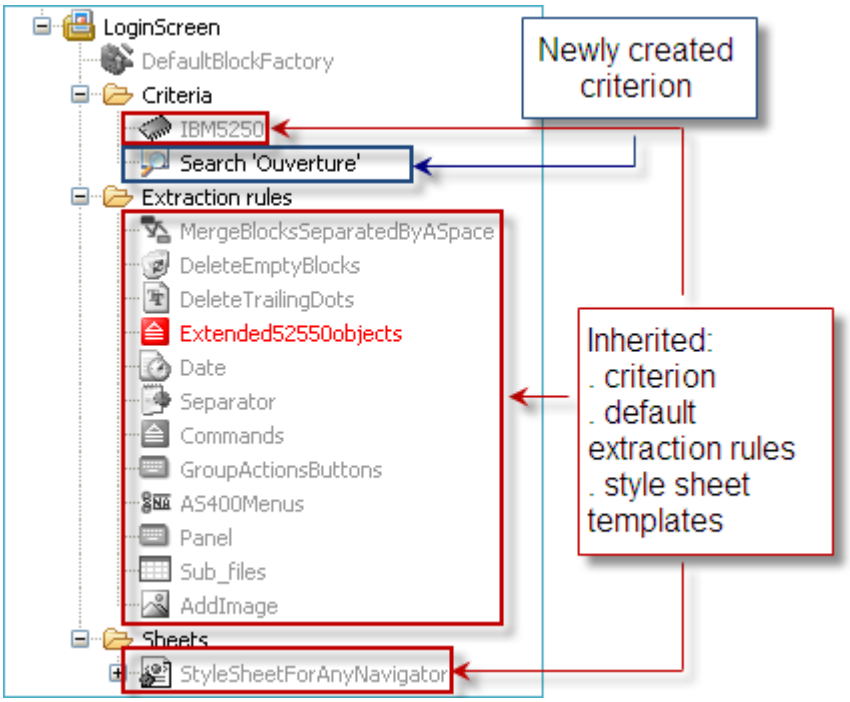


Figure 2 - 57: Inherited and new elements

The `LoginScreen` screen class is now created.

WHAT COMES NEXT?

We will now check that, when accessed, this screen is detected as belonging to the defined screen class, `LoginScreen`.

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

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

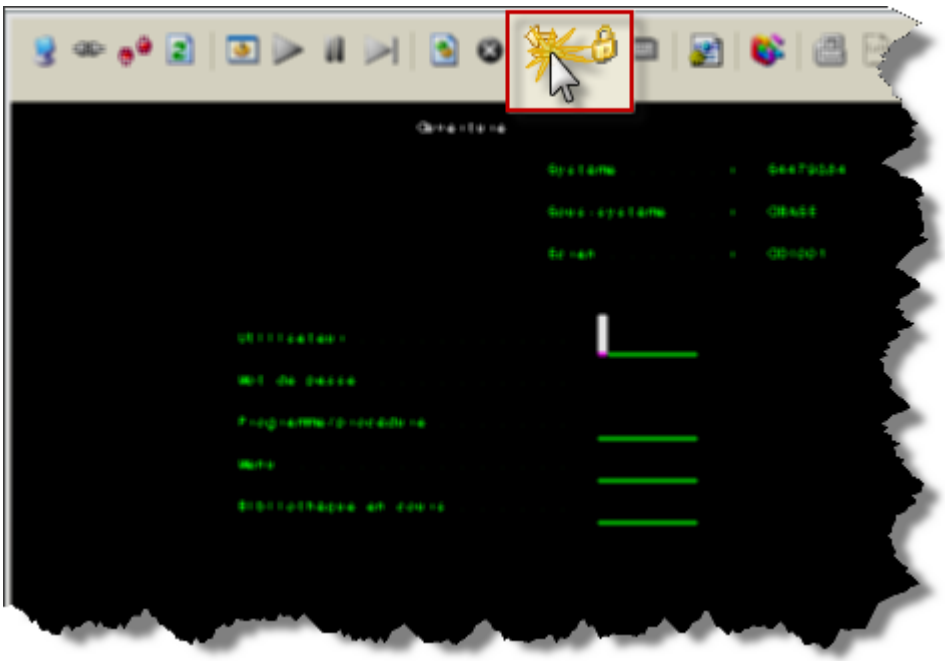


Figure 2 - 58: Clicking Go to current screen class icon

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

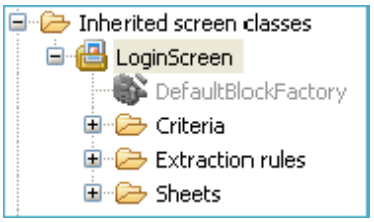


Figure 2 - 59: Highlighted screen class after Go to current screen class object command

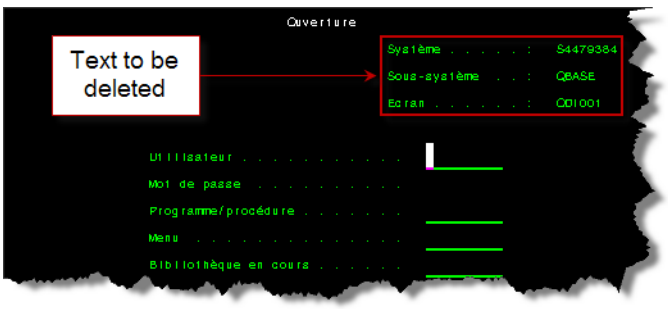
WHAT COMES NEXT?

When a screen is detected as belonging to a given screen class, associated extraction rules (either default or user-defined) are executed.

In our case, we want a *Removing of blocks* extraction rule to be executed in addition to default extraction rules on detection of the `LoginScreen` screen class.

The next step therefore consists in creating and setting the `DeleteTopblocks` extraction rule, which is described in the table below:

Table 2 - 4: `DeleteTopBlocks` extraction rule

Extraction rule name	Description
`DeleteTopBlocks`	Extraction rule type: *Removing of blocks*. Removes text blocks located between login screen title ("Ouverture") and login fields: 

The procedure for creating and setting the *Removing of blocks* extraction rule is described below.

To create a Removing of blocks extraction rule

1. Right-click on the required screen class (or **Extraction rules** screen class folder): for example `LoginScreen`.
2. Select ***New > Extraction rule***:

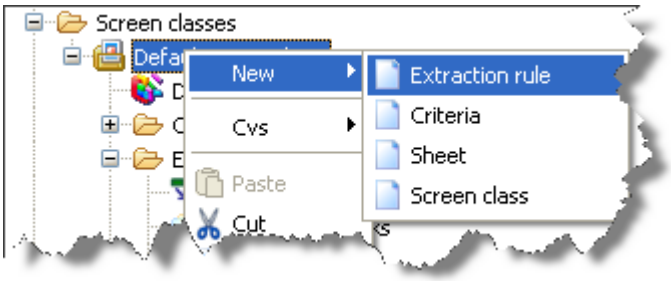


Figure 2 - 60: Creating new extraction rule

A **New Extraction Rule** wizard opens.

3 Click on **Removing of blocks**, which is the extraction rule to be used for removing blocks:

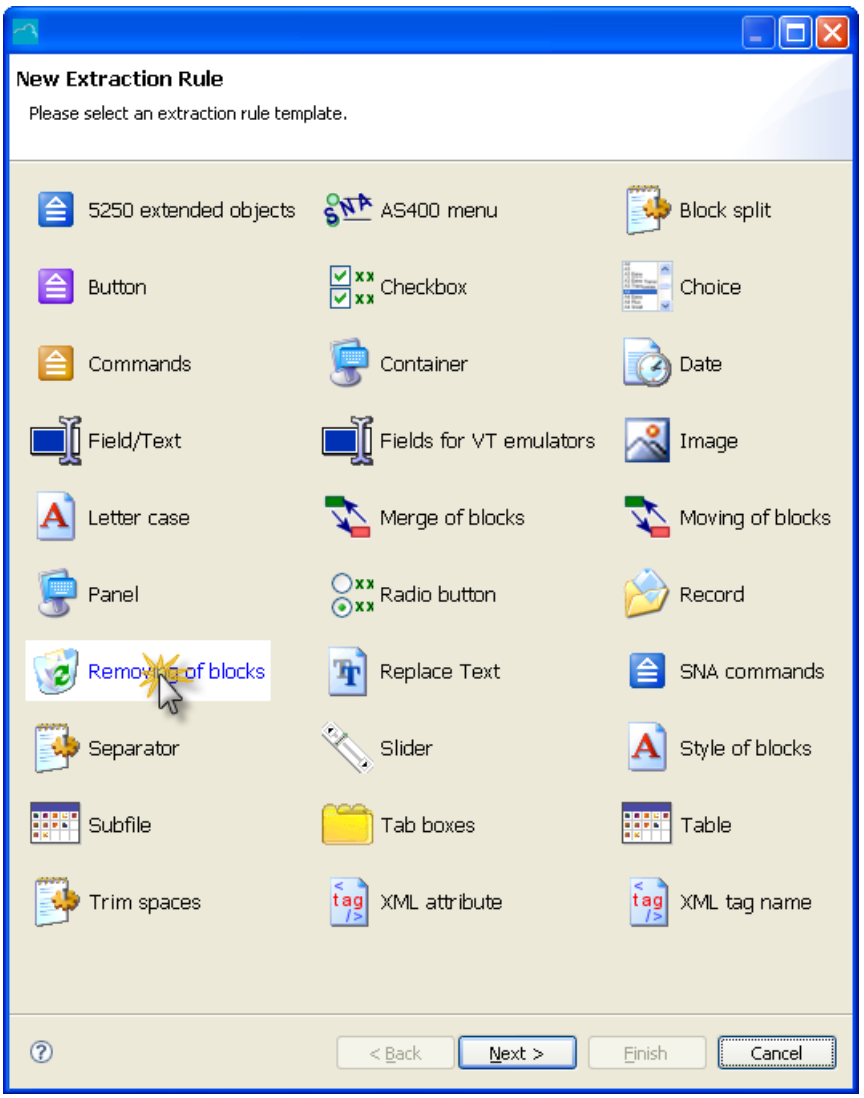


Figure 2 - 61: New Extraction Rule wizard - Removing of blocks

4 Click **Next**.

5 Type in the name of the new extraction rule (for example `DeleteTopBlocks`):

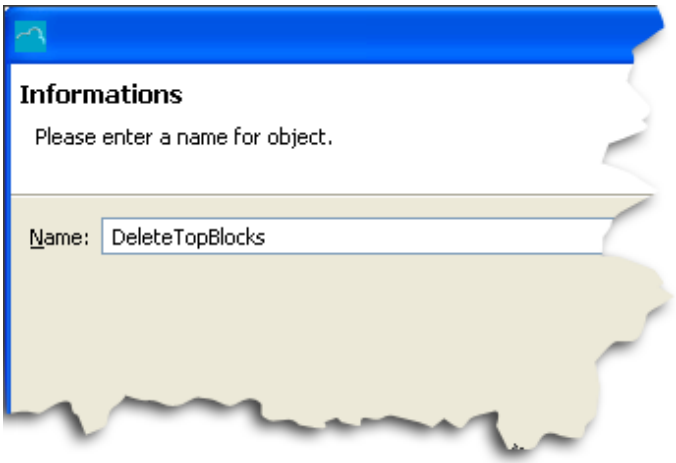


Figure 2 - 62: New extraction rule name - DeleteTopBlocks

6 Click on **Finish**.

The new extraction rule appears in the **Extraction rules** folder of the `LoginScreen` screen class, at the bottom of the extraction rules list:

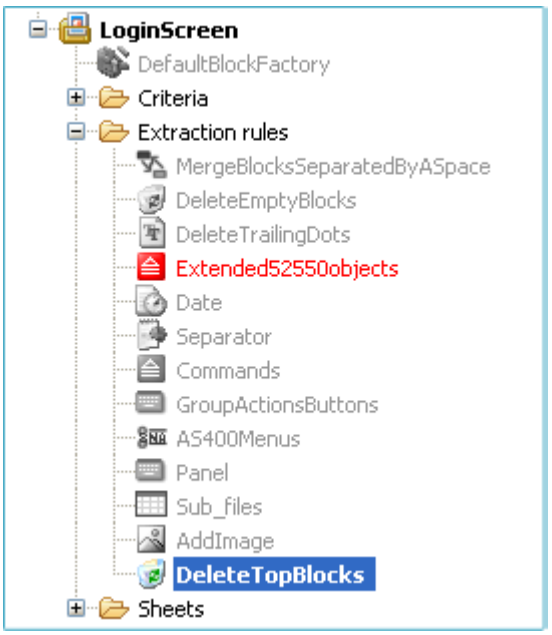


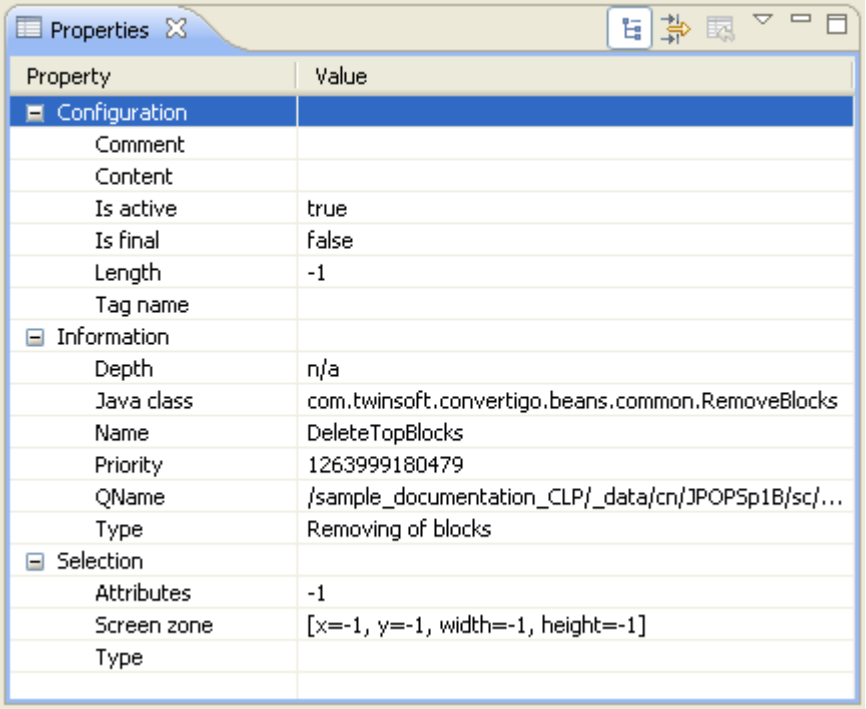
Figure 2 - 63: New extraction rule (DeleteTopBlocks) in screen class Extraction rules folder

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

The `DeleteTopBlocks` extraction rule object is now created. We will now set the extraction rule **Screen zone** parameter defining the screen zone to which the rule applies.

8 In the **Projects View**, select the `DeleteTopBlocks` extraction rule.

The **Properties View** is automatically updated:



Properties

Property	Value
Configuration	
Comment	
Content	
Is active	true
Is final	false
Length	-1
Tag name	
Information	
Depth	n/a
Java class	com.twinsoft.convertigo.beans.common.RemoveBlocks
Name	DeleteTopBlocks
Priority	1263999180479
QName	/sample_documentation_CLP/_data/cn/JPOPSp1B/sc/...
Type	Removing of blocks
Selection	
Attributes	-1
Screen zone	[x=-1, y=-1, width=-1, height=-1]
Type	

Figure 2 - 64: DeleteTopBlocks extraction rule properties

9 Click on the default value of the **Screen zone** property. The property name is highlighted and a button appears in the **Value** column:

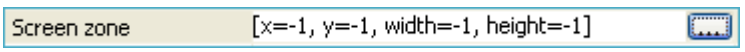


Figure 2 - 65: Screen zone property

To set the **Screen zone** property, you can either:

A. click on the button, set **Top**, **Width**, **Right**, **Left** values in the **Screen zone** window which appears and click **OK**:

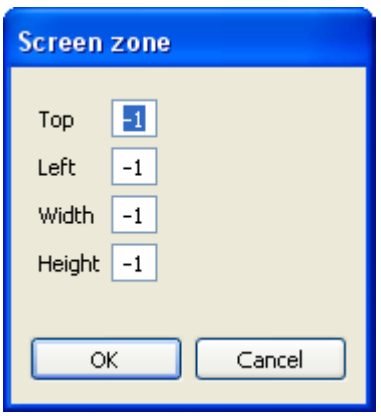


Figure 2 - 66: Screen zone window

B. OR select the required screen zone (that is to say, the zone including the text to be extracted) in the **Emulator** view then click on the **Link current selection to highlighted property in Properties panel** icon :

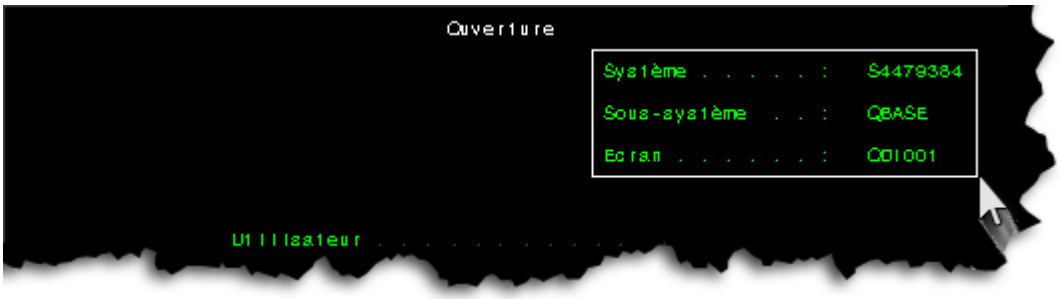


Figure 2 - 67: Selecting the screen zone in Emulator

Figure 2 - 68: Linking selection to Screen zone property

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

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

Screen zone	[x=45, y=1, width=34, height=3]

Figure 2 - 69: Automatic setting of Screen zone property value

The `DeleteTopBlocks` extraction rule is now created and properly set.

Other properties such as **Content** or **Length** can also be set for the **Removing of blocks** extraction rule:

Table 2 - 5: *Removing of blocks* extraction rule properties

Property	Description
Content	Blocks with a content identical to the **Content** property value are removed.
Length	Specifies the length of the blocks to be removed. Starting and ending spaces in blocks are ignored. Not applied if set to `-1`. If set to `0`, blocks containing only spaces are deleted.

We can now check that the newly created extraction rule returns expected results.

To check that the Removing of blocks extraction rule removes selected blocks

Assuming that the **Emulator** displays the login screen of the legacy application, click on the **Generate XML** icon in the **Emulator toolbar**.

Expected result: Extracted blocks located within the `DeleteTopBlocks` screen zone are deleted from the **XML** output. The selected text zone does not appear in the final HTML page in the **Browser** output.

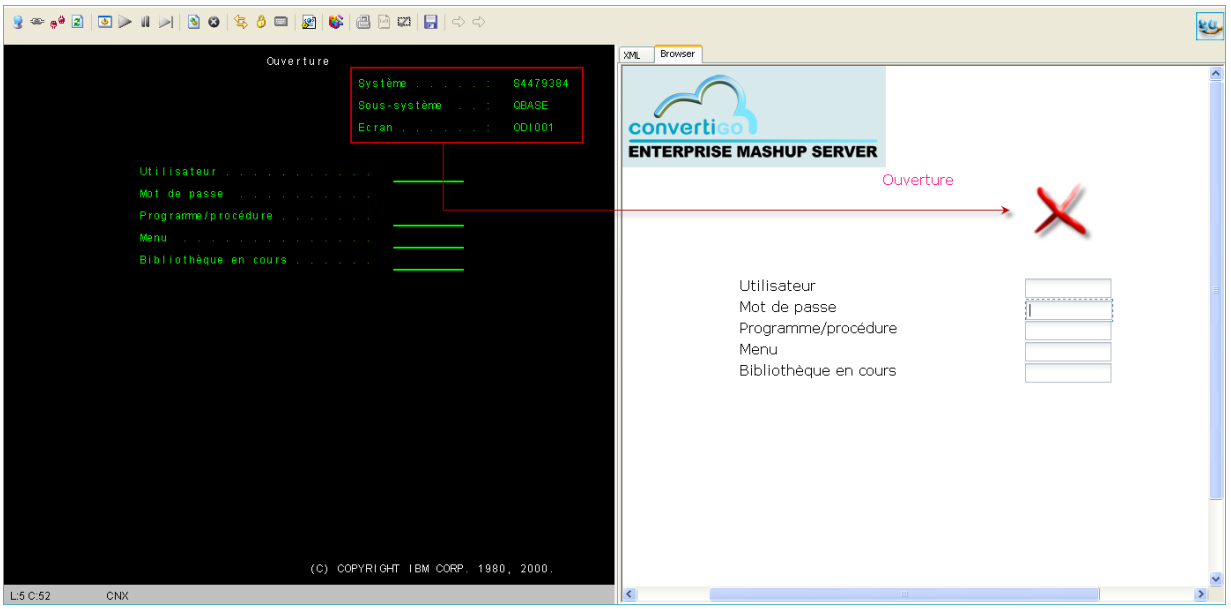


Figure 2 - 70: Browser output after execution of default and DeleteTopBlocks extraction rules

The logo added as template banner (see "Customizing the Webization Page Template" on page 2-12) also appears. No webized action button appears because none is included in the legacy screen. The webized static text is formatted as changed in the `sna.css` file (see "Editing the Generic Webization CSS" on page 2-28).

In the **Browser** output, the text block is removed - the test is conclusive.

Creating Extraction Rules on Improperly Webized Screens

Let's now consider that the following legacy application screen is accessed:

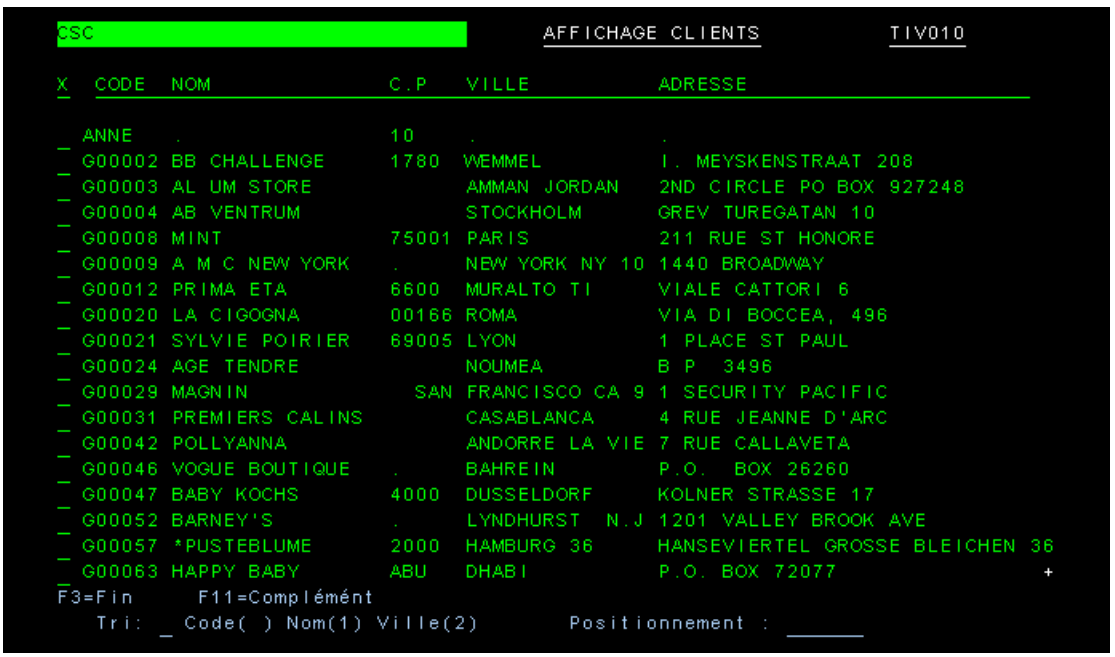


Figure 2 - 71: Customer list screen

It includes a table that default extraction rules fail to webize correctly:

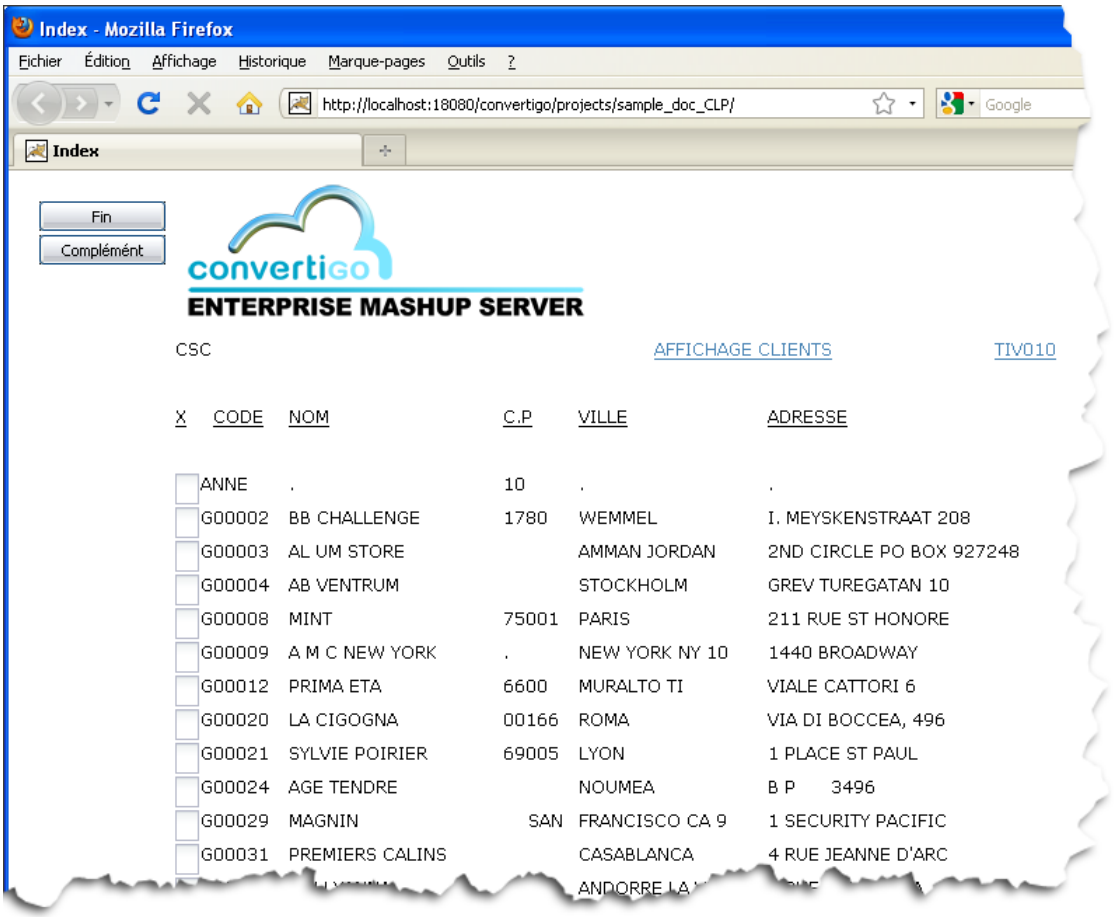


Figure 2 - 72: Webized screen with improperly webized table

To properly webize this screen, a specific extraction rule must therefore be created, and executed on detection of the required screen (see Figure 2 - 71).

To this end, a specific screen class (called `CustomerListScreen`) as well as a *Table* extraction rule (called `ExtractCustomers`) will be created.

The following describes the `CustomerListScreen` screen class:

Table 2 - 6: `LoginScreen` screen class

Name	Description	Detection criterion
`CustomerListScreen`	Corresponds to the legacy application screen displaying a customer list.	Of **Find string** type. Text string is "`TIV010`".

This screen class is a child to the `DefaultScreenClass` screen class (see *"Screen Class"* on page 1-5). As such, it inherits its criterion (the *emulator technology*), default extraction rules and style sheet templates.

The following figure shows which element of the legacy application screen has been chosen in this sample project as detection criterion for the `CustomerListScreen` screen class:

Figure 2 - 73: "TIV010" text string (detection criterion of CustomerListScreen screen class)

To create a new `CustomerListScreen` screen class

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

   A legacy application screen is displayed in the **Emulator**.

2. Browse through legacy screens until reaching the required screen (see Figure 2 - 71).

3. Using the mouse, define a screen zone including the "`TIV010`" text string.

   The screen zone is outlined in white:

Figure 2 - 74: Selecting a detection criterion on screen

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

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

   A new screen class is automatically generated in the **Projects View** tree structure. It is called after the text string selected as detection citerion: `TIV010`.

   As mentioned before, screen class names can be changed to reflect more clearly the screen class that they represent. For example, in our project, we can decide to rename the `TIV010` screen class, which identifies a screen listing customers, into `CustomerListScreen`.

5. Rename the screen class to `CustomerListScreen` (for more information on how to rename an object, see *"To rename a screen class"* on page 2-35).

6. Press `Enter`.

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

   The newly created screen class is now renamed.

   It has inherited its parent screen class (`DefaultScreenClass`) criterion, extraction rules and sheets.

   These inherited elements appear greyed in the **Criteria**, **Extraction rules** and **Sheets**

folder of the screen class:

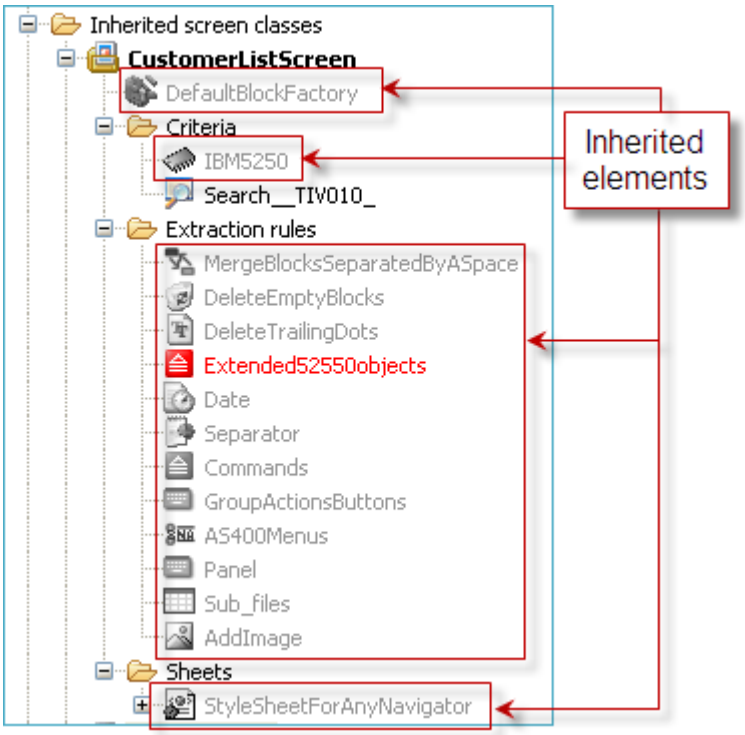


Figure 2 - 75: Inherited elements

The `CustomerListScreen` screen class is now created.

We will now check that, when accessed, this screen is detected as belonging to the defined screen class, `CustomerListScreen`.

8 In the **Emulator** toolbar, click on the **Go to current screen class object** icon .

The `CustomecrListScreen` screen class object is highlighted in the **Projects View**.

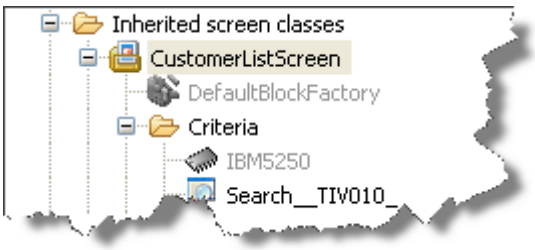


Figure 2 - 76: Highlighted screen class after Go to current screen class object command

WHAT COMES NEXT?

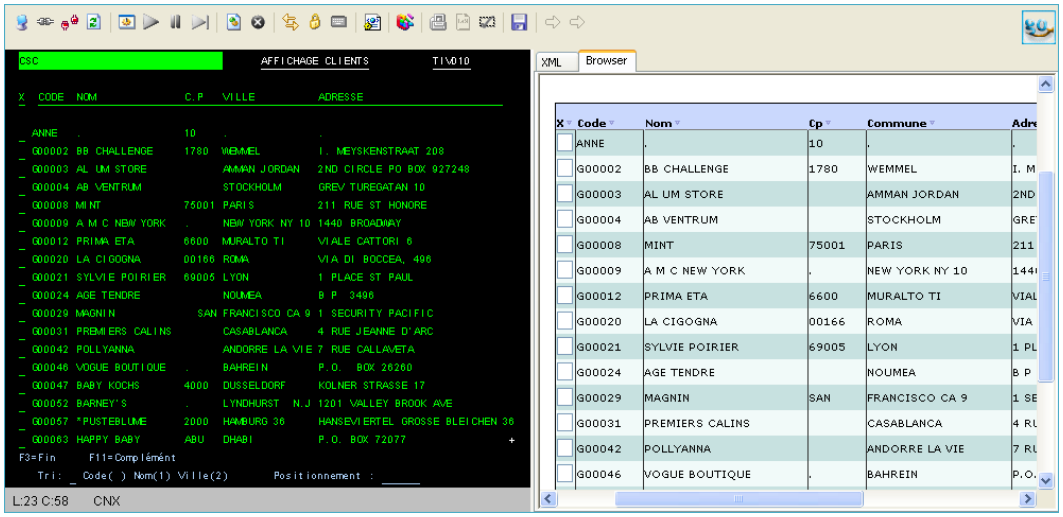
When a screen is detected as belonging to a given screen class, associated extraction rules (either default or user-defined) are executed.

In our case, we want a *Table* extraction rule to be executed in addition to default extraction rules on detection of the `CustomerListScreen` screen class.

The next step therefore consists in creating and setting the `ExtractCustomers` extraction rule, which is described in the table below:

Table 2 - 7: `ExtractCustomers` extraction rule

Extraction rule name	Description
`ExtractCustomers`	Extraction rule type: *Table*. Extracts customer list into a table (see example below).



The procedure for creating and setting the *Table* extraction rule is described below.

To create a Table extraction rule

1. Right-click on the required screen class (or **Extraction rules** screen class folder): for example `CustomerListScreen`.
2. Select ***New > Extraction rule***

   A **New Extraction Rule** wizard opens.
3. Click on **Table**, which is the extraction rule for creating a table:

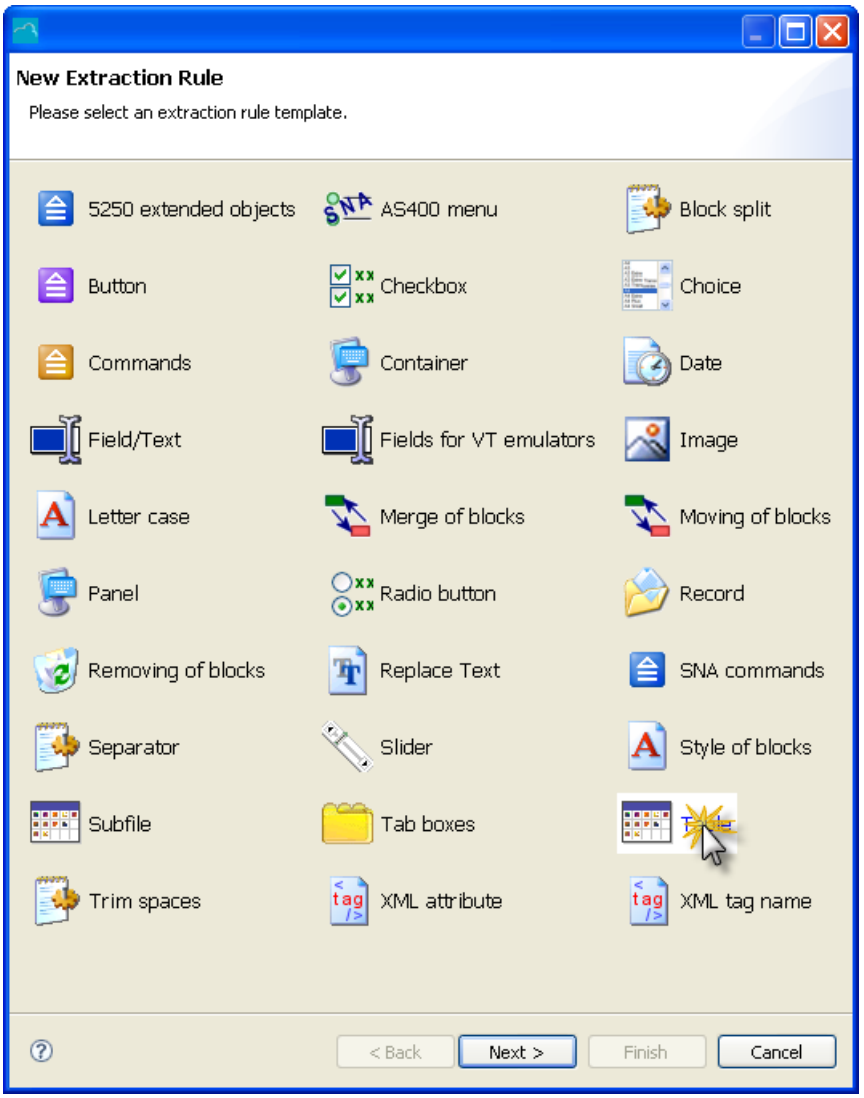


Figure 2 - 77: New Extraction Rule wizard - Table

4 Click **Next**.

5 Type in the name of the new extraction rule (for example `ExtractCustomers`):

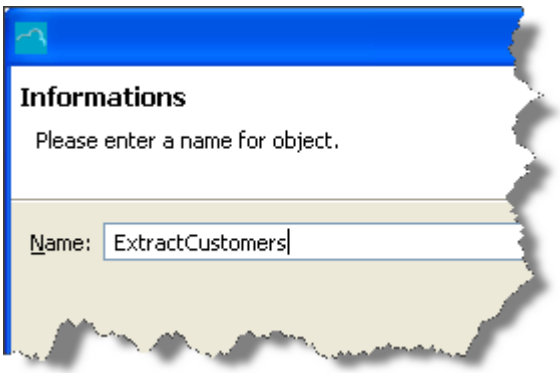


Figure 2 - 78: New extraction rule name - ExtractCustomers

6 Click on **Finish**.

The new extraction rule appears in the **Extraction rules** folder of the `CustomerListScreen` screen class, at the bottom of the extraction rules list:

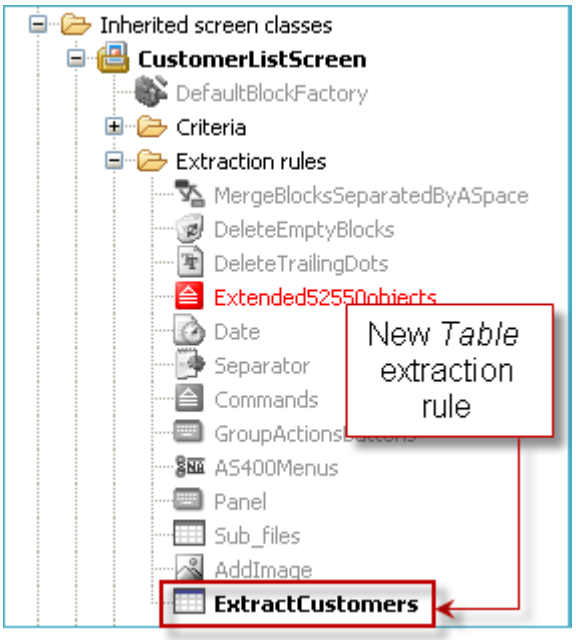


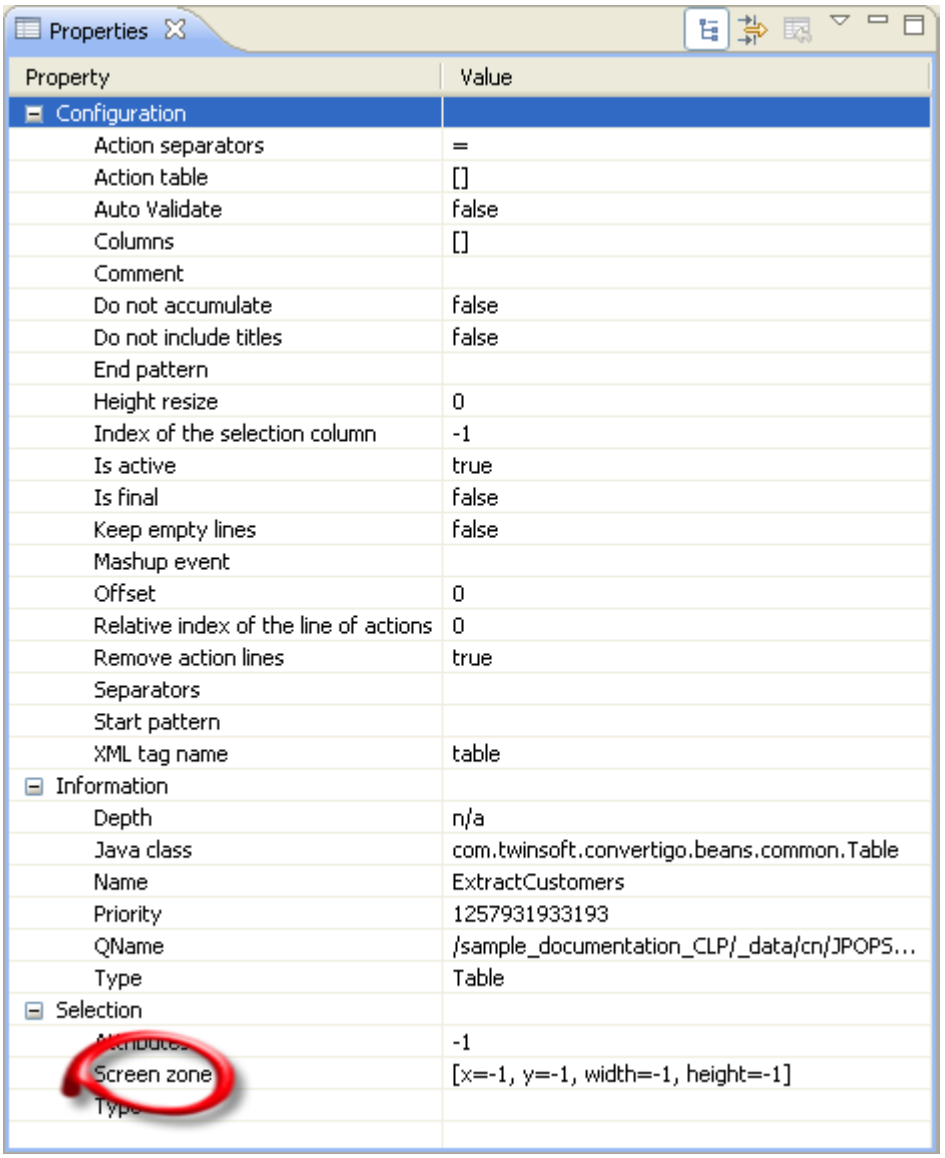
Figure 2 - 79: New extraction rule (ExtractCustomers) in screen class Extraction rules folder

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

The `ExtractCustomers` extraction rule object is now created. We will now set the extraction rule **Screen zone** parameter defining the screen zone to which the rule applies.

8 In the **Projects View**, select the `ExtractCustomers` extraction rule.

The **Properties View** is automatically updated:



Property	Value
Configuration	
Action separators	=
Action table	[]
Auto Validate	false
Columns	[]
Comment	
Do not accumulate	false
Do not include titles	false
End pattern	
Height resize	0
Index of the selection column	-1
Is active	true
Is final	false
Keep empty lines	false
Mashup event	
Offset	0
Relative index of the line of actions	0
Remove action lines	true
Separators	
Start pattern	
XML tag name	table
Information	
Depth	n/a
Java class	com.twinsoft.convertigo.beans.common.Table
Name	ExtractCustomers
Priority	1257931933193
QName	/sample_documentation_CLP/_data/cn/JPOPS...
Type	Table
Selection	
[illegible]	-1
Screen zone	[x=-1, y=-1, width=-1, height=-1]
Type	

Figure 2 - 80: ExtractCustomers extraction rule properties

9 Click on the default value of the **Screen zone** property.

The property name is highlighted and a button appears in the **Value** column:

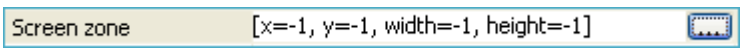


Figure 2 - 81: Screen zone property

To set the **Screen zone** property, you can either:

A. click on the button, set **Top**, **Width**, **Right**, **Left** values in the **Screen zone** window which appears and click **OK**:

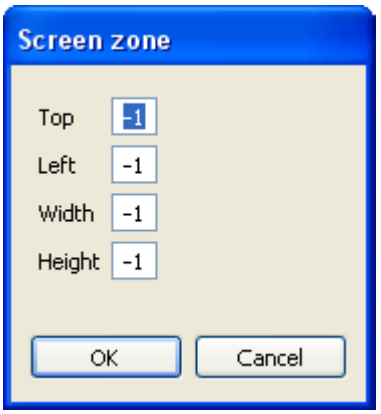


Figure 2 - 82: Screen zone window

B. OR select the required screen zone (that is to say, the whole screen zone containing data to be turned into a table) in the **Emulator** view then click on the **Link current selection to highlighted property in Properties panel** icon :

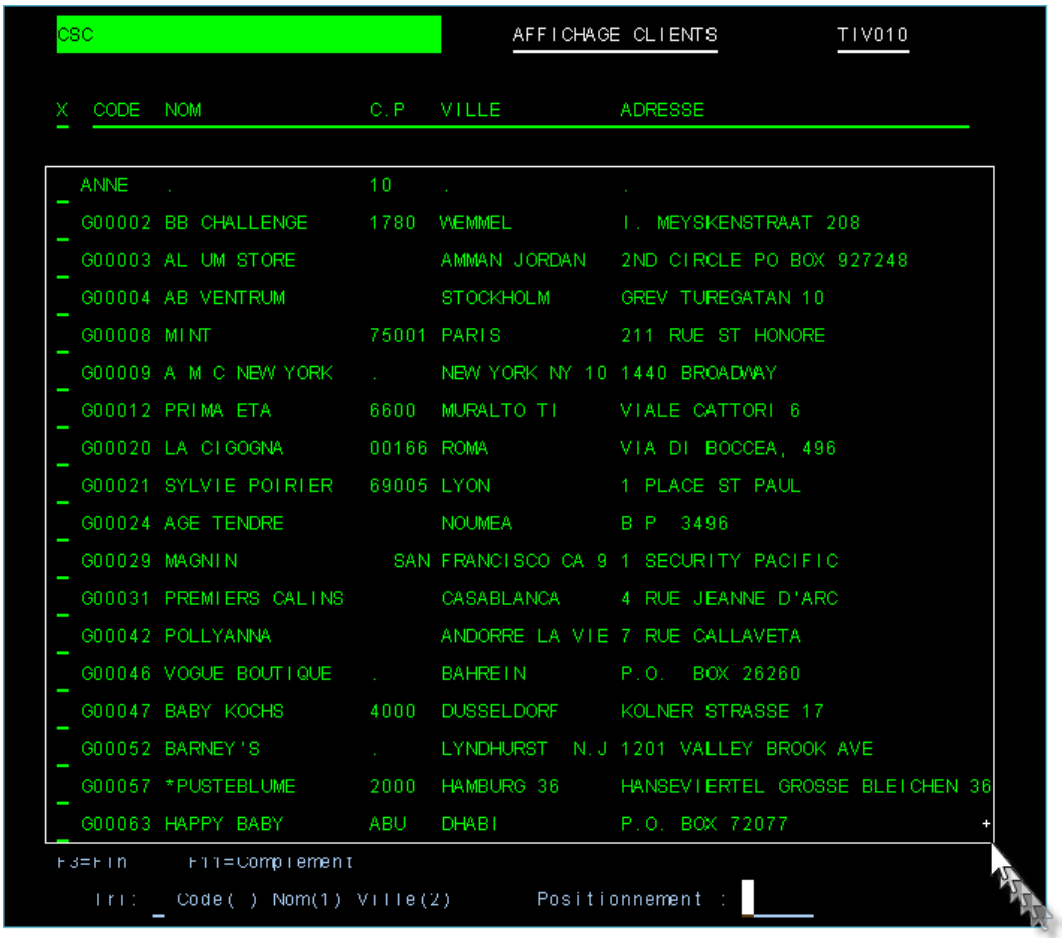


Figure 2 - 83: Selecting the screen zone in Emulator

Figure 2 - 84: Linking selection to Screen zone property

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

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

Screen zone	[x=1, y=4, width=79, height=18]

Figure 2 - 85: Automatic setting of Screen zone property value

The other two main properties of a *Table* extraction rule are **Columns** and **Offset**:

Table 2 - 8: *Table* extraction rule properties

Property	Description
Columns	This parameter describes explicitly the columns of the table. For each column, the Convertigo programmer determines a label (any string), an x start position (left to right position starting by 0) and an ending position. If the property is not defined, the rule table fixes them automatically.
Offset	The table is displayed on the webized screen exactly where the data zone (set through the **Screen Zone** property) is. In most cases, you can use the **Offset** parameter to shift the table up to cover the existing titles. Usually this is to be set to the number of lines of the title zone (1 in this project).

We will now set the table columns. The following figure shows which columns are to be set:

Figure 2 - 86: Table columns to be set

11 In the **Properties View**, select **Columns**.

The property name is highlighted and a button appears in the **Value** column:

Columns	[]

Figure 2 - 87: Columns property

12 To set the **Columns** property, you can either:

A. click on the button, set manually each **column** and click **OK**:

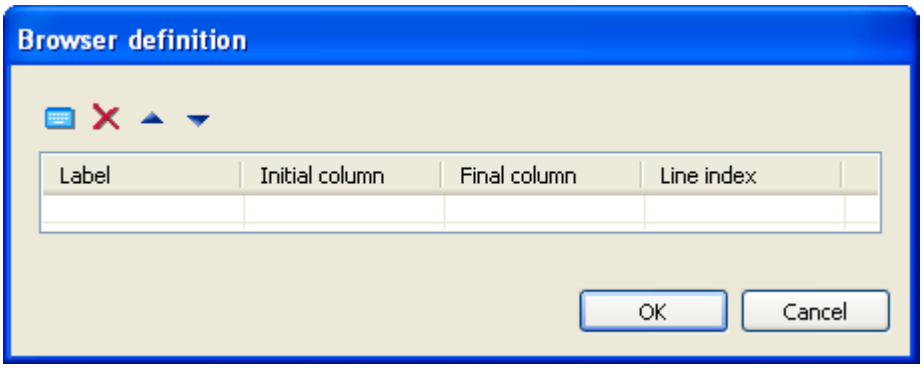


Figure 2 - 88: Columns table window

The table below describes shortly the window columns:

Table 2 - 9: Columns definition table

Column name	Description
Label	Text of the title displayed (Webization). You can use the \ character in a label to specifiy a line break within the column title
Initial column	Starting position of the column (0 based)
Final column	Ending position of the column (0 based)
Line index	For 'folded' tables where a logical line of data is represented physically as several lines. This gives the index of the physical line starting from 0

B. OR, in the **Emulator**, select the required column including the column title (X for the first column) with the mouse (from starting to ending point) then click on the **Add element from current selection to highlighted property in Properties Panel** icon .

It is highly recommended that you use this option, which greatly eases the process of setting columns.

The zone is delineated in white (red dotted lines show the column starting and ending points):

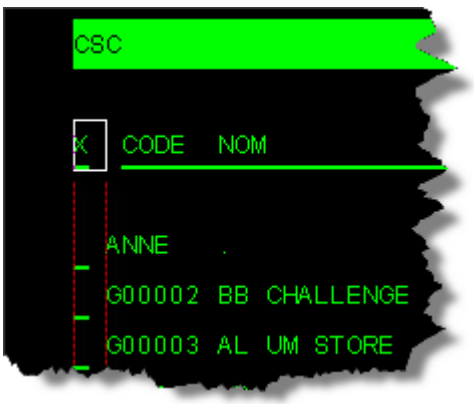


Figure 2 - 89: First column setting

After you have clicked on the icon:

- the whole column is marked in the **Emulator**:

Figure 2 - 90: First column set

- the **Columns** property is updated:

Columns	[[X, 1, 2, 0]]

Figure 2 - 91: Columns property updated

If you select the property in the **Properties View** then click on the button in the **Value** column, the column definition table appears in the window, showing the setting of the first column:

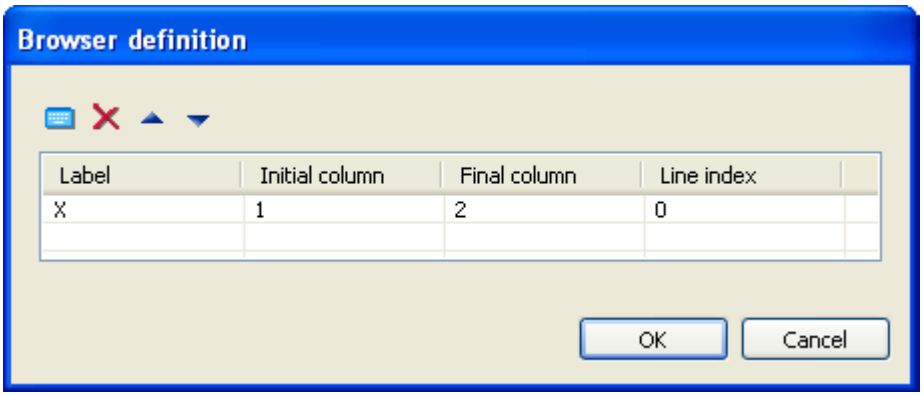


Figure 2 - 92: First column setting in column definition table

We will now set the remaining five columns: `Code`, `Nom` ("Name"), `CP` ("Zip code"), `Ville` ("City") and `Adresse` ("Address").

> *It is highly recommended that every* initial column *is just next to the preceding column* final column *(for example X column* final column *is 2, next column* initial column *should be 3).*

13 Using figure 2 - 86 as a reference, set the remaining five columns by repeating point 12 (option B) of this procedure for each column, one after another.

14 After columns have been set, select the **Columns** property in the **Properties View**.

The property name is highlighted and a button appears in the **Value** column.

15 Click on the button.

The column definition table opens, displaying column settings:

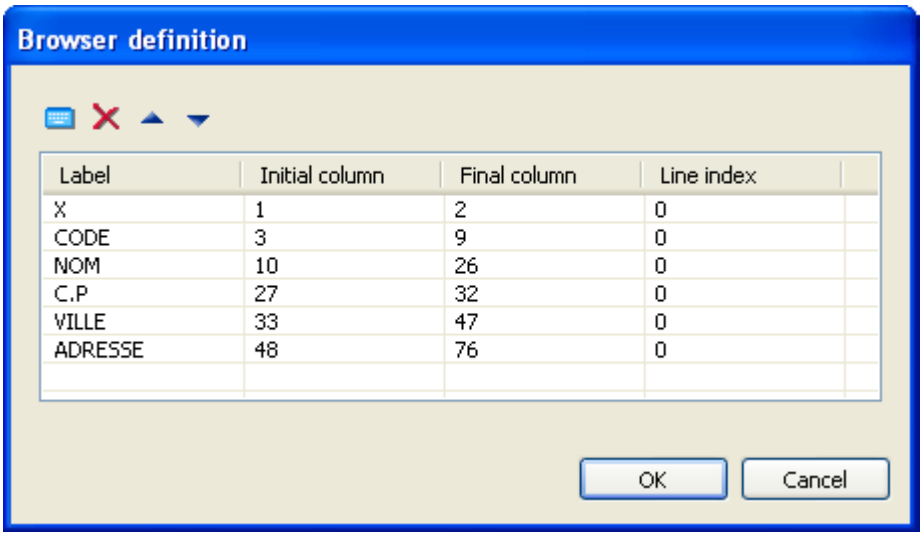


Figure 2 - 93: Settings of table columns

Column names match the names of the columns selected in the **Emulator**.

16 Should you want to change the column names, click the name to change, then enter a new name. For example, change:

- `CODE` to `Code` and press `Enter`,
- `NOM` to `Name` and press `Enter`,
- `C.P` to `Zip code` and press `Enter`,
- `VILLE` to `City` and press `Enter`,
- `ADRESSE` to `Address` and press `Enter`.

17 Click **OK**.

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

The `ExtractCustomers` extraction rule is now properly set. We will check that the specified screen zone is properly webized as a table.

19 Call the webized screen from a Web browser (see *"To call webized legacy application screens from a Web browser"* on page 2-10):

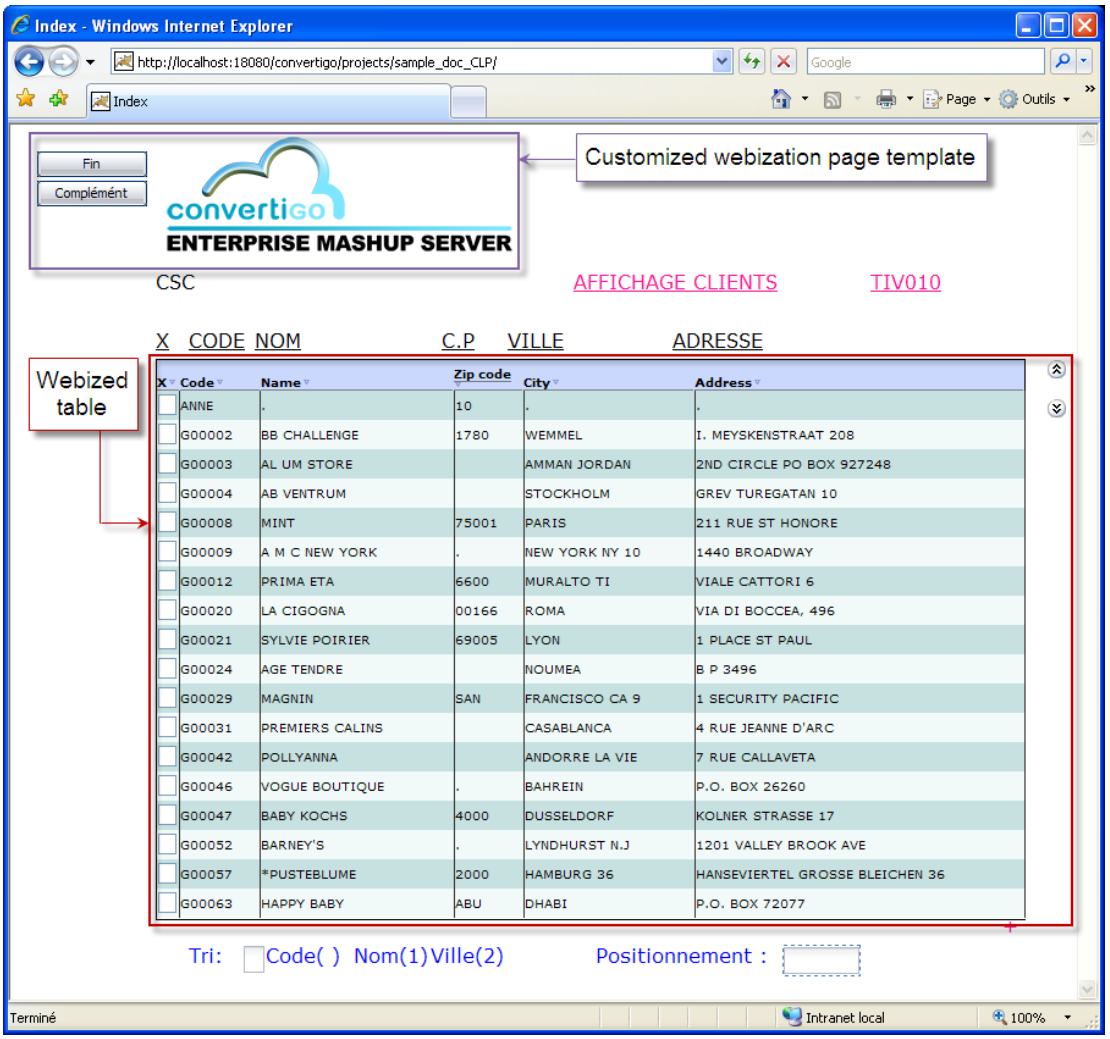


Figure 2 - 94: Webized customer list screen

The legacy screen table listing customers has been properly webized as a table.

You can sort the table columns alphabetically by clicking on the arrow on the left of each column title:

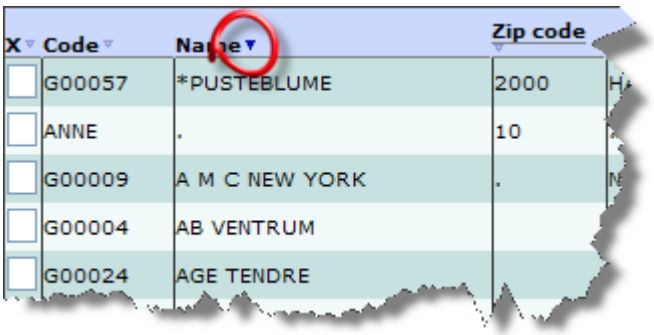


Figure 2 - 95: Sorting columns

Figure 2 - 96 shows that the table title line of the legacy screen has been webized as static text (since it was not selected in the extraction rule screen zone) and appears above the table in the webized screen:

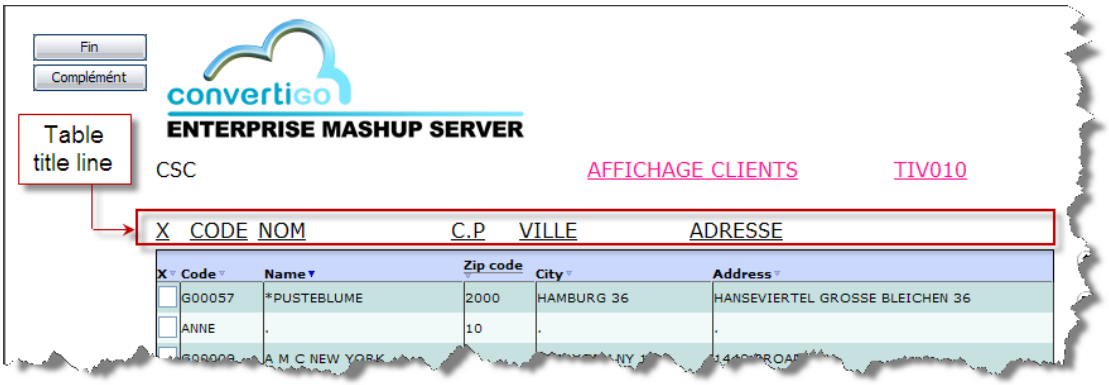


Figure 2 - 96: Table title line webized as static text

Since the table already contains titles, we will now offset the table by one line using the **Offset** property to hide this redundant title line.

20 In the **Properties View**, assuming that the `ExtractCustomers` extraction rule is selected in the **Projects View**, click in the **Value** column of the **Offset** property:

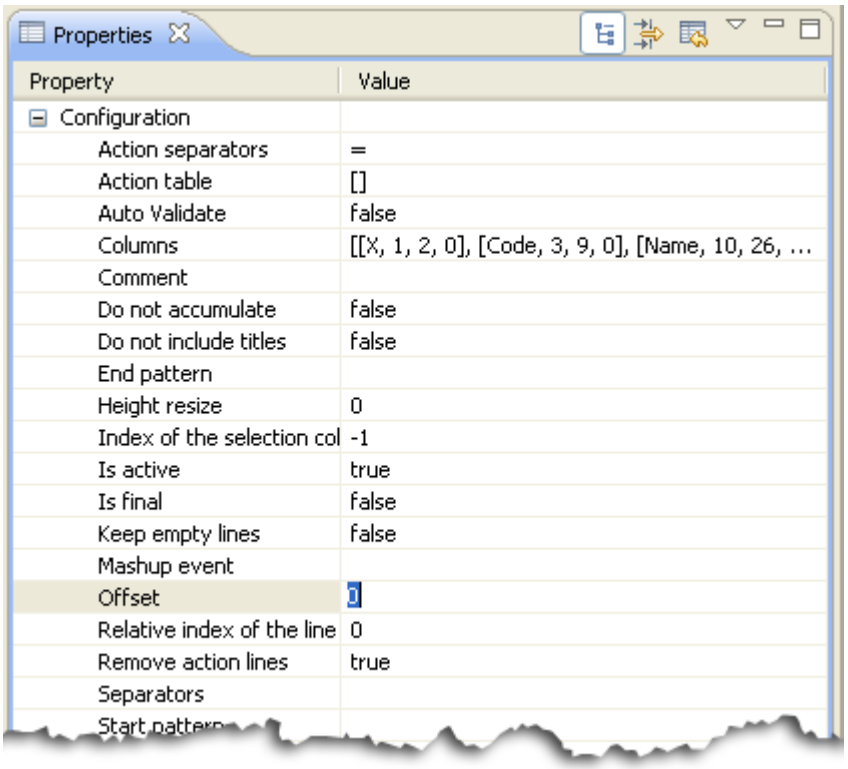


Figure 2 - 97: Table extraction rule Offset property

21 Type in an offset value (for example `1`, which corresponds to the number of lines by which the table is offset).

22 Press `Enter`.

We will now check that the offset value is high enough to hide the webized title line.

23 Call the webized screen from a Web browser (see *"To call webized legacy application screens from a Web browser"* on page 2-10):

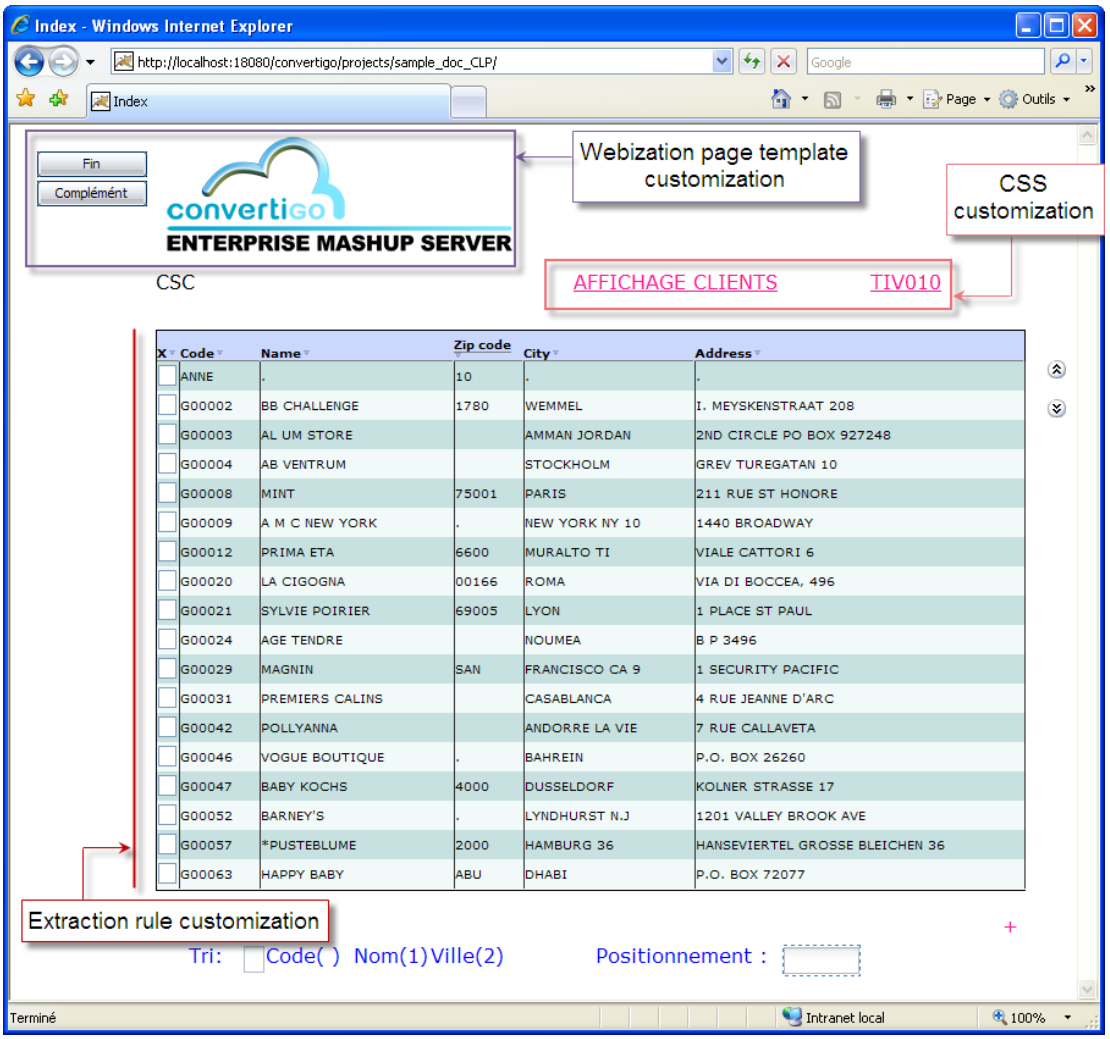


Figure 2 - 98: Webized table with offset value

The screen is now properly webized.

The result of the various customization methods appear in figure 2 - 98:

- in the upper-left corner, the result of the webization page customization;
- in pink and font size 16 pts, the result of the generic webization CSS customization;
- central, the result of the *Table* extraction rule set for the accessed legacy screen.