AURORA

**THE FRONT-END**

AURORA

The front-end

Development documentation.

(C)opyright 2000-2003 cyberSoft.

Version 3.0

[AURORA 1](#_Toc93374858)

[1. INTRODUCTION 6](#_Toc93374859)

[2. Environment 7](#_Toc93374860)

[2.1 Overview 7](#_Toc93374861)

[3. Design Environment 9](#_Toc93374862)

[3.1 Menu Structure 9](#_Toc93374863)

[3.2 Component PropertIes and Event Types 21](#_Toc93374864)

[3.2.1 JCS BEANS 21](#_Toc93374865)

[3.2.1.1 JCSTable 21](#_Toc93374866)

[3.2.1.2 JCSRadioButton 22](#_Toc93374867)

[3.2.1.3 JCSButton 23](#_Toc93374868)

[3.2.1.4 JCSTextPane 24](#_Toc93374869)

[3.2.1.5 JCSList 25](#_Toc93374870)

[3.2.1.6 JCSComboBox 26](#_Toc93374871)

[3.2.1.7 JCSCheckBox 27](#_Toc93374872)

[3.2.1.8 JCSLabel 28](#_Toc93374873)

[3.2.1.9 JCSRegion 28](#_Toc93374874)

[3.2.1.10 JCSBrowser 29](#_Toc93374875)

[3.2.1.11 JCSTree 30](#_Toc93374876)

[3.2.1.12 JCSFilePicker 30](#_Toc93374877)

[3.2.1.13 JCSTextField 31](#_Toc93374878)

[3.2.1.14 Popup Field (JCSHandleButton) 33](#_Toc93374879)

[3.2.1.15 JCSCurrencyField 35](#_Toc93374880)

[3.2.1.16 JCSDateField 36](#_Toc93374881)

[3.2.1.17 JCSMaskField 37](#_Toc93374882)

[3.2.1.18 JCSDateTimeField 39](#_Toc93374883)

[3.2.1.19 JCSPasswordField 40](#_Toc93374884)

[3.2.1.20 JCSPopupMenu 41](#_Toc93374885)

[3.2.1.21 JCSPopupItem 41](#_Toc93374886)

[3.2.2 JCS CONTAINERS 42](#_Toc93374887)

[3.2.2.1 JCSPanel 42](#_Toc93374888)

[3.2.2.2 JCSButtonGroup 43](#_Toc93374889)

[3.2.2.3 JCSTabbedPane 43](#_Toc93374890)

[3.2.2.4 JCSTabpage 44](#_Toc93374891)

[3.2.3 JCS UTILITY BEANS 45](#_Toc93374892)

[3.2.3.1 JCSReport 45](#_Toc93374893)

[3.2.3.2 JCSTextPrinter 45](#_Toc93374894)

[3.2.3.3 JCSTimer 46](#_Toc93374895)

[3.2.3.4 JCSShell 46](#_Toc93374896)

[3.2.3.5 JCSFtp 46](#_Toc93374897)

[3.2.3.6 JCSFileSystem 46](#_Toc93374898)

[3.2.3.7 JCSEncryption 46](#_Toc93374899)

[3.2.3.8 JCSImage 47](#_Toc93374900)

[3.2.3.9 JCSSmartCard 47](#_Toc93374901)

[3.2.3.10 JCSDateUtility 47](#_Toc93374902)

[3.3 Component Design 48](#_Toc93374903)

[3.3.1 JCS BEANS 48](#_Toc93374904)

[3.3.1.1 Table Component 48](#_Toc93374905)

[3.3.1.1.1 Table Operations 52](#_Toc93374906)

[3.3.1.2 Radio Button Component 54](#_Toc93374907)

[3.3.1.4 Text Pane 56](#_Toc93374908)

[3.3.1.5 List 57](#_Toc93374909)

[3.3.1.5.1 ListOperations 58](#_Toc93374910)

[3.3.1.6 Combobox 60](#_Toc93374911)

[3.3.1.6.1 ComboBox Operations 63](#_Toc93374912)

[3.3.1.7 Check Box 64](#_Toc93374913)

[3.3.1.8 Label 65](#_Toc93374914)

[3.3.1.9 Region 66](#_Toc93374915)

[3.3.1.9.1 Designing New Region (Region Template) 66](#_Toc93374916)

[3.3.1.9.2 Using Previously Designed Region 68](#_Toc93374917)

[3.3.1.9.3 Region Operations 70](#_Toc93374918)

[3.3.1.10 JCSBrowser 71](#_Toc93374919)

[3.3.1.10.1 Browser Operations 72](#_Toc93374920)

[3.3.1.11 JCSTree Component 73](#_Toc93374921)

[3.3.1.11.1 Tree Operations 74](#_Toc93374922)

[3.3.1.12 JCSFilePicker 76](#_Toc93374923)

[3.3.1.13 Text Field 79](#_Toc93374924)

[3.3.1.13.1 Text Field Operations 79](#_Toc93374925)

[3.3.1.14 Handle Button Field 80](#_Toc93374926)

[3.3.1.14.1 Designing New Popup (Popup Template) 80](#_Toc93374927)

[3.3.1.14.2 Popup Screen Design 81](#_Toc93374928)

[3.3.1.14.3 Popup Query Design 82](#_Toc93374929)

[3.3.1.14.4 Using Previously Designed Popup 85](#_Toc93374930)

[3.3.1.14.4 Popup Operations 87](#_Toc93374931)

[3.3.1.15 Currency Field 90](#_Toc93374932)

[3.3.1.16 Date Field 91](#_Toc93374933)

[3.3.1.17 Mask Field 92](#_Toc93374934)

[3.3.1.17.1 Using Mask And Pattern Properties 93](#_Toc93374935)

[3.3.1.18 DateTime Field 94](#_Toc93374936)

[3.3.1.19 Password Field 95](#_Toc93374937)

[3.3.1.20 Popup Menu 96](#_Toc93374938)

[3.3.1.21 Popup Menu Item 97](#_Toc93374939)

[3.3.2 JCS CONTAINERS 99](#_Toc93374940)

[3.3.2.1 Panel 99](#_Toc93374941)

[3.3.2.1.1 Panel Operations 99](#_Toc93374942)

[3.3.2.2 Button Group 100](#_Toc93374943)

[3.3.2.2.1 Button Group Operations 101](#_Toc93374944)

[3.3.2.3 Tabbed Pane 102](#_Toc93374945)

[3.3.2.3.1 Tabbed Pane Operations 103](#_Toc93374946)

[3.3.2.4 Tab Page 104](#_Toc93374947)

[3.3.2.4.1 Tab Page Operations 105](#_Toc93374948)

[3.3.3 JCS UTILITY BEANS 107](#_Toc93374949)

[3.3.3.1 JCSReport 107](#_Toc93374950)

[3.3.3.1.1 Report Operations 108](#_Toc93374951)

[3.3.3.2 JCSTextPrinter 108](#_Toc93374952)

[3.3.3.2.1 TextPrinter Operations 109](#_Toc93374953)

[3.3.3.3 JCSTimer 113](#_Toc93374954)

[3.3.3.3.1 Timer Operations 113](#_Toc93374955)

[3.3.3.4 JCSShell 117](#_Toc93374956)

[3.3.3.4.1 Shell Operations 117](#_Toc93374957)

[3.3.3.5 JCSFtp 121](#_Toc93374958)

[3.3.3.5.1 Ftp Operations 122](#_Toc93374959)

[3.3.3.6 JCSFileSystem 125](#_Toc93374960)

[3.3.3.6.1 FileSystem Operations 126](#_Toc93374961)

[3.3.3.7 JCSEncryption 131](#_Toc93374962)

[3.3.3.7.1 JCSEncryption Operations 132](#_Toc93374963)

[3.3.3.8 JCSImage 134](#_Toc93374964)

[3.3.3.8.1 JCSImage Operations 135](#_Toc93374965)

[3.3.3.9 JCSSmartCard 136](#_Toc93374966)

[3.3.3.9.1 JCSSmartCard Operations 136](#_Toc93374967)

[3.3.3.10 JCSClientContext 137](#_Toc93374968)

[3.3.3.10.1 JCSClientContext Operations 138](#_Toc93374969)

[3.4 Rule Design 141](#_Toc93374970)

[3.5 Variable Design 148](#_Toc93374971)

[3.5.1 Variable Usage In Actions 149](#_Toc93374972)

[3.5.1.1 Variable usage as an input/output for a remote call 149](#_Toc93374973)

[3.5.1.2 Variable usage with components 151](#_Toc93374974)

[3.5.2 Variable usage in rule definition 152](#_Toc93374975)

[3.5.3 Variable usage with popups (as input/output) 152](#_Toc93374976)

[3.6 Action Design 152](#_Toc93374977)

[3.6.1 Bean Action 154](#_Toc93374978)

[3.6.2 Remote Call Action 155](#_Toc93374979)

[3.6.3 Reference Call Action 159](#_Toc93374980)

[3.6.4 Variable Setting Action 160](#_Toc93374981)

[3.6.5 Page Call Action 161](#_Toc93374982)

[3.7 Event Design 163](#_Toc93374983)

[3.8 Multilingual Design 166](#_Toc93374984)

[3.9 Utilities 171](#_Toc93374985)

[3.9.1 Tab Order 171](#_Toc93374986)

[3.9.2 Miscellaneous Functions 171](#_Toc93374987)

[4. Reporting Framework 172](#_Toc93374988)

[4.1 Designing Report Visually in EDS 172](#_Toc93374989)

[4.2 Designing Report Template Using VSReport 174](#_Toc93374990)

[4.3 Sample Report Design 175](#_Toc93374991)

[4.4 Sample Special Report (Statement of Account) Design 184](#_Toc93374992)

[5.Sample Screen Design 188](#_Toc93374993)

[6.Authentication &Authorization 193](#_Toc93374994)

[7.Caching In Ere 199](#_Toc93374995)

[8.Using Help In Ere 201](#_Toc93374996)

[9.Web Services 204](#_Toc93374997)

[9.1 Using Aurora Services As Web Services 205](#_Toc93374998)

[9.1.1 Introduction to WSDL 205](#_Toc93374999)

[9.1.2 WSDL Generation from Dictionary Service Definitions 206](#_Toc93375000)

[9.1.3 Assumptions Made in WSDL for AURORA Services 210](#_Toc93375001)

[9.1.4 Sample Usage 211](#_Toc93375002)

[9.1.5 WSDL Sample for an AURORA Web Service 212](#_Toc93375003)

[9.2 Using Outside Web Services 214](#_Toc93375004)

[9.2.1 Calling Web Servıce 214](#_Toc93375005)

[9.2.2 WSDL And Bag Key Name Mapping 214](#_Toc93375006)

[9.2.3 Data Type Mapping Between SOAP And Bag 215](#_Toc93375007)

[9.2.4 Resources 216](#_Toc93375008)

[9.2.4.1 Limitations 216](#_Toc93375009)

[9.2.5 Sample Usages 217](#_Toc93375010)

[9.2.5.1 Search “AURORA” in Google 217](#_Toc93375011)

[10. Preferences 220](#_Toc93375012)

[10.1 User Preferences 220](#_Toc93375013)

[10.1.1 Look & Feel 221](#_Toc93375014)

[10.1.1.1 Primary Colors 225](#_Toc93375015)

[10.1.1.2 Secondary Colors 225](#_Toc93375016)

[10.1.1.3 Black and White 226](#_Toc93375017)

[10.1.1.4 Redefinition of Colors 227](#_Toc93375018)

[10.1.1.5 Fonts 228](#_Toc93375019)

[10.1.2 Session 228](#_Toc93375020)

[10.2 Project Preferences 230](#_Toc93375021)

[10.3 Global Preferences 232](#_Toc93375022)

[10.3.1 Why Need Global Preferences ? 233](#_Toc93375023)

[11. ADDING NEW BEAN 235](#_Toc93375024)

[11.1 AURORA BEAN INTERFACES 235](#_Toc93375025)

[11.2 WRITING BEAN CLASS 236](#_Toc93375026)

[11.3 WRITING BEANINFO CLASS 237](#_Toc93375027)

[12. JASPERREPORTS 1](#_Toc93375028)

[12.1 JASPER REPORT XML (jrxml) 1](#_Toc93375029)

[12.2 DATASOURCE XML (dsxml) 2](#_Toc93375030)

[12.2.1 Sql Type Dsxml 2](#_Toc93375031)

[12.2.2 Service Type Dsxml 3](#_Toc93375032)

[12.2.3 Report Parameters 3](#_Toc93375033)

[12.2 Sub Report 7](#_Toc93375034)

[12.3 Using Text Report Viewer 12](#_Toc93375035)

# 1. INTRODUCTION

AURORA Front-end is composed of a development studio that is called as EBML Development Studio (EDS). EDS is an AURORA specific Graphical User Interface design tool. EDS contains the following two applications:

* EBML Rendering Engine (ERE)
* Content Management Server (CMS) Client Side

CMS is the server-side service test platform of the application and based on java servlet technology.

ERE is the generic EBML rendering engine that will reside on the client machines of the AURORA Application. EDS uses it in order to preview the screens and to test the screens that use services and reference data remote calls. Briefly ERE is embedded into EDS, to test the screens. It is based on Java’s JFC and XML technologies.

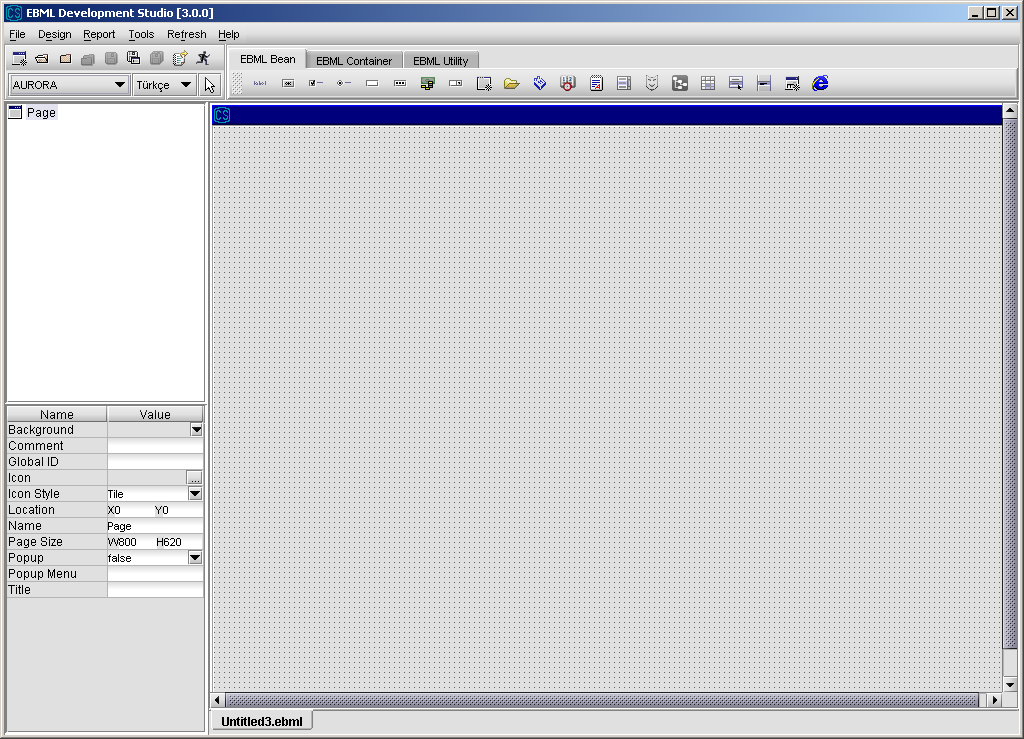
# 2. Environment

## 2.1 Overview

EBML Development Studio consists of seven main menu items as : File, Design, Report, Deployment, Tools, Refresh and Help. Details of each menu item is explained below.

On the upper left corner there is a section that shows the selected page layout and its components.

The properties of the selected page layout or component can be viewed, just below the section under this section. For some of the components, there is a Customizer for setting some of the detailed properties of that component.



On top of the page, there is a toolbar having the following buttons on it.

**Shortcuts to menu items :** New Page, Open File, Close File, Close All Files, Save File, Save File As, Save All Files, Deployment, Test with CMS.

**JCS Beans :** JCSTable, JCSRadioButton, JCSButton, JCSTextPane, JCSList, JCSComboBox, JCSCheckBox, JCSLabel, JCSRegion, JCSBrowser, JCSTree, JCSFilePicker, JCSTextField, JCSPopupFiedl, JCSCurrencyField, JCSDateField, JCSMaskField, JCSDateTimeField, JCSPasswordField, JCSPopupMenu, JCSPopupItem.

**JCS Containers :** JCSPanel, JCSButtonGroup, JCSTabbedPane, JCSTabPage.

**JCS Utility Beans :** JCSReport, JCSTextPrinter, JCSTimer, JCSShell, JCSFtp, JCSFileSystem, JCSDocTool, JCSEncryption, JCSImage, JCSSmartCard.

# 3. Design Environment

## 3.1 Menu Structure

**EBML Development Studio - File Menu**

File Menu provides new page design, region design, popup design, file open, file close and file save operations to the developer. While preferences can be reached from File Menu, recently used files can be seen in Files Menu.

**EBML Development Studio - File Menu - New**

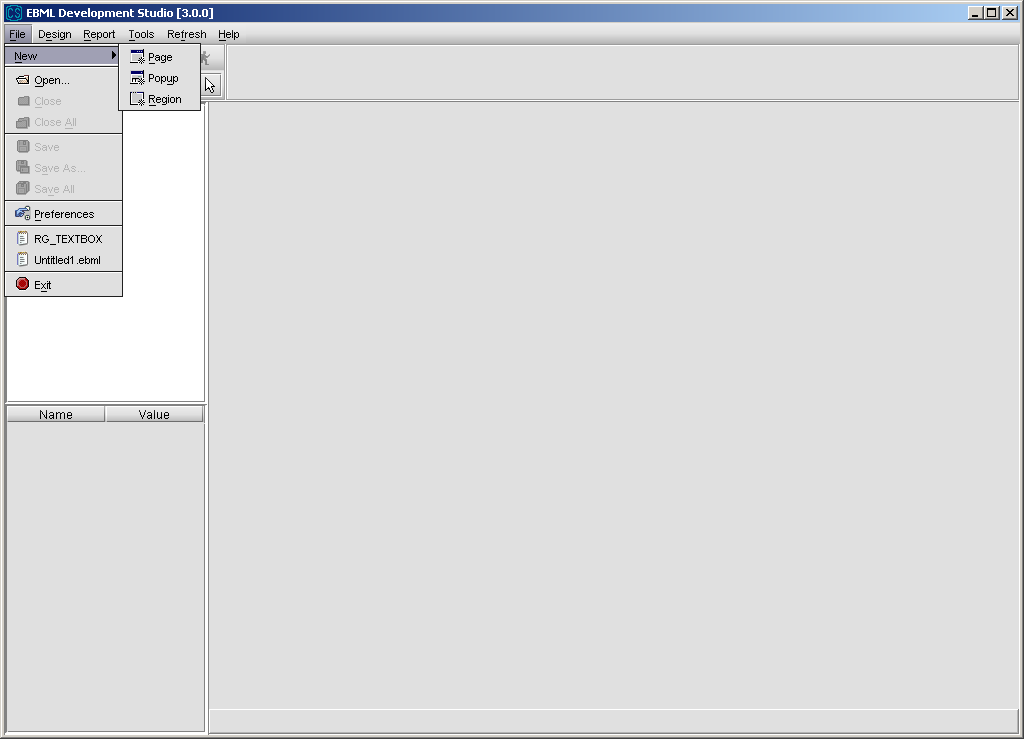
**NewPage :** used for designing a page.

**New Region :** used for designing a new region. Region names must start with

“RG\_”.

**New Popup :** used for designing a new popup. Popup names must start with

“PP\_”.



**EBML Development Studio - File Menu – Open**

File Menu – Open is used for opening files. It will be better to set an initial EbmlFiles directory and save the generated EBML files in that directory.

Using the configuration file that is located in **C:\Documents and Settings\userName\EDSConfig.xml**, developer can set the initial EbmlFiles directory and also set the number of backups for the EBML files. Default values for directory and backup number is initialDir = “c:\AURORA\EbmlFiles” and numberBackup=”3”.

**Example** of an Configuration.xml :

<configuration>

<debug mode="true"/>

<directory local="C:\AURORA\EbmlFiles"/>

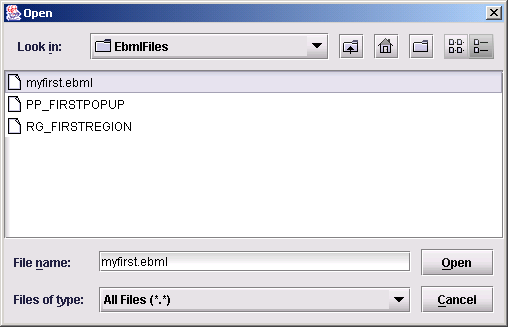
<backup numberBackup="3"/>

<window height="620 " width="800 buttonSet="false "/>

</configuration>

After this setting, for all the generated EBML files 3 backup is generated under

**C:\Documents and Settings\userName\EDS** directory.

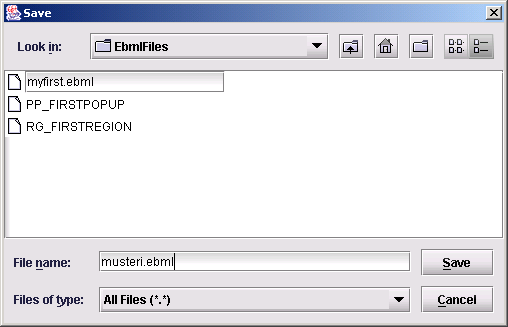
****

**EBML Development Studio - File Menu – Close, Close All**

In this version of EDS, more than one file can be open at the same time. “Close” option, closes the active file and “Close All” option closes all open files.

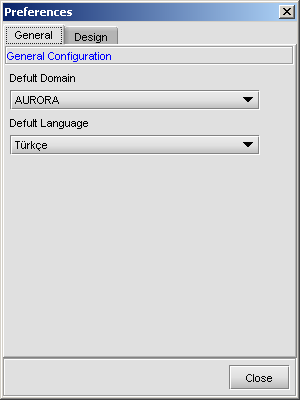
**EBML Development Studio - File Menu – Save, Save As, Save All**

File Menu – Save and Save As is used to save active file under the specified directory. Save All is used to save all open files.

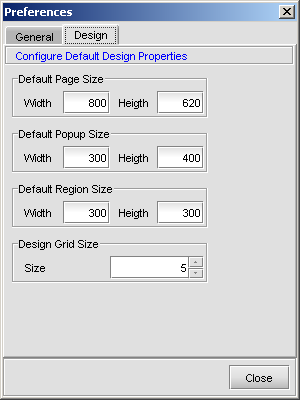
****

**EBML Development Studio - File Menu – Exit**

File Menu – Preferences is used to view and change preferences. Preferences dialog has two tab pages. In the first one, General Tab, working domain and default language can be changed.

****

In the second tab, Design, design grid size and default sizes of pages, popups, regions can be adjusted.

****

**EBML Development Studio - File Menu - Exit**

File Menu – Exit is used to exit from the EBML Development Studio environment.

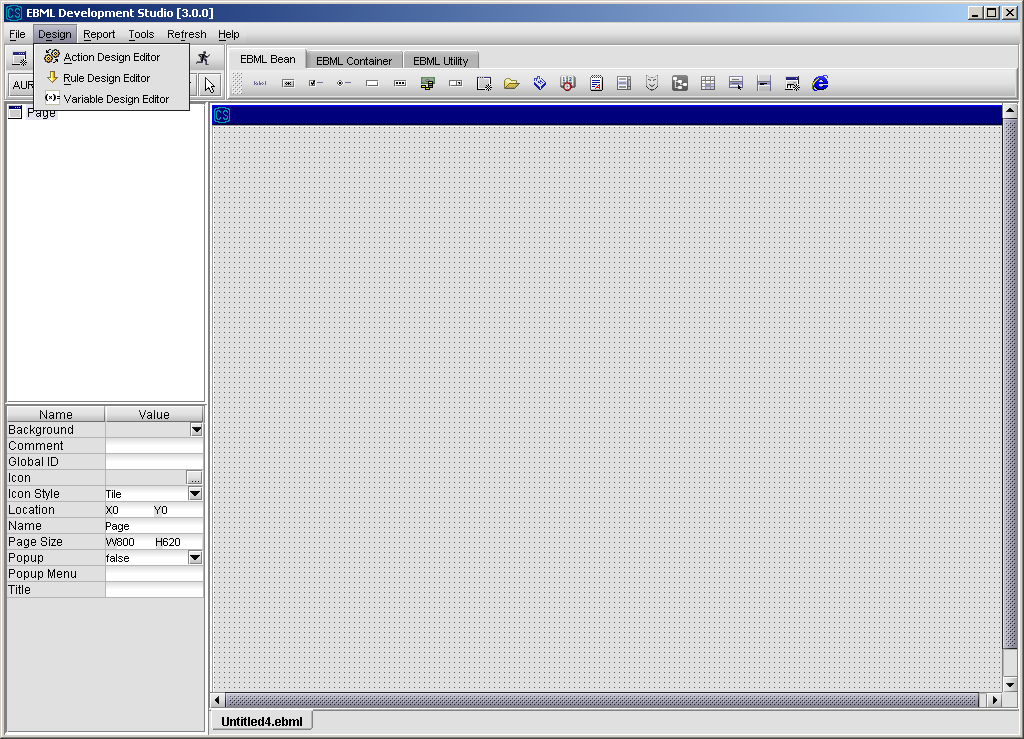
**EBML Development Studio - Design Menu**

Design Menu provides action design, rule design and variable design editors to the developer.

**Action Design Editor :** Used for designing an action. User can create a bean action, a remote call, a reference call, a variable setting, or a page call.

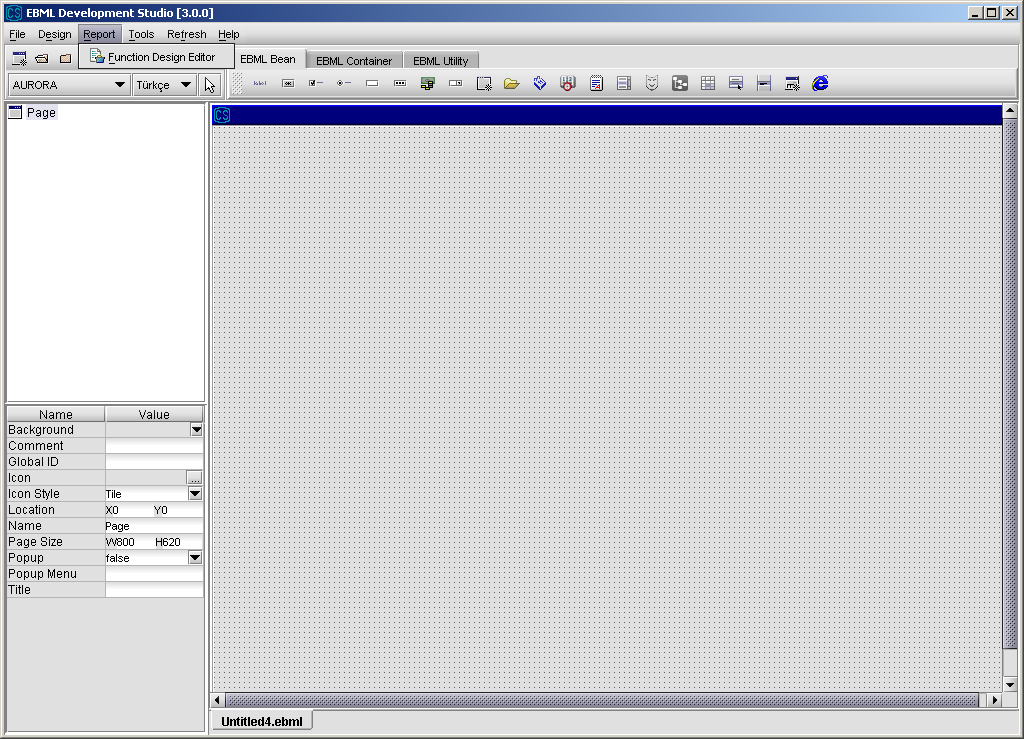
**Variable Design Editor :** Used for defining a new variable. Variables are used for storing values in the page.

**Rule Design Editor :** used for defining a new rule. Rule is used to define different actions according to different controls on the page. For example, if the value chosen from combobox is *firstvalue*, user is directed to page A and if value is *secondvalue* user is directed to page B.



**EBML Development Studio – Report Menu**

**Function Design Wizard : U**sed for designing a new function. Functions are used for designing reports. Details of reporting is explained in **Reporting Framework** section.

****

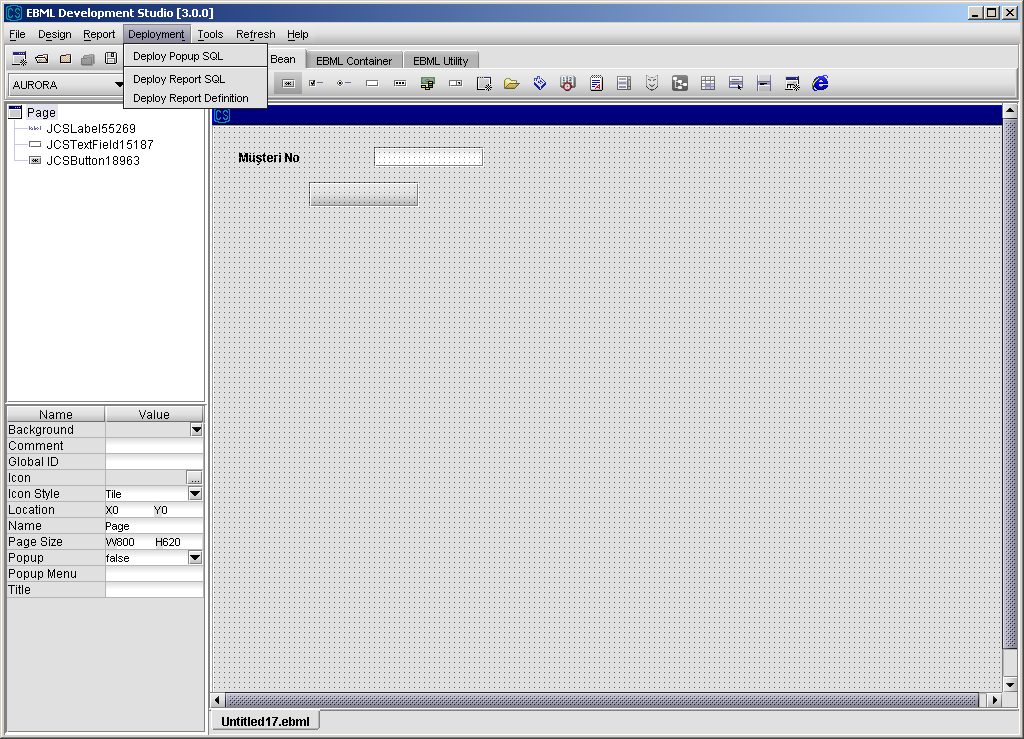
**EBML Development Studio – Deployment Menu**

Deployment Menu provides menu items for deploying report definition files, report SQLs and popup SQLs.

**Deploy Popup SQL :** Used for deploying an SQL file that belongs to a popup file.

**Deploy Report SQL :** Used for deploying an SQL file that belongs to a report.

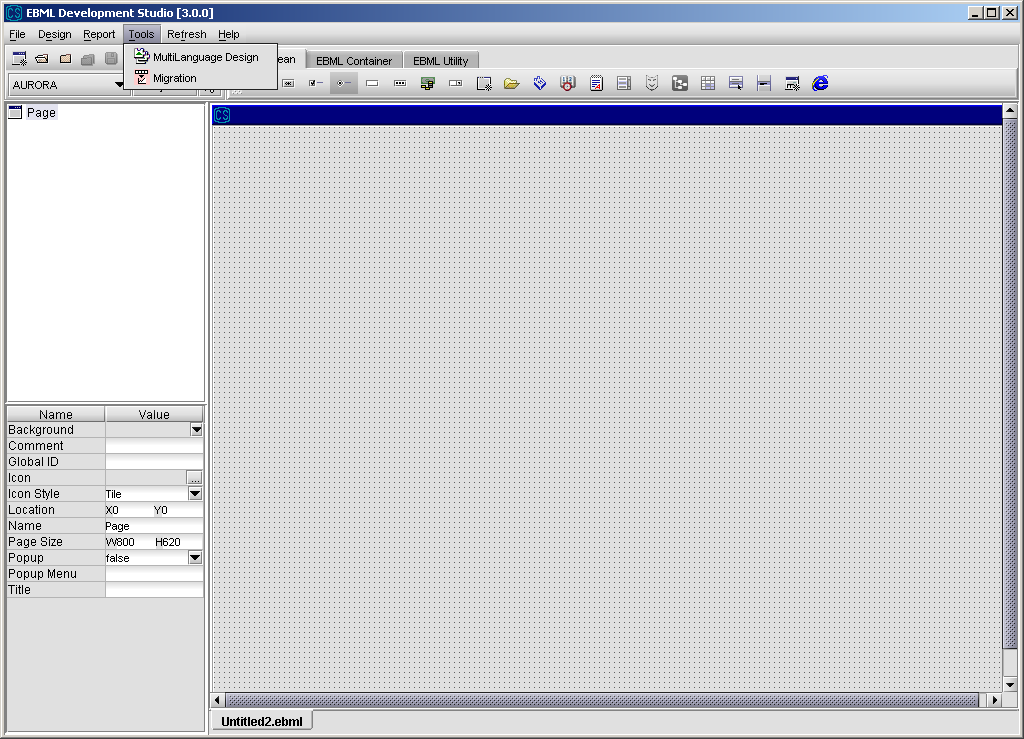
**Deploy Report Definition :** Used for deploying the XML file that defines what is included and how they are shown in a report file

****

**EBML Development Studio – Tools Menu**

**Multilanguage Design :** If page will be used in a multilingual environment, its different properties for different languages is set using this menu item.

**Migration :** Pages from previous EDS version can be migrated to new version using this menu item.

****

**EBML Development Studio - Refresh Menu**

Refresh operations are used to refresh popup list, region list, service list, language list and domain list. Whenever there is a change in these types of data, refresh operation should be performed for the changes to become active.

**Refresh Popup List :** used for refreshing popup list from CMS.

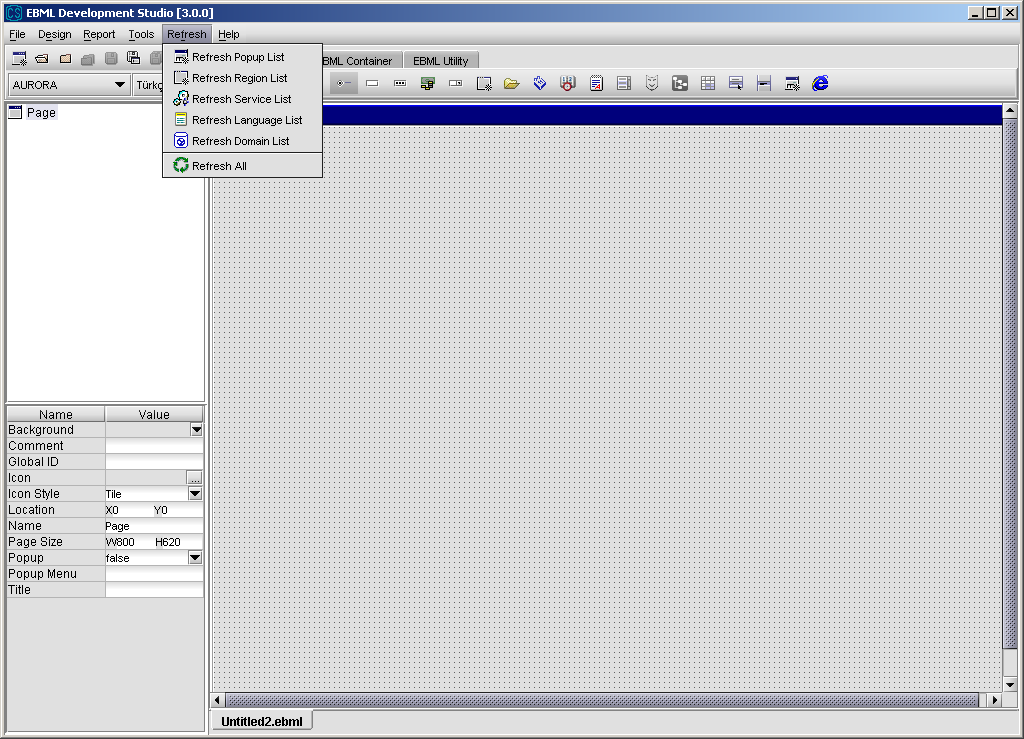
**Refresh Region List :** used for refreshing region list from CMS.

**Refresh Service List :** used for refreshing service list from CMS

**Refresh Language List :** used for refreshing language list from CMS.

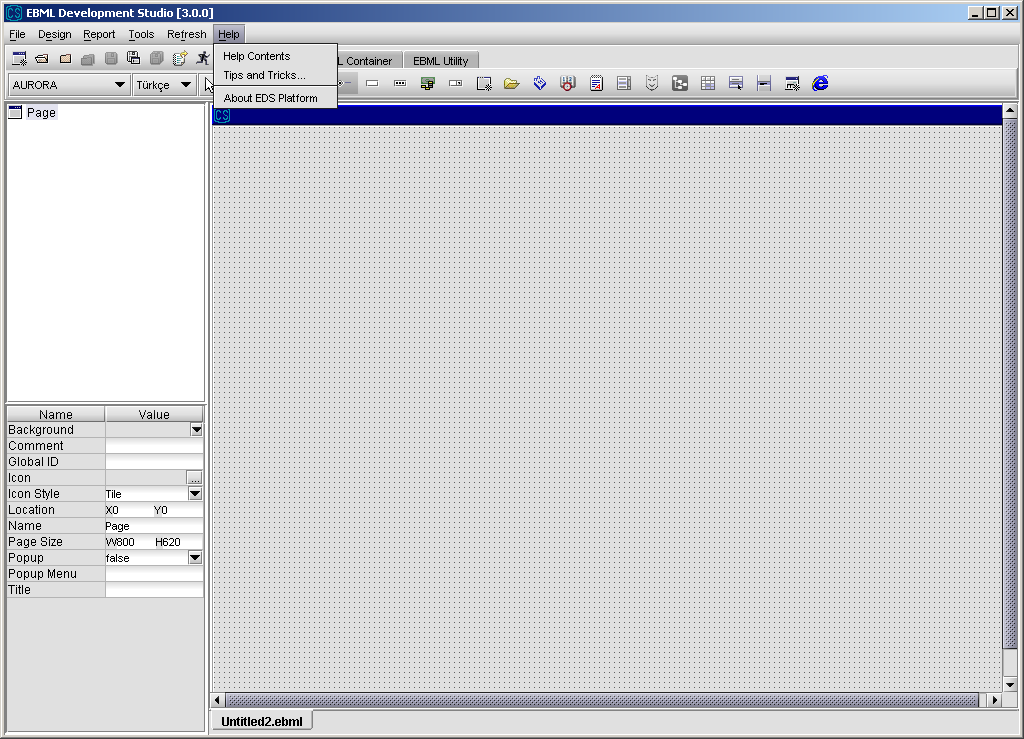
**Refresh Domain List :** used for refreshing domain list from CMS.

**Refresh All :** used to refresh all of the items above.



**EBML Development Studio - Help Menu**

Help is used for giving the developer all the necessary information for using EBML Development Studio – Design Mode.



## 3.2 Component PropertIes and Event Types

### 3.2.1 JCS BEANS

If a component is invisible and its disabled properties is set to “True”, component’s “MustEnter” and “MustFill” properties will be ignored.

#### 3.2.1.1 JCSTable

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Appendable | When a service is called, table is cleared as default. If this property is set true, table is not cleared and incoming data is appended end of table. | default value is “false” |
| Background | background color of the table | default value is “255,255,255” |
| CellSelectionEnabled | true/false | default value is “false” |
| ColumnSelectionAllowed | true/false | default value is “false” |
| Comment | user defined information | none |
| Deletable | true/false : deletable rows. | default value is “true” |
| Editable | true/false : editable rows. | default value is “true” |
| Extendable | true/false : sets the extendibility of the table | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Global ID | For future use | None |
| Height | height of a table | none |
| Insertable | true/false | default value is “true” |
| Location | x,y coordinate of the table | none |
| MinRowCount | Minimum row count of table | default value is “50” |
| MustEnter | true/false | default value is “false” |
| Name | name of table | none |
| Popup Menu | The page name of the popup menu, e.g. Page.myPopupMenu, which will be shown when a mouse right click is done on the table | none |
| Row Header Visible | true : row header is visible  false : row header is invisible | default value is “true” |
| Row Height | height of rows | default value is “20” |
| RowSelectionAllowed | true/false | default value is “true” |
| Send ColumnWise | true : sends column  identifier as a BagKey  false: sends column index. | default value is “false” |
| Show Grid | true/false | default value is “true” |
| Sortable | Refers to if column can be sorted when user double-click column header. |  |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Title Line Count | Refers to how many lines table column header has to show column title. | Default value is “1” |
| Visible | true/false : visibility | default value is “true” |
| Width | width of table | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| cellDataChanged | the data of a table cell is changed |
| rowDeleted | row is deleted from the JCSTable |
| rowInserted | row is inserted in the JCSTable |
| rowSelected | row is selected in the Table |
| mouseDoubleClicked | double mouse click on the row header |
| mousePressed | single mouse click on the row header |

#### 3.2.1.2 JCSRadioButton

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property** | **Value – Meaning** | | **Restriction/Default Value** | |
| Background | background color of the radio button | | default value is “224,224,224” | |
| Comment | user defined information | | none | |
| Enabled | true/false | | default value is “true” | |
| Font | font type and size of text | | default value for font type “Dialog”, for font size “12” | |
| Foreground | font color of the text | | default value is “0,0,0” | |
| Global ID | For future use | | None | |
| Height | height of the radio button | | none | |
| Location | x,y coordinate of the radio button | | none | |
| Name | name of the radio button | | none | |
| Read Only | true/false | | default value is “false” | |
| Selected | true/false | | default value is “false” | |
| Tab Order | 0, 1, 2,... | | default value is “0” | |
| Text | Text area | | default value is “JCSRadioButton” | |
| Tool Tip Text | tooltip for the component | | none | |
| Visible | True/false : visibility | | default value is true | |
| Width | width of the radio button | | none | |
| **Event Type** | | **Meaning** | |
| actionPerformed | | JCSRadioButton is pressed | |
| itemStateSelected | | JCSRadioButton is selected | |
| itemStateUnSelected | | JCSRadioButton is unselected | |

#### 3.2.1.3 JCSButton

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the button | default value is “224,224,224” |
| Comment | user defined information | none |
| Enabled | true/false | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of the button | none |
| Horizontal Alignment | alignment of name on the button | default value is “CENTER” |
| Icon | a name of a bitmap file to be displayed on button | bitmap file must reside in local directory under c:\AURORA\Images. Also bitmap file must be deployed to the server. (\\Aurora01\Aurora\CMS\Data\ebml\images) |
| Location | x,y coordinate of the button | none |
| Name | name of the button | none |
| Tab Order | 0, 1, 2,... | default value is 0 |
| Text | text area | default value is “Button” |
| Tool Tip Text | tooltip for the component | none |
| Visible | true/false : visibility | default value is true |
| Width | width of the button | none |

#### 

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | JCSButton is pressed |

#### 3.2.1.4 JCSTextPane

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the text pane | default value is “255,255,255” |
| Comment | user defined information | none |
| Content Type | content type of text pane:  Plain: All characters can be entered  Alpha: Only letters  Numbers: Only numbers  AlphaNumeric: Both letters and numbers, nothing else | default value is “Plain” |
| Editable | true/false | default value is “true” |
| Exceptions | extra keys that can be entered to text field. With this option user can define extra keys –like ‘&’- that can be inserted in the text field. | none |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of the text pane | none |
| LetterCase | letter case of text pane; UpperCase/LowerCase/NoCase | default value is “NoCase” |
| Limit | max number of character in the text area.-1 means no limit. | default vale is “-1” |
| Location | x,y coordinate of the table | none |
| Mime Type | Values can be  text/plain  text/html  text/rtf | default value is text/plain |
| Must Enter | true: must have a value  false: not must | default value is false |
| Name | name of the text pane | none |
| Tab Order | 0, 1, 2,... | default value is 0 |
| Text | text content | default value is “JCSTextPane” |
| Visible | true/false : visibility | default value is “true” |
| Width | width of the text pane | none |

#### 

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| focusGained | when focus comes on JCSTextPane |
| focusLost | another item is selected after selecting JCSTextPane |

#### 3.2.1.5 JCSList

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the list | default value is “255,255,255” |
| Comment | user defined information | none |
| Enabled | true/false  list enabled/disabled in runtime | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of the list | none |
| Location | x,y coordinate of the list | none |
| Name | name of the list | none |
| Reference Data | is for defining reference data  set to a reference data table name like “PRM\_…” | none |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Uniqueness | true/false | default value is “false” |
| Visible | true/false : visibility | defaılt value is “true” |
| Width | width of the List | default value is “150” |

#### 

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| focusLost | another item is selected after selecting JCSList |
| valueChanged | another item is selected on the JCSList |
| mouseDoubleClicked | mouse is double clicked on JCSList |

#### 3.2.1.6 JCSComboBox

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the combo box | default value is “255,255,255” |
| Comment | user defined information | none |
| Enabled | true/false  combo enabled/disabled in runtime | default value is “true” |
| Filtered | True: In this case, combobox have filter-key-value triples, instead of key-value pairs. When the filter is set via setFilter method, only items whose filter property matches the filter are listed.  False: All items are listed. | default value if “false” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of combobox | none |
| Limit | -1 | default value is “-1” |
| Location | x,y coordinate of the combobox | none |
| Must Enter | true: must have a value if it sends to service as a input parameter.  false: not must | default value is “false” |
| Name | name of combobox | none |
| ReadOnly | true/false | default value is “false” |
| Reference Data | is for defining reference data  set to a reference data table name like “PRM\_…” | none |
| Select Only | true : combo is only selectable  false : combo can also be edited. When “enter” is pressed first match is selected | default value is “true” |
| Show Text | true/false  shows both text and value fields for each data | default value is “false” |
| Sorted | true/false | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Tool Tip Text | tooltip for the component | none |
| ValueUpperCase | true/false | default value is “false” |
| Visible | true/false : visibility | default value is “true” |
| Width | width of combobox | default value is “110” |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | like itemStateChanged but not require a different selection |
| focusLost | another item is selected after selecting JCSComboBox |
| itemStateChanged | an item is selected from the combo box |

#### 3.2.1.7 JCSCheckBox

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the checkbox | default value is “224,224,224” |
| Comment | user defined information | none |
| Enabled | true/false  checkbox enabled/disabled in runtime | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of checkbox | none |
| Location | x,y coordinate of the checkbox | none |
| Name | name of checkbox | none |
| ReadOnly | true/false | default value is “false” |
| Selected | true/false | default value is false |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | text of checkbox | default value is “CheckBox” |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of checkbox | none |

#### 

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | JCSCheckBox is pressed |
| ItemStateSelected | JCSCheckBox is selected |
| itemStateUnSelected | JCSCheckBox is unselected |

#### 3.2.1.8 JCSLabel

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Comment | user defined information | none |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “32,32,64” |
| Global ID | For future use | None |
| Height | height of label | none |
| Horizontal Alignment | horizontal alignment of label; left,center,right,leading,trailing | default value is “LEADING” |
| Icon | a name of a bitmap file (jpg, gif) to be displayed on label | bitmap file must reside in local directory under c:\AURORA\Images. Also bitmap file must be deployed to the server.  (\\Aurora01\Aurora\CMS\Data\ebml\images) |
| Location | x,y coordinate of the label | none |
| Name | name of label | default value is “JCSLabel” |
| Text | text of label | default value is “JCSLabel” |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of label | none |

#### 

#### 3.2.1.9 JCSRegion

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Border Visible | true/false | default value is “true” |
| Font | font type and size of text of title | default value for font type “Dialog”, for font size “12” |
| Global ID | For future use | None |
| Height | height of region | none |
| Location | x,y coordinate of the region | none |
| Name | name of region | must start with RG\_ |
| Popup | true/false  popup type region enabled/disabled in runtime | default value is false |
| Page Load Once | When region is used for popup dialog, this value determines whether page load method will be called every time when popup is shown. When it is set to “true” page load is called only once when reion popup is first shown. Otherwise every time when popup is visible, region page load method will be called. | Default value is “false”. This means that region page load method will be called every time when region popup dailog is visible. |
| Region Name | list of already deployed regions | default value is “EMPTY\_REGION” |
| Tab Order | 0, 1, 2,... | default value is 0 |
| Title | title of region | default value is “Region Title” |
| Visible | true/false | default value is “true” |
| Width | width of region | none |

#### 3.2.1.10 JCSBrowser

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Comment | user defined information | None |
| Global ID | For future use | None |
| Height | height of browser | none |
| Location | x,y coordinate of the browser | none |
| Name | name of the browser component | None |
| Tab Order | 0, 1, 2,... | default value is “0” |
| URL | URL address | None |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of browser | none |

#### 3.2.1.11 JCSTree

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the tree | default value is “255,255,255” |
| Comment | user defined information | none |
| Enabled | true / false | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Global ID | For future use | none |
| Height | height of tree | none |
| Location | x,y coordinate of the tree | none |
| Name | name of the tree component | none |
| Popup Menu Name | The page name of the popup menu, e.g. Page.myPopupMenu, which will be shown when a mouse right click is done on the table | none |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Tree Type | default: Normal tree  checkbox: A checkbox appears at the left of the every tree node | default value is “DEFAULT” |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of tree | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| mouseClicked | mouse is clicked on JCSTree |
| mouseDoubleClicked | mouse is double-clicked on JCSTree |
| valueChanged | value of a node is chaned in JCSTree |

#### 3.2.1.12 JCSFilePicker

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the filepicker | default value is “255,255,255” |
| Binary | read the file binary or not | default value is “false” |
| Comment | user defined information | none |
| Selection Mode | values; Files Only, Directories Only, Files and Directories | default value is “Files Only” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of the filepicker | none |
| Initial Directory | directory of the file | none |
| Location | x,y coordinate of the file picker | none |
| Multiselection Enable | true : more than one file can selected  false : only one file can be selected | default value is “false” |
| Name | name of the component | none |
| SelectedFile | name of the file selected | none |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of the file picker | default value is “150” |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| fileSelected | file is selected and the button “Choose” is pressed |

#### 

#### 3.2.1.13 JCSTextField

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| AutoSkip | false/true  autoskip disabled/enabled | default value is “false” |
| Background | background color of the text field | default value is “255,255,255” |
| Comment | user defined information | none |
| Content Type | content type of text field:  Plain: All characters can be entered  Alpha: Only letters  Numbers: Only numbers  AlphaNumeric: Both letters and numbers, nothing else | default value is “Plain” |
| Editable | true/false | default value is “true” |
| Exceptions | extra keys that can be entered to text field. With this option user can define extra keys –like ‘&’- that can be inserted in the text field. | none |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | none |
| Height | height of text field | none |
| Horizontal Alignment | horizontal alignment of label; left,center,right,leading,trailing | default value is “LEADING” |
| LetterCase | letter case of text field; UpperCase/LowerCase/NoCase | default value is “NoCase” |
| Limit | number of characters that can be entered to text field | default value is “–1” |
| Location | x,y coordinate of the text field | none |
| Min. Char. Count | Refers to minimum input character count to make this bean valid before service call. When this value is set “true”, “Must Enter” property is set to true to indicate something must be entered at least minimum character count. Otherwise service call show an error dialog. | Default value is “0” |
| Must Enter | data must be entered | default value is “false” |
| Must Fill | text field must be filled | default value is “false” |
| Name | name of text field | none |
| ReadOnly | false/true  user can enter data/user can not enter data | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | text of text field | none |
| Tool Tip Text | tooltip for the text field | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of text field | none |
| ZeroPad | filling the left part of text inserted with 0s until “limit” value | defaut value is “false” |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while JCSTextField is selected |
| focusGained | JCSTextField is selected |
| focusLost | another item is selected after selecting JCSTextField |

#### 3.2.1.14 Popup Field (JCSHandleButton)

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| All At Once | list all query result or not | default value is “false” |
| Auto Execute | auto execution when data is set to handlebutton field |  |
| AutoSkip | false/true  autoskip disabled/enabled | default value is false |
| Background | background color of the pop up field | default value is “255,255,255” |
| Bigger | used to increase the size of the popup 50% so that popup window will be larger both at design and run-time. | default value is false |
| Comment | user defined information | none |
| Content Type | content type of popup field:  Plain: All characters can be entered  Alpha: Only letters  Numbers: Only numbers  AlphaNumeric: Both letters and numbers, nothing else | default value is “Plain” |
| Editable | true/false  user can enter data/user can not enter data | default value is “true” \*\*\*\* aynı isimli 2 tane var”” |
| Exceptions | extra keys that can be entered to text field. With this option user can define extra keys –like ‘&’- that can be inserted in the text field. | none |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of popup field | none |
| Horizontal Alignment | horizontal alignment of text on pop-up field | default value is “LEADING” |
| LetterCase | letter case of popup field; UpperCase/LowerCase/NoCase | default value is “NoCase” |
| Limit | number of characters that can be entered to popup field | default value is “–1” |
| Location | x,y coordinate of the popup field | none |
| Must Enter | data must be entered | default value is false |
| Must Fill | popup field must be filled | default value is false |
| Name | name of popup field | must start with “PP\_” |
| Popup | popup field that will appear when the handled button is clicked | default value is “EMPTY\_POPUP” |
| Query Block Size | If All At Once property is false, the size of each data block | default value is “50” |
| Query Immediately | query without user action | default value is “false” |
| ReadOnly | true/false  used to set readonly property of the handle button field. | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Template | used to set masked field template format if masked field will be used on a handle button. | format of the value is “(-----)--/--“ |
| Text | text displayed on the popup in design environment | default value is “JCSHandleButtonField” |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Warning Service | name of the service to be called | none |
| Width | width of popup field | none |
| ZeroPad | filling the left part of text inserted with 0s until “limit” value | default value is true |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while JCSHandleButton is selected |
| focusGained | JCSHandleButton is selected |
| focusLost | another item is selected after selecting JCSHandleButton |
| onShowPopup | popup is fired |

#### 3.2.1.15 JCSCurrencyField

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| AutoSkip | false/true  autoskip disabled/enabled | default value is “false” |
| Background | background color of the currency field | default value is “255,255,255” |
| Allow Negative | false/true  can be nagative/cannot be negative | default value is “false” |
| Comment | user defined information | none |
| Editable | true/false  user can enter data/user can not enter data | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of currency field | none |
| Location | x,y coordinate of the currency field | none |
| MustEnter | data must be entered | default value is “false” |
| MustFill | currency field must be filled | default value is “false” |
| Name | name of currency field | none |
| Pattern | declares total number of digits and precision that an inserted value can have | default value is “18,0” |
| Read Only | false/true  user can enter data/user can not enter data | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | text of currency field | only numeric values can be entered default value is “0” |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of currency field | None |
| clearWith | Set a key stroke that clears currency field content according to its pattern | Key stroke can contain a mask and character key. For instance,  control SPACE  control DELETE  shift UP  F2  control F3  ESCAPE  BACK\_SPACE  Make sure that character keys are capital while mask keys are not. |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while JCSCurrencyField is selected |
| focusGained | JCSCurrencyField is selected |
| focusLost | another item is selected after selecting JCSCurrencyField |

#### 3.2.1.16 JCSDateField

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| AutoSkip | false/true  autoskip disabled/enabled | default value is “false” |
| Background | background color of the date field | default value is “255,255,255” |
| Comment | user defined information | none |
| Editable | true/false  user can enter data/user can not enter data | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of date field | none |
| Horizontal Alignment | horizontal alignment of text on date field | default value is “Leading” |
| Location | x,y coordinate of the date field | none |
| MustEnter | data must be entered | default value is “false” |
| Name | name of date field | none |
| ReadOnly | false/true  user can enter data/user can not enter data | default value is “false” |
| Seperator | separator type for date, values; dot, forward slash | default value is “Dot” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | text of date field | only numeric values can be entered |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of date field | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while JCSDateField is selected |
| focusGained | JCSDateField is selected |
| focusLost | another item is selected after selecting JCSDateField |

#### 3.2.1.17 JCSMaskField

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| AutoSkip | false/true  autoskip disabled/enabled | default value is “false” |
| Background | background color of the mask field | default value is “255,255,255” |
| Comment | User defined information | none |
| Editable | True/false | default value is “True” |
| Enabled | True/false | default value is “true” |
| Exceptions | Exception charachter squence | These values are only acceptable where mask character “E” is. |
| Font | Font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | Font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of mask field | none |
| Horizontal Alignment | horizontal alignment of text on mask field | default value is “LEADING” |
| Location | x,y coordinate of the mask field | None |
| Must Enter | Data must be entered | default value is “false” |
| Must Fill | currency field must be filled | default value is “false” |
| Name | name of mask field | none |
| Pattern | pattern of mask field | Meaning of mask characters:  # : Digit  A : Letter  X : Alpha-numeric character  U : Upper-case character  L : Lower –case character  E : One of “exceptions” character  \* : Any character  “-” means use the value defined in mask field |
| ReadOnly | false/true  user can enter data/user can not enter data | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | Text of mask field | none |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of text field | none |
| ZeroPad | filling the left part of text inserted with 0s until “limit” value | default value is “false” |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while JCSMaskField is selected |
| focusGained | JCSMaskField is selected |
| focusLost | another item is selected after selecting JCSMaskField |

#### 3.2.1.18 JCSDateTimeField

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| AutoSkip | false/true  autoskip disabled/enabled | default value is “false” |
| Background | background color of the date time field | default value is “255,255,255” |
| Comment | user defined information | none |
| Editable | True/false | default value is “True” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of date time field | none |
| Horizontal Alignment | horizontal alignment of text on date time field | default value is “LEADING” |
| Location | x,y coordinate of the date time field | none |
| MustEnter | data must be entered | default value is “false” |
| Name | name of date time field | none |
| ReadOnly | false/true  user can enter data/user can not enter data | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | text of date time field | Only numeric values can be entered |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |
| Width | width of date time field | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while Date Time Field is selected |
| focusGained | Date Time Field is selected |
| focusLost | another item is selected after selecting Date Time Field |

#### 3.2.1.19 JCSPasswordField

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| AutoSkip | false/true  autoskip disabled/enabled | default value is “false” |
| Background | background color of the password field | default value is “255,255,255” |
| Comment | user defined information | none |
| Content Type | content type of password field:  Plain: All characters can be entered  Alpha: Only letters  Numbers: Only numbers  AlphaNumeric: Both letters and numbers, nothing else | default value is “Plain” |
| Editable | true/false  user can enter data/user can not enter data | default value is “true” \*\*\*\* aynı isimli 2 tane var”” |
| Exceptions | extra keys that can be entered to password field. With this option user can define extra keys –like ‘&’- that can be inserted in the text field. | none |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “0,0,0” |
| Global ID | For future use | None |
| Height | height of password field | none |
| LetterCase | letter case of password field; UpperCase/LowerCase/NoCase | default value is “NoCase” |
| Limit | number of characters that can be entered to password field | none |
| Location | x,y coordinate of the password field | none |
| MustEnter | data must be entered | default value is “false” |
| Name | name of password field | none |
| ReadOnly | false/true  user can/not enter data | default value is “false” |
| Tab Order | 0, 1, 2,... | default value is “0” |
| Text | text of password field | default value is “JCSPasswordField” |
| Tool Tip Text | tooltip for the component | none |
| Visible | visibility of component (hidden/unhidden) | default value is “true” |
| Width | width of password field | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | ”Enter” button is pressed while JCSPasswordField is selected |
| focusGained | JCSPasswordField is selected |
| focusLost | another item is selected after selecting JCSPasswordField |

#### 3.2.1.20 JCSPopupMenu

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Enabled | true / false | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Global ID | For future use | None |
| Location | x,y coordinate of the table | none |
| Name | name of the popup component | None |
| Text | The text that will be displayed on top of popup menu | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |

#### 3.2.1.21 JCSPopupItem

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Enabled | true / false | default value is “true” |
| Font | font type and size of text | default value for font type “Dialog”, for font size “12” |
| Global ID | For future use | None |
| Location | x,y coordinate of the table | none |
| Name | name of the popup item component | None |
| Text | The text that will be displayed on top of popup menu | none |
| Visible | visibility of the component (hidden/unhidden) | default value is “true” |

### 3.2.2 JCS CONTAINERS

#### 3.2.2.1 JCSPanel

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the panel | default value is “224,224,224” |
| Comment | user defined information | none |
| Font Phrase | font type and size of text of title | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “32,32,64” |
| Global ID | For future use | None |
| Height | height of the Panel | none |
| Indent | distance between grids on the page layout | default value is “10” |
| Location | x,y coordinate of the table | none |
| Name | name of the Panel | none |
| Title | true/false | default value is “Panel Title” |
| Tool Tip | tooltip for the component | none |
| Visible | true/false : visibility | default value is “true” |
| Width | width of the Panel | None |
| Icon | Background image | None |
| Icon Style | Determines how icon is displayed on panel. Values are  tile : Copies image to all empty area  strech: Resizes images to all display area  center : Centers images. | Default value is tile |
| Title Visible | Show or hide panel border | Default value is “true” |

#### 3.2.2.2 JCSButtonGroup

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the button group | default value is “224,224,224” |
| Comment | user defined information | none |
| Font Phrase | font type and size of text of title | default value for font type  “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “32,32,64” |
| Global ID | For future use | None |
| Height | height of combobox | none |
| Indent | distance between grids on the page layout | default value is “10” |
| Location | x,y coordinate of the table | none |
| Name | name of the button group | none |
| Title | title of the button group | default value is ”ButtonGroup Title” |
| Tool Tip | tooltip for the component | none |
| Visible | true/false : visibility | default value is “true” |
| Width | width of the button group | None |
| Icon | Background image | None |
| Icon Style | Determines how icon is displayed on panel. Values are  tile : Copies image to all empty area  strech: Resizes images to all display area  center : Centers images. | Default value is tile |

#### 3.2.2.3 JCSTabbedPane

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the panel | default value is “224,224,224” |
| Comment | user defined information | none |
| Font Phrase | font type and size of text of title | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “32,32,64” |
| Global ID | For future use | None |
| Height | height of the Panel | none |
| Indent | distance between grids on the page layout | default value is “10” |
| Location | x,y coordinate of the tabbedpane | none |
| Name | name of the Panel | none |
| Title | true/false | default value is “Panel Title” |
| Tool Tip | tooltip for the component | none |
| Visible | true/false : visibility | default value is “true” |
| Width | width of the Panel | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| stateChanged | Fires an event when tab pages are changed |

#### 3.2.2.4 JCSTabpage

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Background | background color of the panel | default value is “224,224,224” |
| Comment | user defined information | none |
| Font Phrase | font type and size of text of title | default value for font type “Dialog”, for font size “12” |
| Foreground | font color of the text | default value is “32,32,64” |
| Global ID | For future use | None |
| Height | height of the Panel | none |
| Indent | distance between grids on the page layout | default value is “10” |
| Location | x,y coordinate of the table | none |
| MovingAllowed | true: moving allowed to move in design time  false: not allowed to move in design time | default value is “true” |
| Name | name of the Panel | none |
| Title | true/false | default value is “Panel Title” |
| Tool Tip | tooltip for the component | none |
| Visible | true/false : visibility | default value is “true” |
| Width | width of the Panel | None |
| Tab Title | Title of inner panel | None |
| Icon | Background image | None |
| Icon Style | Determines how icon is displayed on panel. Values are  tile : Copies image to all empty area  strech: Resizes images to all display area  center : Centers images. | Default value is tile |
| Popup Menu | When user clicks rigth button of mouse, a context menu is displayed. | None |

### 3.2.3 JCS UTILITY BEANS

#### 3.2.3.1 JCSReport

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Comment | user defined information | none |
| Height | height of report | none |
| Location | x,y coordinate of the table | none |
| Report Name | name of the report template to be used | none |
| Visible | true/false | default value is “true” |
| Width | width of report | none |

#### 3.2.3.2 JCSTextPrinter

|  |  |  |
| --- | --- | --- |
| **Property** | **Value - Meaning** | **Restriction/Default Value** |
| Charset | Character set | default value is “cp857” |
| Comment | user defined information | none |
| End Escapes | Comma-separated special escape character codes . For instance, for tearing off printed paper, “Form Feed (12 as byte)” is sent | Values must be separated with commas.  (12,27,14 ,eg.) |
| Printer Name | name of hardware port that printer uses (LPT1) | in most PC’s this value is “LPT1” |
| Start Escapes | Comma-separated special escape character codes which is sent before printing data. These values are generally used to format data or force printer for a predefined function. | none |

#### 3.2.3.3 JCSTimer

|  |  |  |
| --- | --- | --- |
| **Property** | **Value - Meaning** | **Restriction/Default Value** |
| Delay | delay time for event; 0 is no delay, 1 is 1 second, 2 is 2 second etc. | default value is “0” |
| Repeats | whether repeat delay or not | default value is “true” |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| actionPerformed | fires an event when delay time is encountered |

#### 3.2.3.4 JCSShell

|  |  |  |
| --- | --- | --- |
| **Property** | **Value - Meaning** | **Restriction/Default Value** |
| Command line | command line | none |

#### 3.2.3.5 JCSFtp

|  |  |  |
| --- | --- | --- |
| **Property** | **Value - Meaning** | **Restriction/Default Value** |
| Host | host address of ftp | none |
| Local File | local address of ftp | none |
| Remote File | remote address of ftp | none |

#### 3.2.3.6 JCSFileSystem

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| File | absolute path of the file on which JCSFileSystem works | none |

#### 3.2.3.7 JCSEncryption

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Input File | file name (with path) to be encryped | none |
| Output File | file name where to store the decrypted file | none |

#### 3.2.3.8 JCSImage

|  |  |  |
| --- | --- | --- |
| **Property** | **Value - Meaning** | **Restriction/Default Value** |
| Comment | user defined information | none |
| Global ID | For future use | none |
| Height | height of the image | none |
| Image | The name of the image file | none |
| ImageStyle | Normal: The picture ise shown in its original size  Stretch: The picture is fitted to the image size | default value is “Normal” |
| Location | x,y coordinate of the table | none |
| Name | name of the image component | none |
| Visible | true/false : visibility | default value is “true” |
| Width | width of the image | none |

|  |  |
| --- | --- |
| **Event Type** | **Meaning** |
| mouseClicked | fires an event when mouse left button is clicked |
| mouseDoubleClicked | fires an event when mouse left button is double-clicked |

#### 3.2.3.9 JCSSmartCard

|  |  |  |
| --- | --- | --- |
| **Property** | **Value - Meaning** | **Restriction/Default Value** |
| Name | name of the component | None |

#### 3.2.3.10 JCSDateUtility

|  |  |  |
| --- | --- | --- |
| **Property** | **Value – Meaning** | **Restriction/Default Value** |
| Name | name of the component | None |

## 3.3 Component Design

### 3.3.1 JCS BEANS

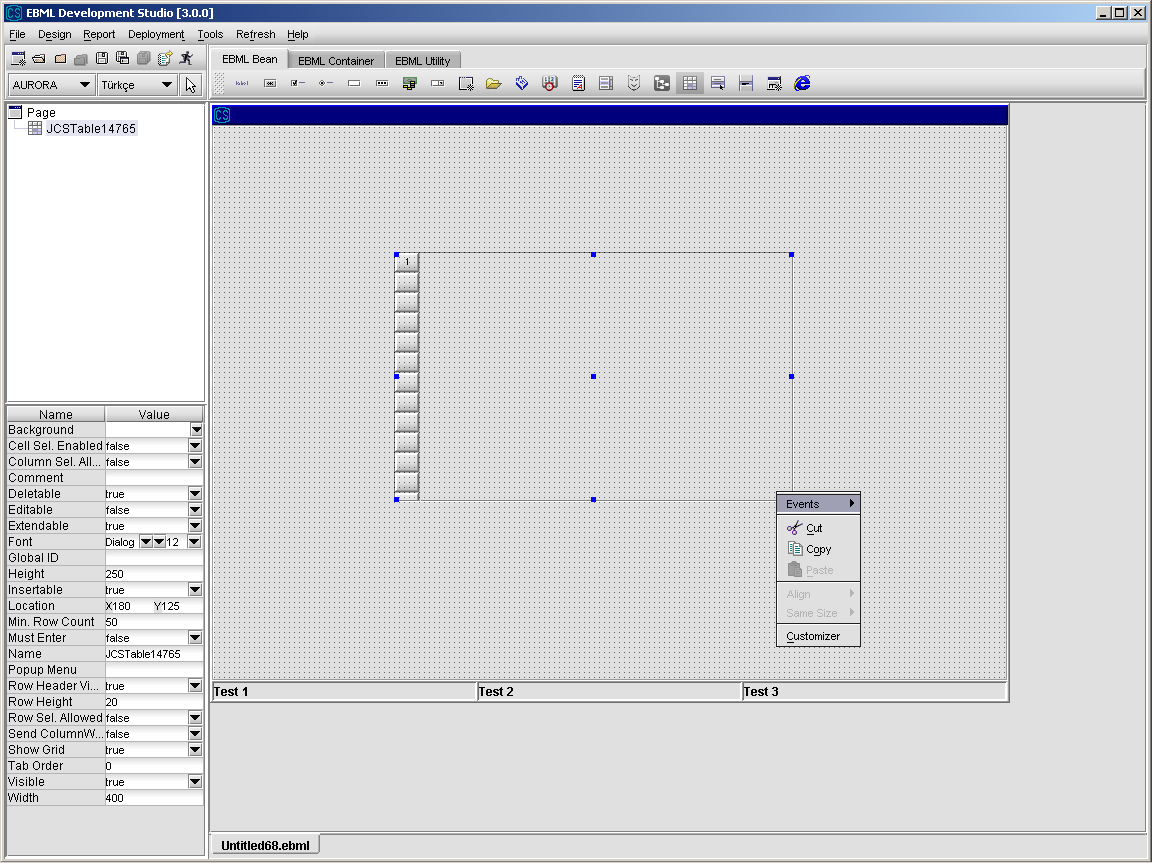
All beans has getter and setter methods for their properties explained in section 3.2.1. Other methods that are specific to beans are explained in the following sections. All beans that can have focus have a grabFocus method that is used to transfer the focus to the bean.

In EDS Design Environment, JCSComponents on the screen may be moved by one pixel to the left, right, up and down using “Shift + Arrows”.

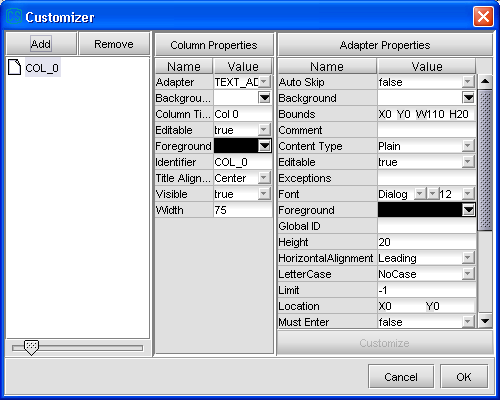
#### 3.3.1.1 Table Component

Table is used to display data in column-row format.

1. Select Table component from the toolbar and drop into page.
2. Set the properties of the Table component.

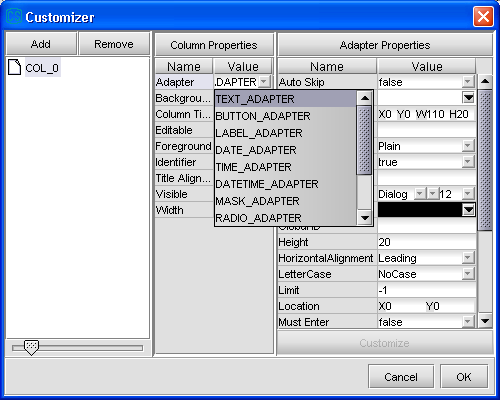


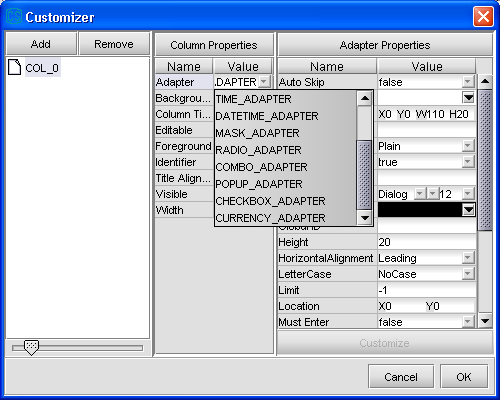
1. Add columns using the customizer for table component. Developer can add new column, remove column, update column and also set editable and visible properties of the table using the Customizer Dialog. In order to open Customizer, right click on the table and click “Customizer” on the popup menu.

****

1. Click on the Add button, in order to add a column to the table. When Add button is clicked a new column with default properties is added. Default properties can be seen above.
2. Set the Column Title and Column Identifier in Column Properties part in the center.
3. Set Editable and Visible properties of the column.
4. Select the Adapter of the column. There are different column adapters. Using column adapters it is possible to define a column cell as a text field, check-box, popup, combobox, date field, text field, mask field or time field.
5. After selecting the adapter of the column, you can also edit adapter specific properties from Adapter Properties part at the right side.
6. In order to update properties of a column, select it and change the property that you want to update. It will be automatically updated. In order to remove a column, select it and click Remove button.
7. Clicking the OK button saves the table design and returns you back to the main screen.

Different types of column adapters are available as shown on the figure below.



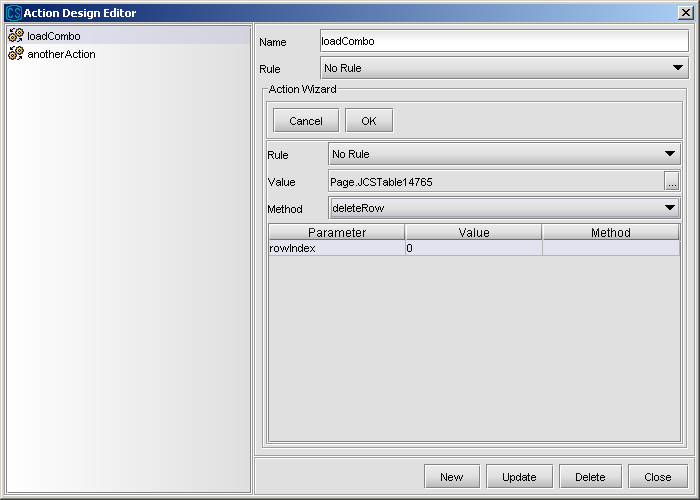


##### 

##### 3.3.1.1.1 Table Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Table component and its columns.

Details about “Bean Action” are given in ”Action Design” section.



**addRow(rowData) :** adds a new row with the given *rowData*

**cleanup:** used for cleaning the selection from the table

**clear:** used for deleting all items

**countOccurences(columnIdentifier,value):** counts the occurrences of *value* in the column identified as *columnIdentifier*

**deleteRow:** used fordeleting the selected row

**deleteRow(rowIndex):** used for deleting the row whose index is *rowIndex*

**export(fileType,filePath):** used for exporting the table data to the file named *filePath* in a format determined with *fileType*.

**getActualRowCount:** used for getting the row count used currently, not the count of visible rows

**getColumnName(columnIndex):** used for getting the name of the column whose index is *columnIndex*

**getItems:** returns the table date in XML format

**getSelectedColumn:** used for getting the index of selected column

**getSelectedRow:** used for getting the index of selected row

**getValueAt(rowIndex,columnIdentifier):** used for getting the value of cell whose row index is *rowIndex* and its column is identified as *columnIdentifier*

**insertRow(rowIndex,rowData):** used for inserting rowData to the row index *rowIndex*

**selectFirst(columnIdentifier,value):** used for selecting first row below selected row in a circular case. If end-of-table state is reached , it goes on to trace from beginning. Value can be filtered with “%”. For instance,

**value :** “xyz” means “look for string exact matches with xyz”

**value :** “%xyz” means “look for string ends with xyz”

**value :** “%xyz%” means “look for string substring of xyz”

**value :** “xyz%” means “look for string starts with xyz”

**selectFirst(columnIdentifier,rowIndex,value):** Same as “selectFirst” with two parameters. Difference is that searching starts from given row index.

**setColumnHidden(columnIdentifier,hidden):** used for settingcolumn visibility

locally

**setColumnName(columnIndex,name):** used for setting column name locally

**setRowBGColor(color):** used for setting the background color of the selected row

**setRowBackground(rowIndex,color):** used for setting the background color of the row whose index is rowIndex

**setCellBackground(rowIndex,columnIndex,color):** used for setting the background of cell whose row index is *rowIndex* and its column index is identified as *columnIndex*

**setCellForeground(rowIndex,columnIndex,color):** used for setting the foreground of cell whose row index is *rowIndex* and its column index is identified as *columnIndex*

**setRowFGColor(color):** used for setting the foreground color of the selected row

**setRowForeground(rowIndex,color):** used for setting the foreground color of the row whose index is rowIndex

**setSelectedRow(rowIndex) :** used for setting a row as selected

**setValueAt(value,rowIndex,columnIdentifier):** used for setting the value of cell whose row index is *rowIndex* and its column is identified as *columnIdentifier* to *value*

**sortColumnAsc(columnIdentifier):** used for sorting the column whose identifier is *columnIdentifier* in ascending order

**sortColumnAscIgnoreCase(columnIdentifier):** used for sorting the column whose identifier is *columnIdentifier* in ascending order ignoring the case

**sortColumnDesc(columnIdentifier):** used for sorting the column whose identifier is *columnIdentifier* in descending order

**sortColumnDescIgnoreCase(columnIdentifier):** used for sorting the column whose identifier is *columnIdentifier* in descending order ignoring the case

**sumColumn(columnIdentifier):** used for summing the values in the column whose identifier is *columnIdentifier*

**updateRow(rowData):** used for updating the data of the selected row to *rowData*

**updateRow(rowIndex,rowData):** used for updating the data of the row whose index is *rowIndex* to *rowData*

**Samples for Table Operations :**

**1. Filling table with data returned from a service**

CSBag outBag = CSBagFactory.createBag();

CSBag comboBag = CSBagFactory.createBag();

comboBag.put(SKY\_CONFIG\_LIST\_NAME, 0, -1, "SIFIR");

comboBag.put(SKY\_CONFIG\_LIST\_NAME, 1, -1, "BIR");

comboBag.put(SKY\_CONFIG\_LIST\_NAME, 2, -1, "IKI");

comboBag.put(SKY\_CONFIG\_LIST, 0, -1, "0");

comboBag.put(SKY\_CONFIG\_LIST, 1, -1, "1");

comboBag.put(SKY\_CONFIG\_LIST, 2, -1, "2");

outBag.put(SKY\_TABLE, comboBag); //filling combobox on the table

outBag.put(SKY\_TABLE, 0, "SKY\_NUMBER", **new** CSCurrency("123123"));

outBag.put(SKY\_TABLE, 0, "SKY\_DECISION", **true**);

outBag.put(SKY\_TABLE, 0, "SKY\_SUBJECT", "");

outBag.put(SKY\_TABLE, 0, "SKY\_DATE", **new** CSDate("25112002"));

outBag.put(SKY\_TABLE, 0, "SKY\_MAIL\_TEXT", "text data");

outBag.put(SKY\_TABLE, 0, "SKY\_DEFECT", "1!5464H4");

outBag.put(SKY\_TABLE, 0, "SKY\_TIME", **new** CSTime("121212"));

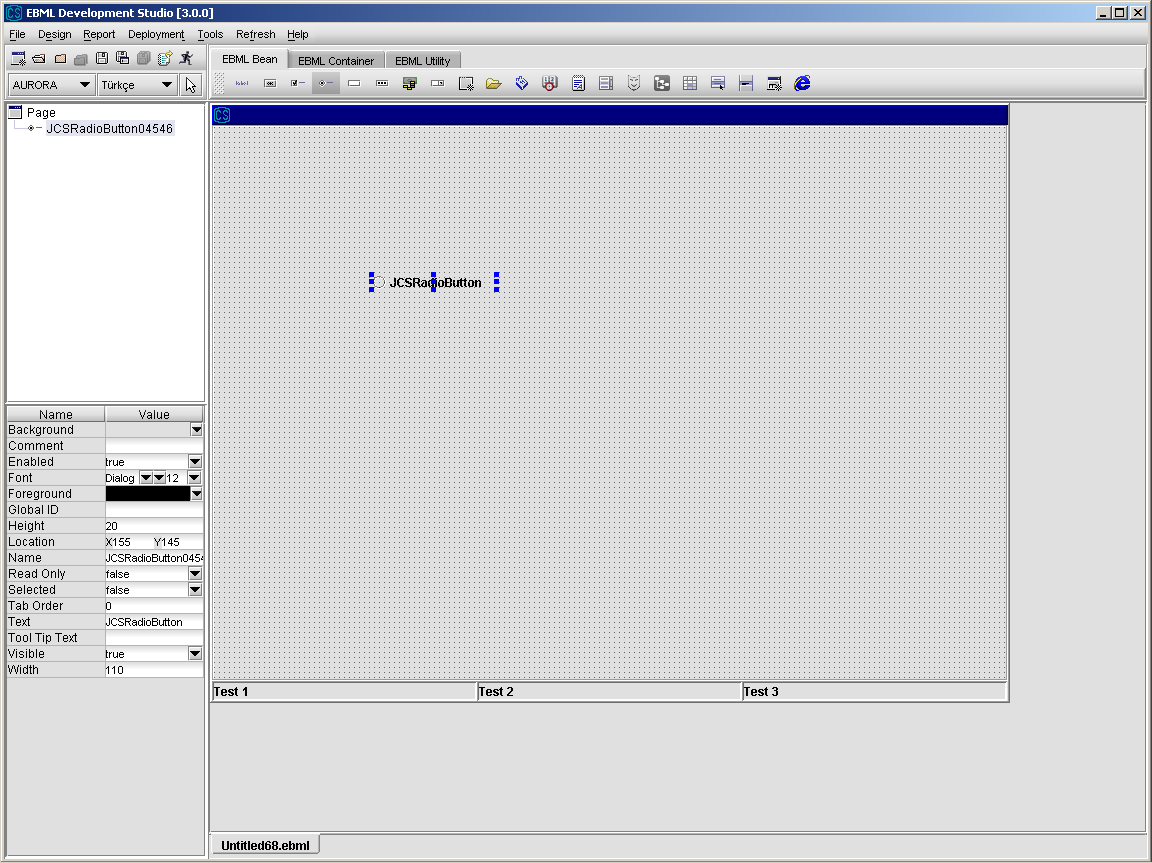
outBag.put(SKY\_TABLE, 0, "SKY\_CONFIG\_LIST", "BIR");

**return** outBag;

#### 3.3.1.2 Radio Button Component

Radio Buttons are used for displaying a group of two or more options that are mutually exclusive- that is, it can be selected only one radio button at a time.

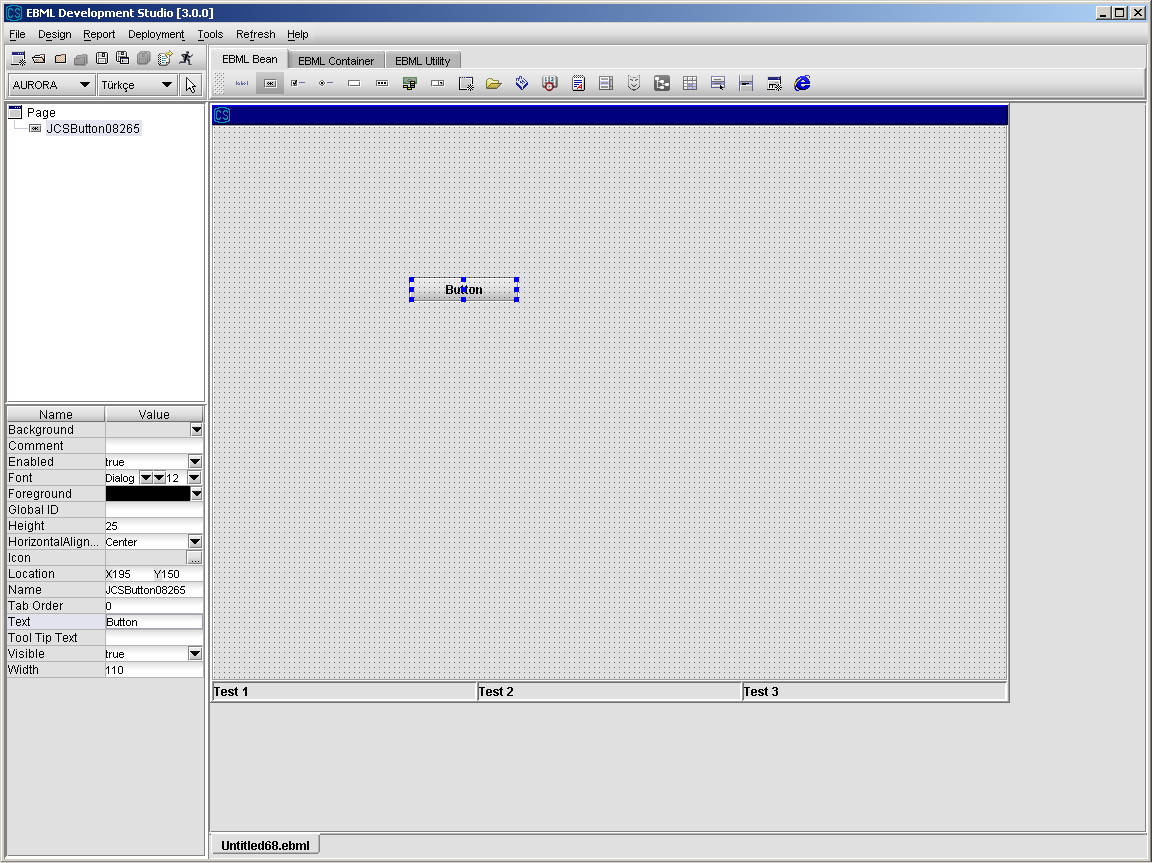
1. Select Radio Button component from the toolbar and drop into page.
2. Set the properties of Radio Button component.

****

**3.3.1.3 Button**

Buttons are used for starting an action or a set of actions.

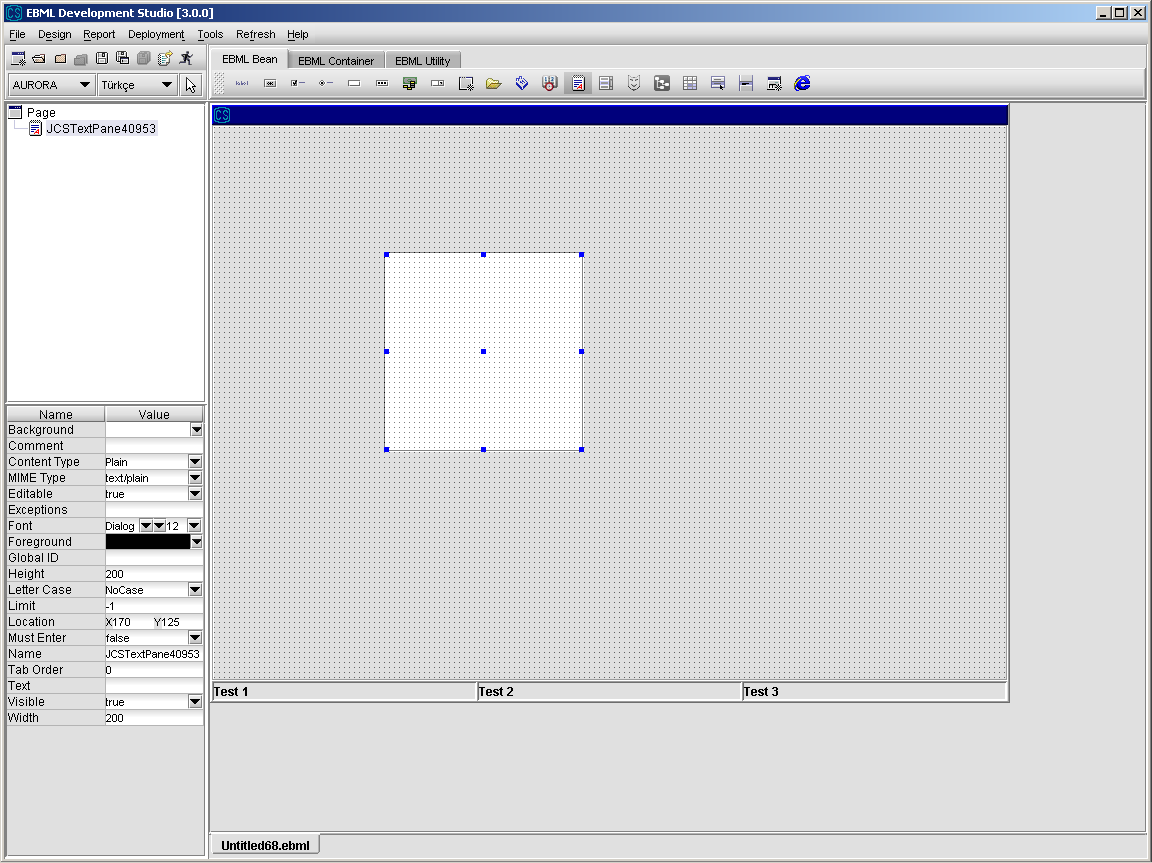
1. Select Button component from the toolbar and drop into page.
2. Set the properties of Button component.



#### 3.3.1.4 Text Pane

Text Pane is used for displaying or entry of large data.

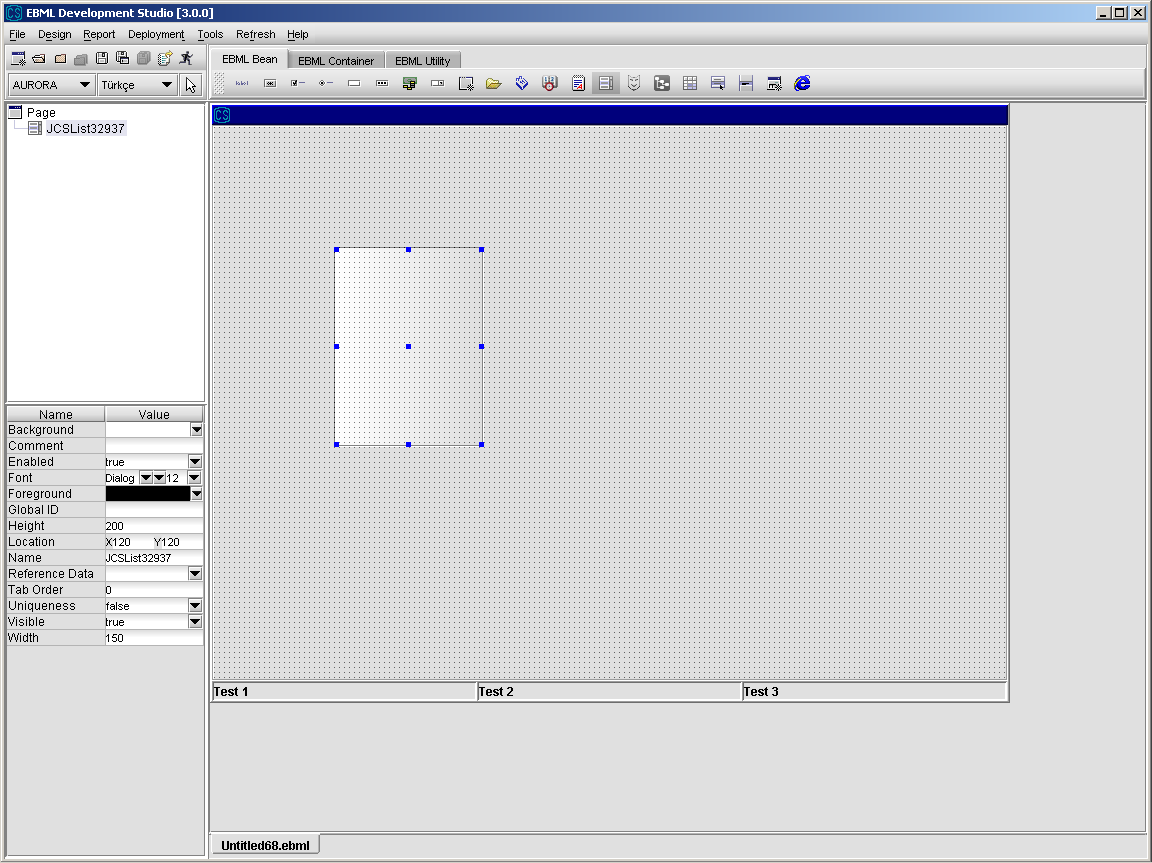
1. Select Text Pane component from the toolbar and drop into page.
2. Set the properties of Text Pane component.



#### 3.3.1.5 List

In many cases, it is quicker and easier to select a value from a list of choices also helps to ensure that the value that’s entered in a field is correct. The list in a list box consists of a row of data.

1. Select List component from the toolbar and drop into page.
2. Set the properties of List component.

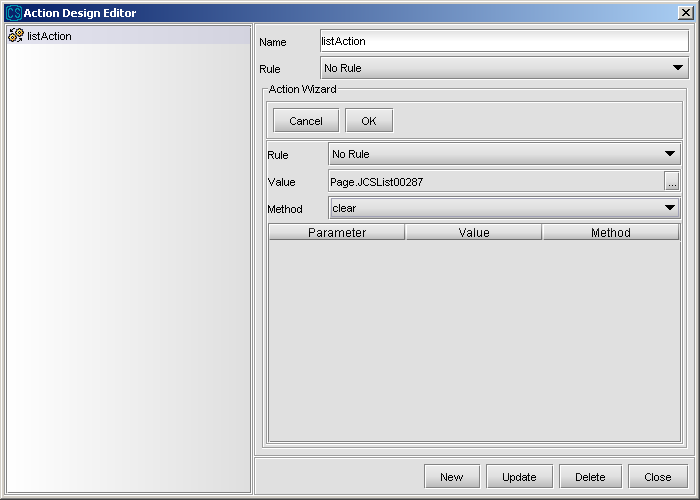
****

1. The content of the list can be filled from a reference data, by setting its Reference Data property or via a remote call.

##### 3.3.1.5.1 ListOperations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of List component.

Details about “Bean Action” are given in ”Action Design” section.



**setSelectedText(key):** used for selecting the item whose key value is *key*

**setSelectedValue(value):** used for selecting the item whose value is *value*

**getSelectedText:** used for getting the text of the selected item

**getSelectedValue:** used for getting the text of the selected item

**getSelectedIndex:** used for getting the index of the selected item

**setSelectedIndex(index):** used for selecting the item whose index is index

**addItem(key,value):** used for adding a new item to the list with key,value pair

**cleanup:** used for cleaning the selection from the list

**removeItem(itemName):** used for removing the item whose key value ise itemName

**clear:** used for deleting all items

**sortAsc:** used for sorting the list in ascending order

**sortDesc:** used for sorting the list in descending order

**getActualRowCount:** used for getting the item count in the list

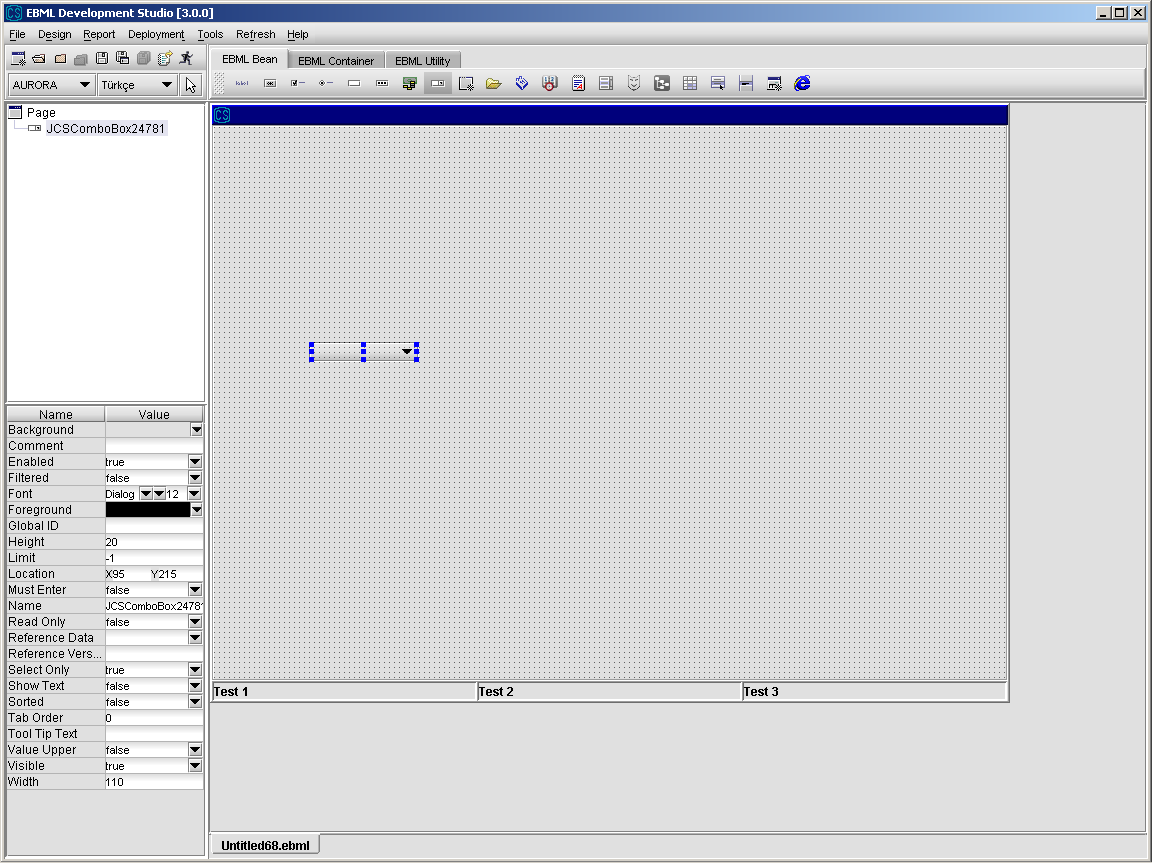
**sortAscIgnoreCase:** used for sorting the list in ascending order ignoring the case

**sortDescIgnoreCase:** used for sorting the list in descending order ignoring the case

#### 3.3.1.6 Combobox

ComboBox is used for displaying data entered statically, from reference data tables or from service calls. Data on the combobox consists of two parts; “key” and “value”. “key” is what is behind the screen and “value” is what is shown on the combobox.

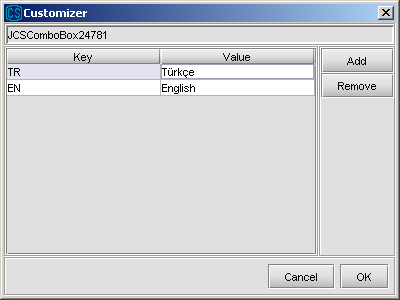
1. Select ComboBox component from the toolbar and drop into page.
2. Set the properties of the ComboBox component.



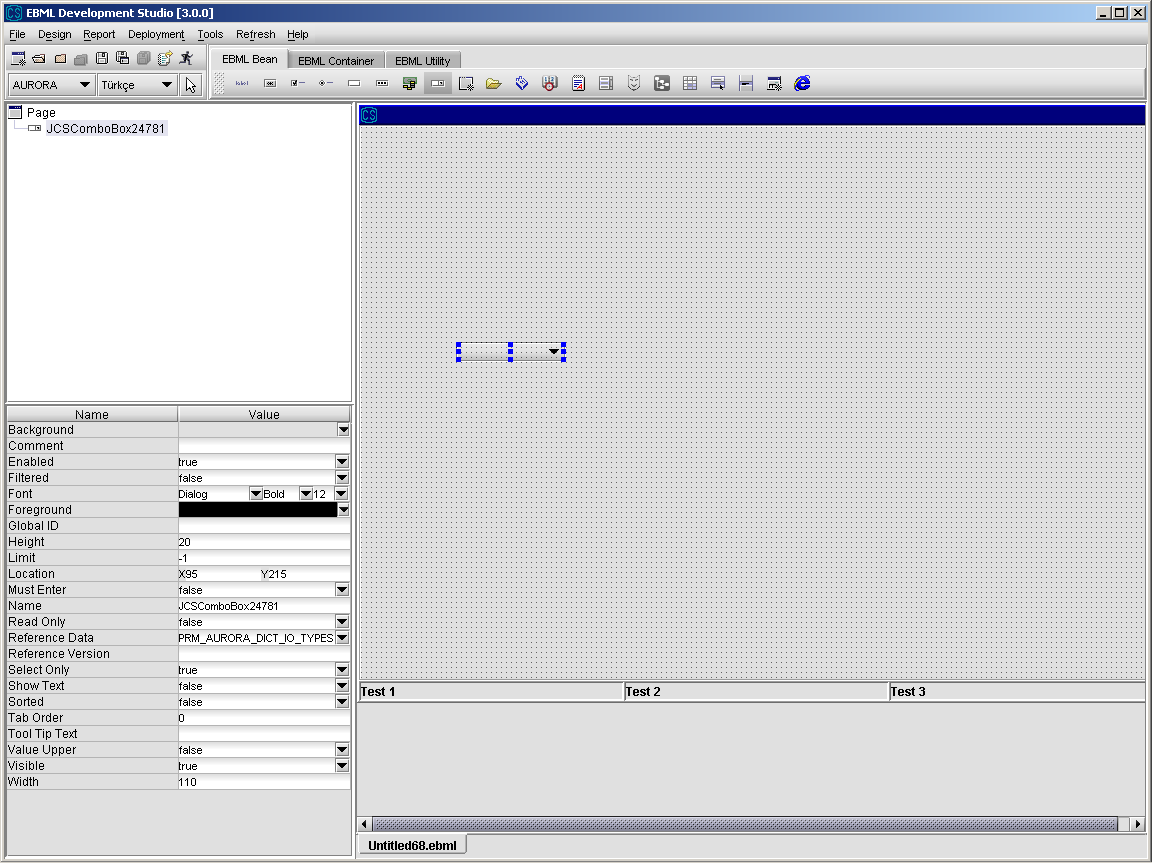
1. Set the data of the combobox component. There are 4 ways to set the data of a

combobox component, which are explained below.

1. Combobox data can be entered (key-value pairs) statically by using Customizer. Combobox component has a customizer to add static data to combobox in design environment. Using Customizer Dialog, developer can add, update and modify combobox data easily.



1. Combobox data can be set using reference data tables without any action. In design environment, set the “Reference Data” property of the combobox to a value of a reference data table from which you want to get data. In the following example, combobox data is set to a reference data called PRM\_AURORA\_DICT\_IO\_TYPES.



1. Combobox data can be set to reference data with reference call. This can be done by using "RefData Name" property of the combobox.
2. Combobox data can be set using remote call actions. Definition of remote call actions are explained in detail in ”Action Design” section.

**Samples for Combobox Operations :**

**1. Filling combobox with data returned from a service**

CSBag outBag = CSBagFactory.createBag();

outBag.put(COMBO\_LIST, 0, 0, "SIFIR");

outBag.put(COMBO\_LIST, 1, 0, "BIR");

outBag.put(COMBO\_LIST, 2, 0, "IKI");

outBag.put(COMBO\_LIST, 0, 1, "0");

outBag.put(COMBO\_LIST, 1, 1, "1");

outBag.put(COMBO\_LIST, 2, 1, "2");

**return** outBag;

In the above example, we put a multidimensional key “COMBO\_LIST” to outBag. Its first index gives its order in the combobox starting from 0. Second index is either 0 or 1. 0 means the given value is the key of the item, 1 means the given value if the value of the item. For example when we say

outBag.put(COMBO\_LIST, 0, 0, "SIFIR");

the key of the first item in the combobox will be “SIFIR” and when we say

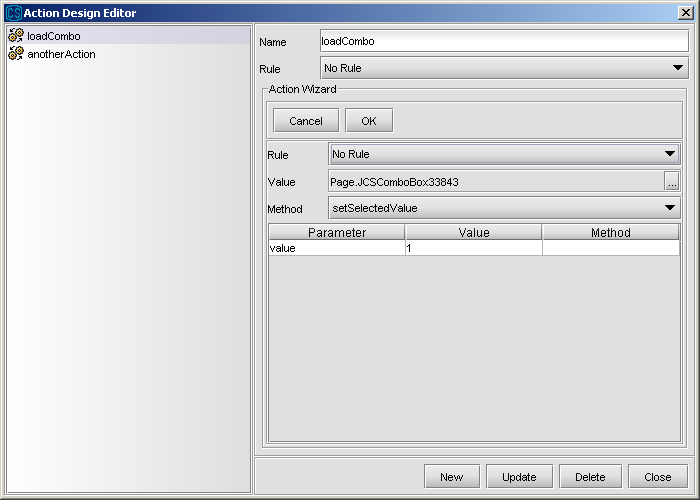
outBag.put(COMBO\_LIST, 0, 1, "0");

the value of the first item in the combobox will be “0”.

##### 3.3.1.6.1 ComboBox Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Combobox component.

Details about “Bean Action” are given in ”Action Design” section.



**addItem(key,value) :** This method will add values specified by the user (key– value) to the combo box.

**cleanup :** used for cleaning the selection from the combobox

**clear :** used for deleting all items

**getSelectedText :** returns text of selected row

**getSelectedIndex :** returns index of selected row

**getSelectedValue :** returns value of selected row

**getItemCount :** returns count of combo items

**refreshReferenceData :** used for changingreference data of combobox locally

**setSelectedText (key) :** sets data by using text

**setSelectedIndex(index) :** sets data by using index

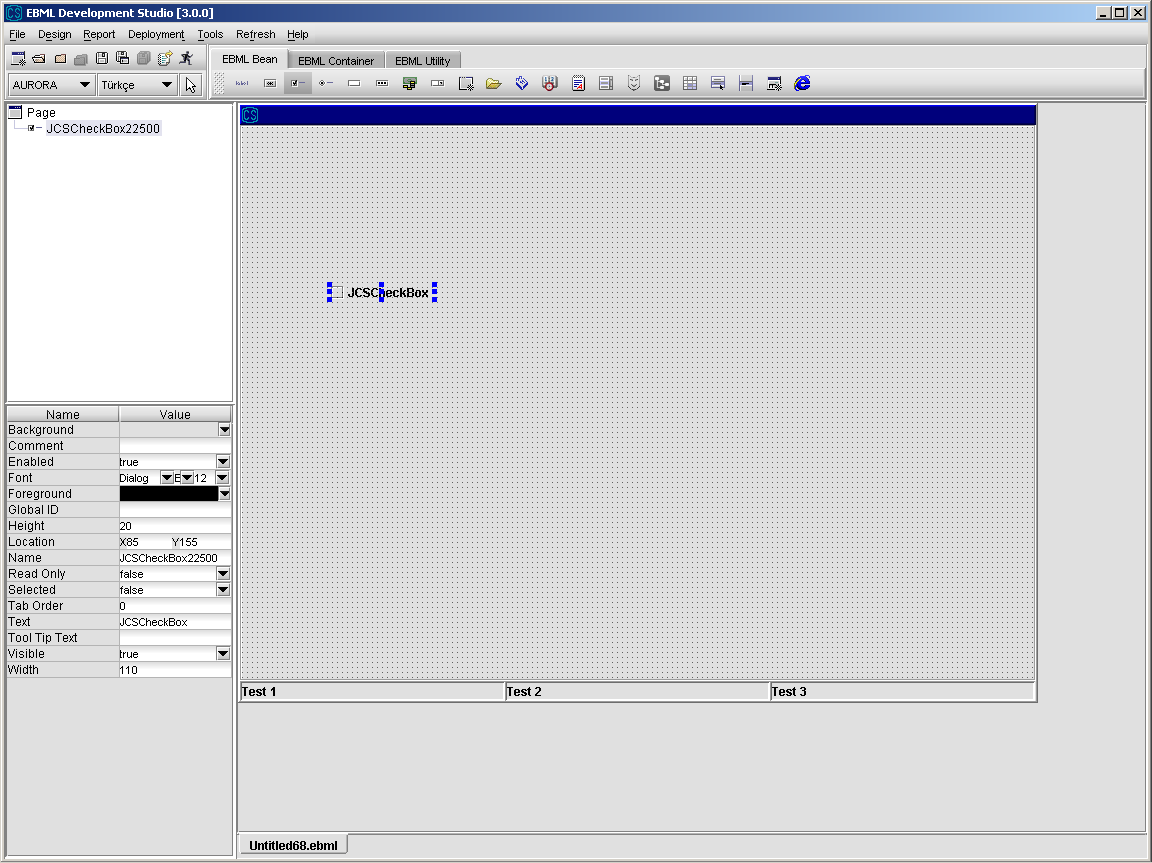
**setSelectedValue (value) :** sets data by using value

**filter(key) :** filters combo items whose filter values equal to key

#### 3.3.1.7 Check Box

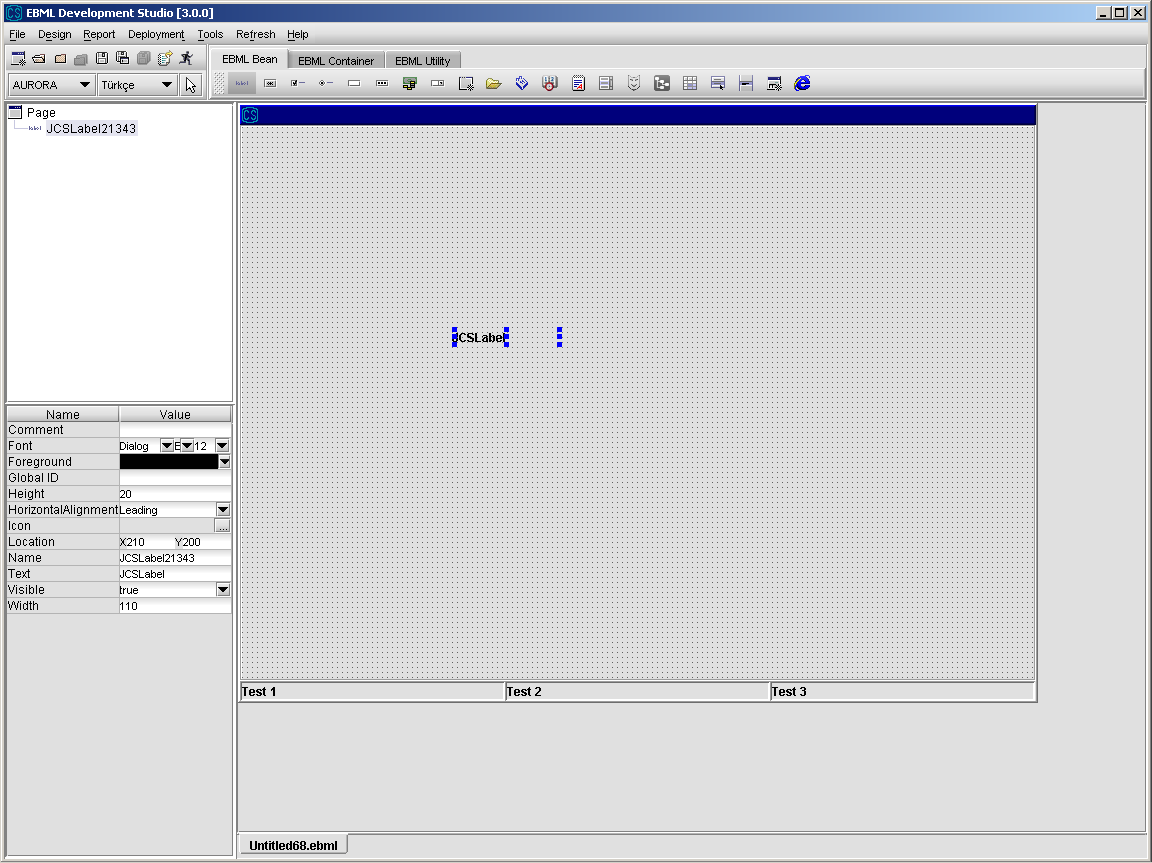
Check boxes are used to display options that are not mutually exclusive- that is, it can be selected more than one check box at a time.

1. Select Check Box component from the toolbar and drop into page.
2. Set the properties of Check Box component.

****

#### 3.3.1.8 Label

1. Select Label component from the toolbar and drop into page.
2. Set the properties of Label component.



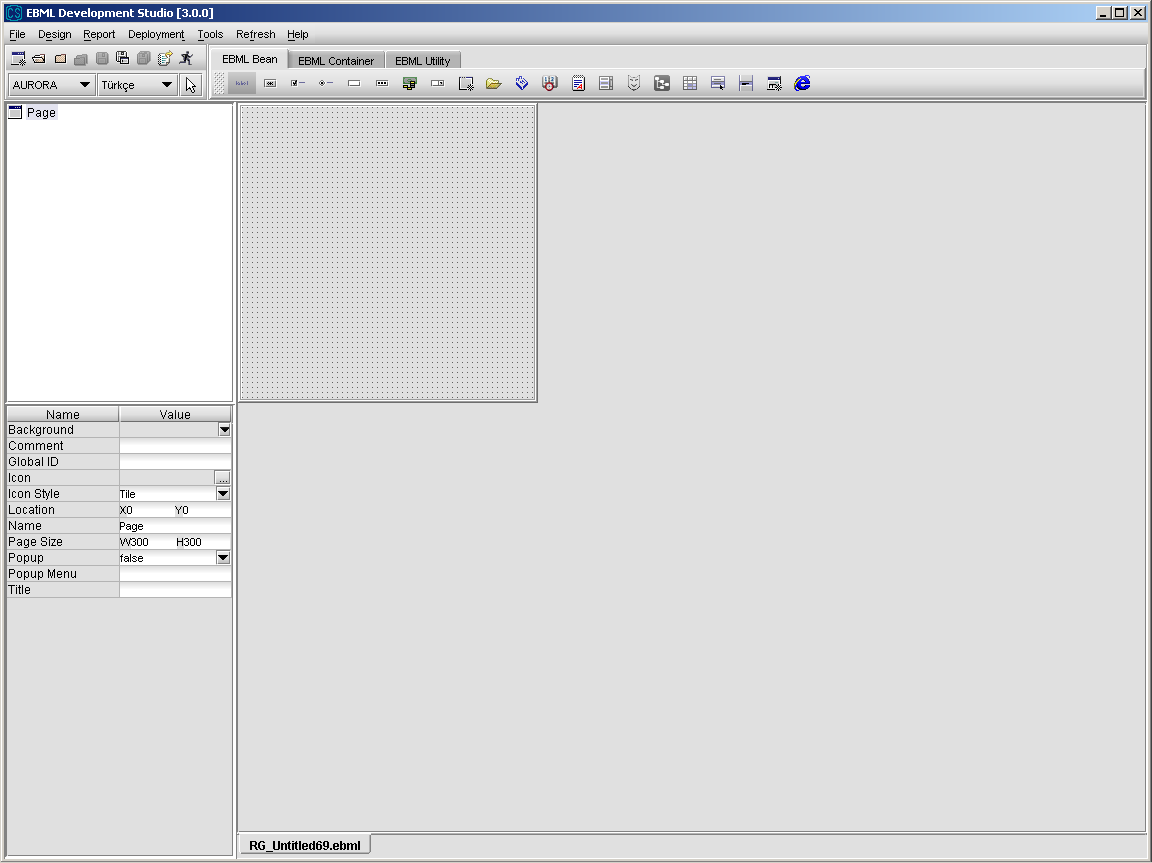
#### 3.3.1.9 Region

Region component is used for standardizing and re-using the repeated parts in screens. If some group of component is used more than one place, prepare a region containing them and refer that region when you want use them.

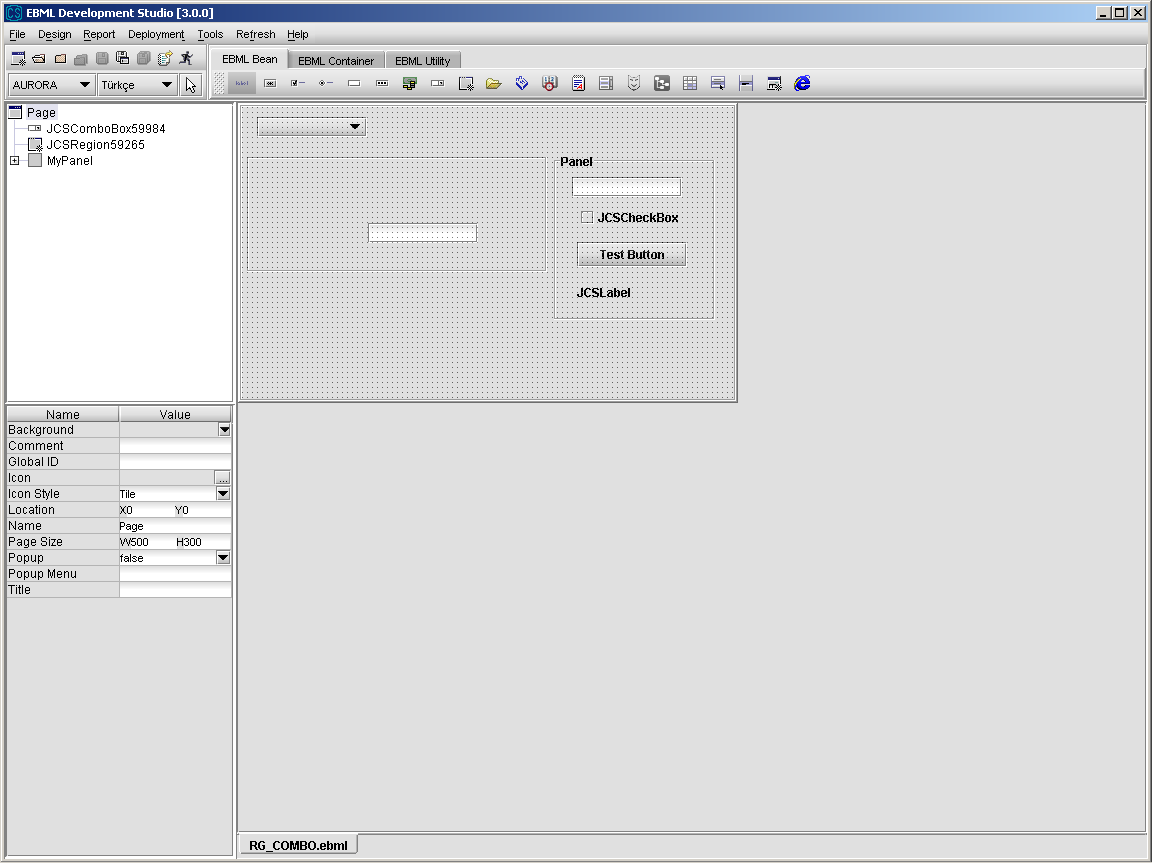
A region can also be used as a “pop-up screen” that pops up on the underlying screen when a specific event takes place such as a button-click. Developer should set “Popup” property to “True” if the region is intended to be hidden and to pop up when a specific event occurs.

##### 3.3.1.9.1 Designing New Region (Region Template)

* Select Region under File\New menu item. An empty region page will appear like below.



* You can now design the region by selecting the components from the toolbar and dropping into page. While designing a region, developer can use any type of the component (label, text field, popup, buttons, panel,...) from the toolbar. Another region can also be placed inside a region.
* After you finish your design, save the region by giving it a name and deploy it to the server by clicking Deploy button. Now, your new region can be used by screens or other regions.

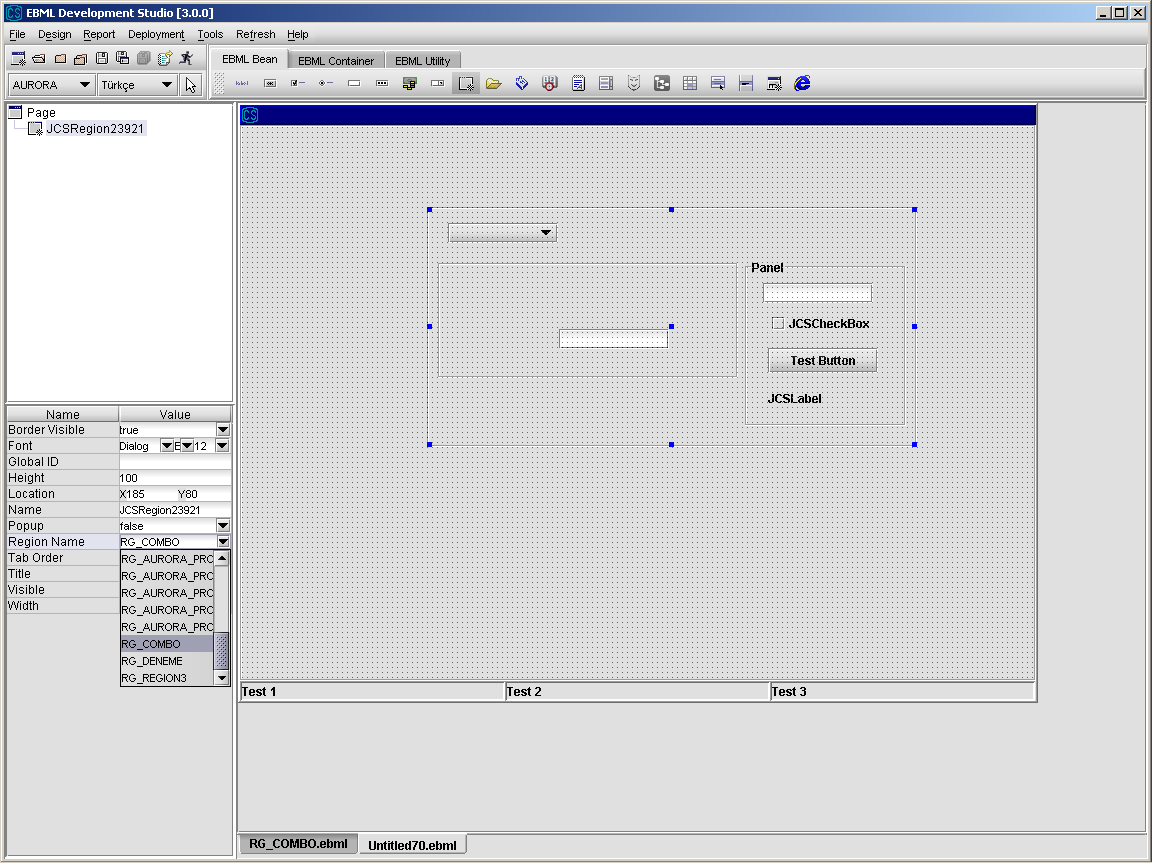


##### 3.3.1.9.2 Using Previously Designed Region

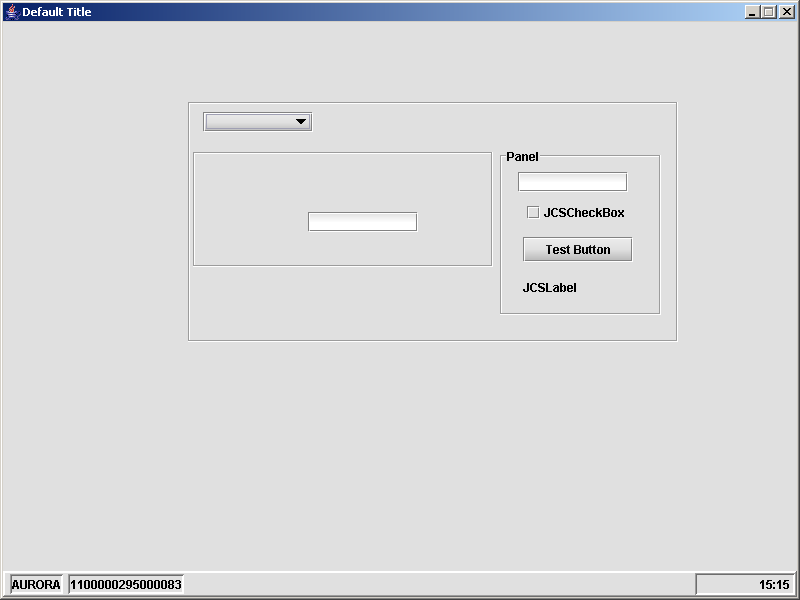
Developer can use the previously defined regions while designing a screen.

1. Select the region component and drag into screen
2. Select a region name from Region Name property. The new region inherits all the

properties of the previously defined region template.



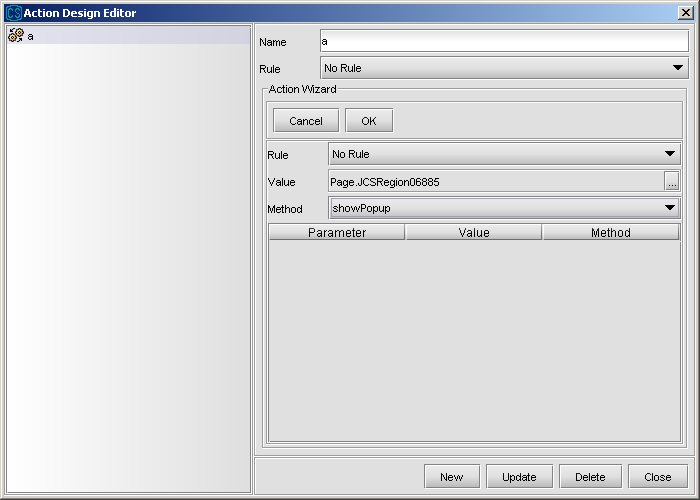
1. Set the height and width of the region on the screen.
2. After completing the design of the screen Save and Deploy the file.
3. Test the file with CMS by clicking the “Test” button.



##### 3.3.1.9.3 Region Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Region component.

Details about “Bean Action” are given in ”Action Design” section.



**renderRegion(regionName) :** user for rendering the region whose name is regionName dynamically

**showPopup :** if popup property of region is true, this method shows the region as popup

**closePopup :** if popup property of region is true, this method closes the shown region

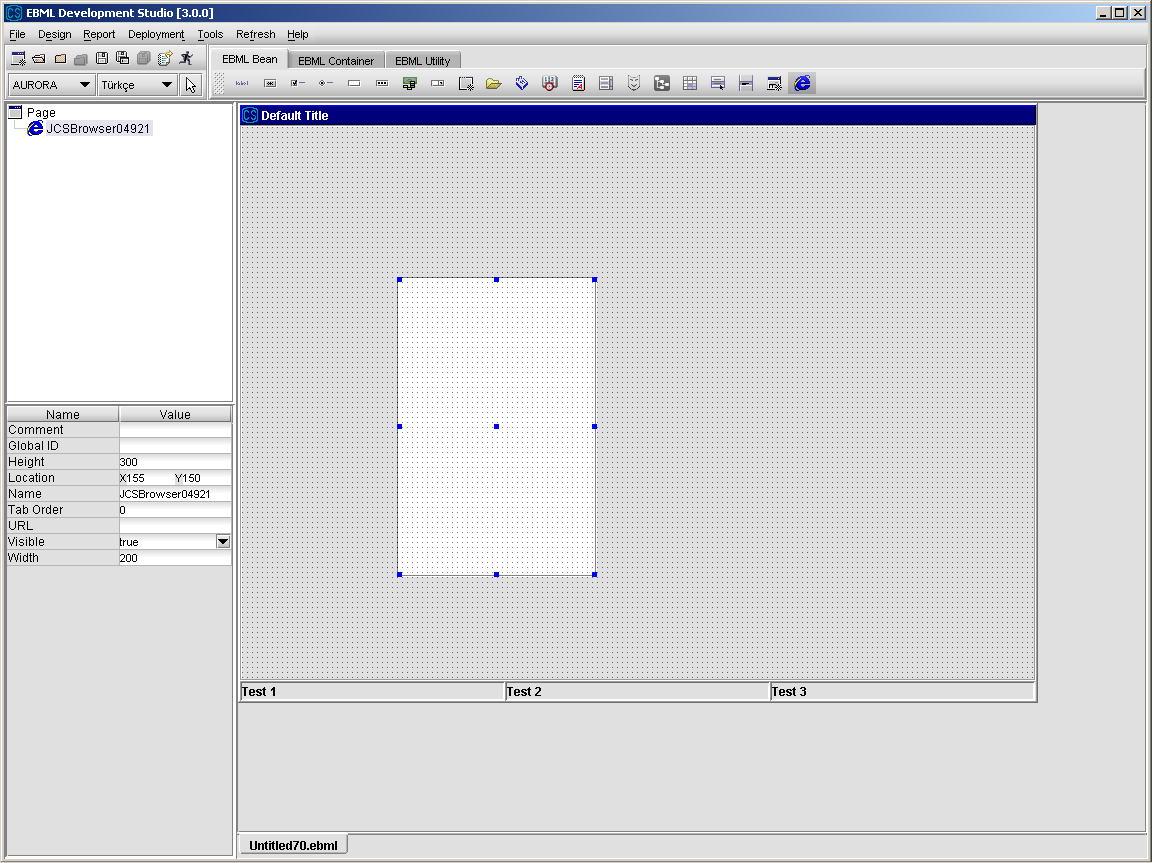
**cleanup :** calls cleanup method for all the beans in the region

**setEnabledAll :** Enable all beans in this container.

#### 3.3.1.10 JCSBrowser

Browser component is used to show html pages.

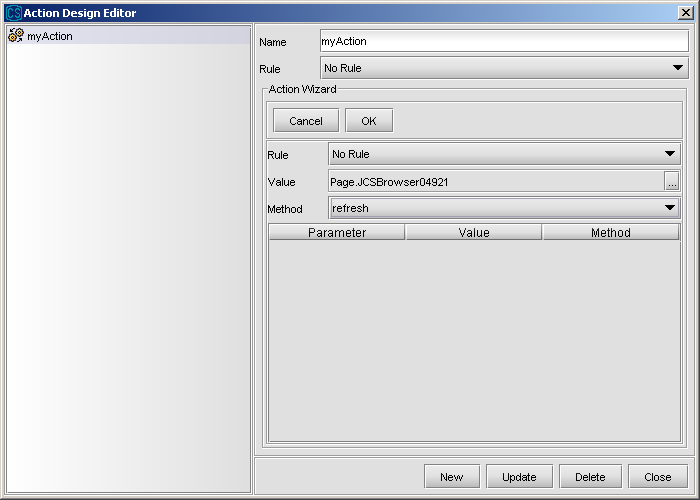
1. Select Browser component from the toolbar and drop into page.
2. Set the properties of Browser component.



##### 3.3.1.10.1 Browser Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Browser component and its columns.

Details about “Bean Action” are given in ”Action Design” section.

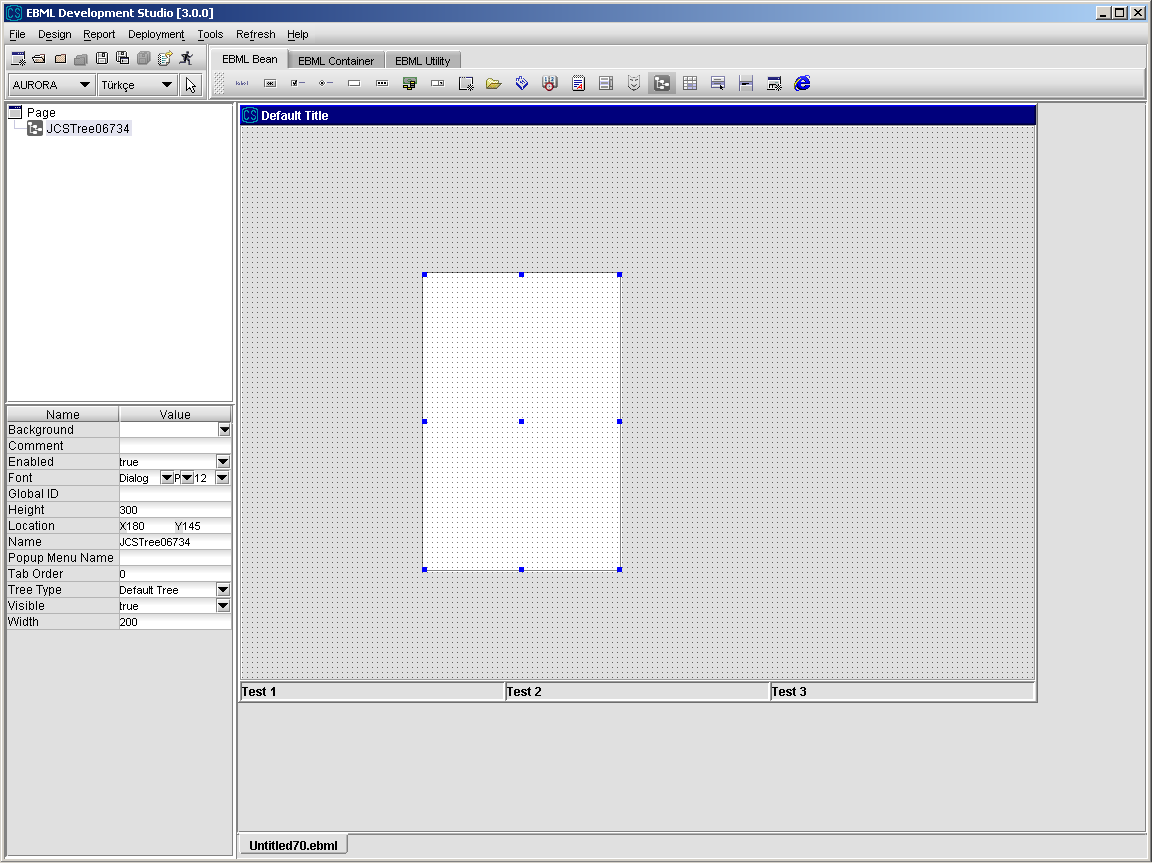


**refresh :** refreshes the stated URL

#### 3.3.1.11 JCSTree Component

JCSTree Component is used to display the information in a tree format.

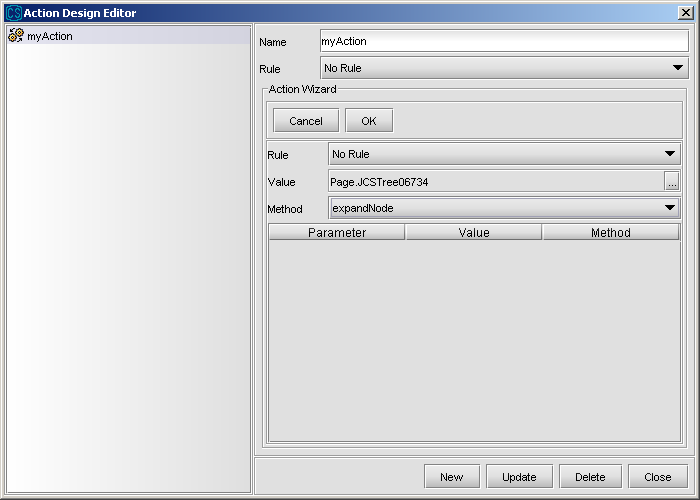
1. Select Tree component from the toolbar and drop into page.
2. Set the properties of the Tree component.



##### 3.3.1.11.1 Tree Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Tree component.

Details about “Bean Action” are given in ”Action Design” section.



**addNode(OID) :** used to add a new node to tree

**cleanup :** used to delete the tree

**collapseNode :** used for collapsing selected nodes of tree

**expandNode :** used for expanding selected nodes of tree

**getNodeData(key) :** returns data of the selected tree node

**getSelectedNode :** returns OID of the selected tree node

**removeNode :** removes selected node with its children nodes from tree

**setNodeData(key,value) :** sets data of the selected node to the given data

**setSelectedNode(OID) :** used for selecting the tree node with given OID

**expandAllNode :** used for expanding all the nodes in the tree

**sortAsc(key) :** used for sorting top level nodes under the selected node with respect to *key,* e.g “CORE\_NAME” in ascending order

**sortAscAll(key) :** used for sorting all nodes under the selected node with respect to *key,* e.g “CORE\_NAME” in ascending order. All nodes are sorted in their levels

**sortDesc(key) :** used for sorting top level nodes under the selected node with respect to *key,* e.g “CORE\_NAME” in descending order

**sordDescAll(key) :** used for sorting all nodes under the selected node with respect to *key,* e.g “CORE\_NAME” in descending order. All nodes are sorted in their levels

**search(key,value) :** used for searching the nodes under the selected one by looking at the *key* of nodes and comparing them with the *value*. First matching node is selected, if any.

**selectParentNode :** used for selecting the parent node of the currently selected node

Following bagkeys are the fixed bagkeys used for tree operations:

CORE\_NAME : used for “tree node name”

CORE\_IMAGE : used for “tree node image”

CORE\_OID : used for “tree node OID”

CORE\_PARENTOID : used fro “tree node parent OID”

\*Developer can add icons on the JCSTree nodes using CORE\_IMAGE bagkey.

**Example:**

This example shows how to construct a tree using a service. A tree with 4 nodes is

constructed using the values returning from service call as a two dimensional bag.

r='0' c='CORE\_OID' --- 0

r='0' c='CORE\_PARENTOID' ---

r='0' c='CORE\_NAME' --- BEN ROOTUM

r='1' c='CORE\_OID' --- 1

r='1' c='CORE\_PARENTOID' --- 0

r='1' c='CORE\_NAME' --- SOL

r='2' c='CORE\_OID' --- 2

r='2' c='CORE\_PARENTOID' --- 0

r='2' c='CORE\_NAME' --- SAG

r='3' c='CORE\_OID' --- 3

r='3' c='CORE\_PARENTOID' --- 2

r='3' c='CORE\_NAME' --- SAG ALT

#### 3.3.1.12 JCSFilePicker

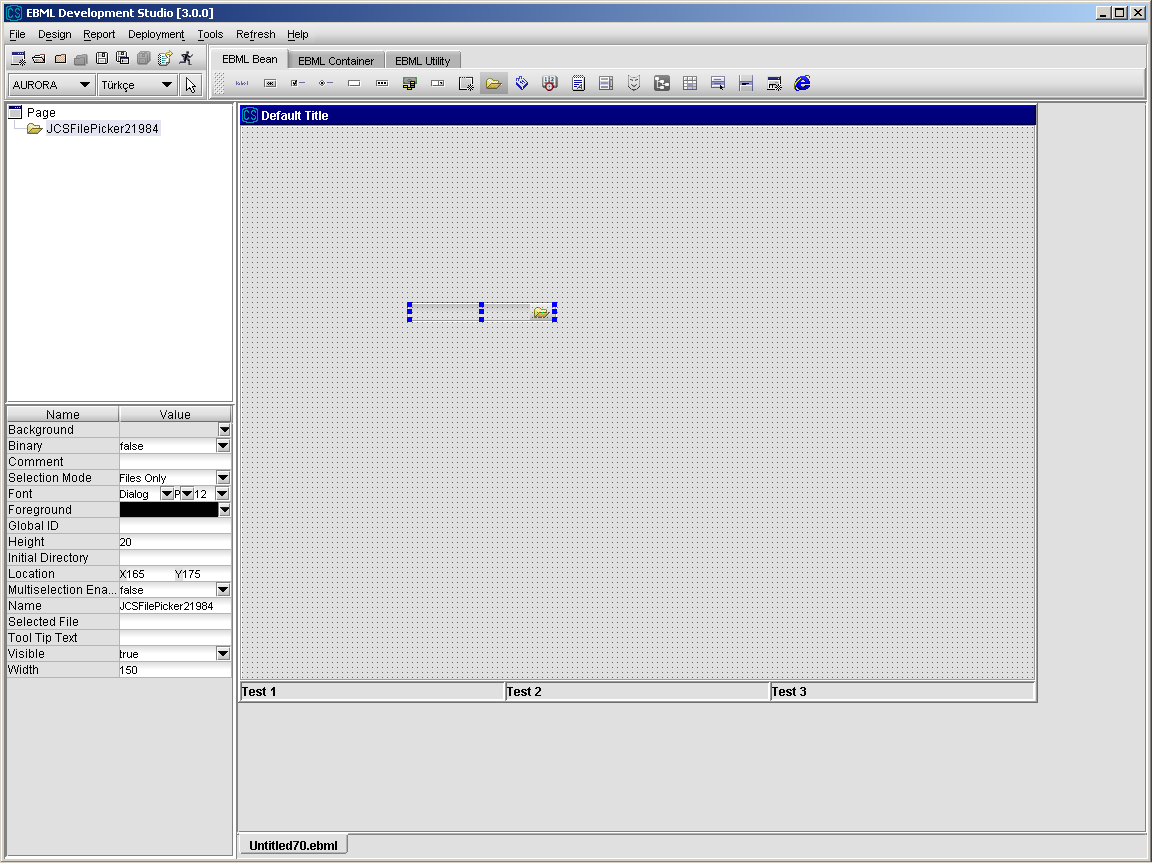
JCSFilePicker Component is used to select files and read their contents.

1. Select FilePicker component from the toolbar and drop into page.
2. Set “Selection Mode” property to the desired value.

FILES\_ONLY : lists only files

DIRECTORIES\_ONLY : lists only directories

FILES\_AND\_DIRECTORIES : lists both files and directories

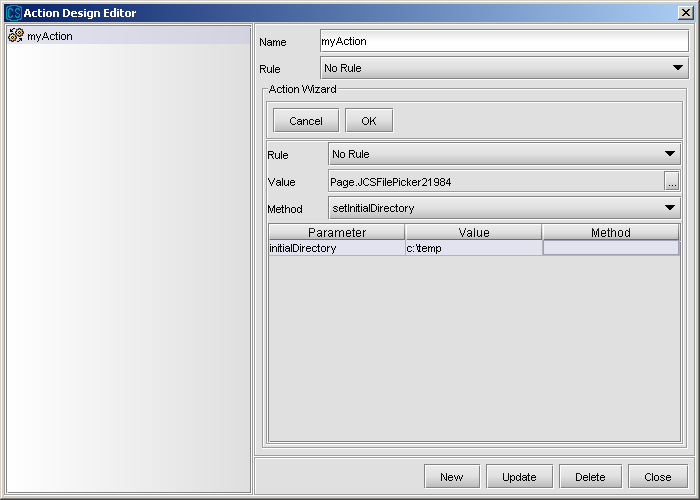


***3.3.1.12.1 FilePicker Operations***

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of FilePicker component.

Details about “Bean Action” are given in ”Action Design” section.

\*FilePicker supports selection of “All Files” option.



**close :** used for closing file

**getContent:** used for getting contents of file

**getFileName :** used for getting name of file

**read:** used for reading contents of file

getMimeType : returns the mime type of the file

getDirectory : used for getting the selected directory

getFileExtension : used for getting the extension of the selected file

getFileNameWithoutExtension : used for getting the name of the selected file without extension

cleanup : used for cleaning file picker field

addFilter(filterType,filterDescription) : used for adding a filter with filterType and filterDescription to the file dialog

removeFilter(filterType) : used for removing the filter whose filter type is filterType

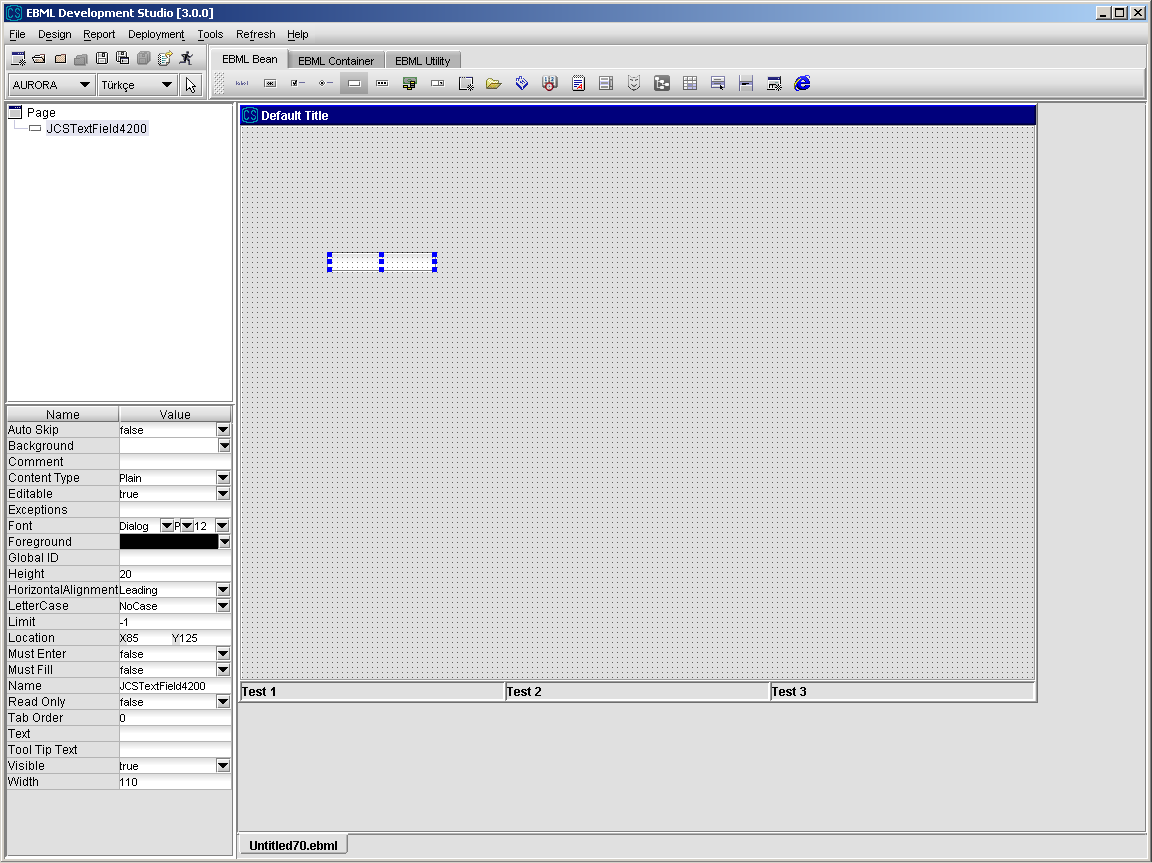
getSelectedFiles : used for getting the selected file(s) in the file dialog

setContent(fileContent) : used for setting the content of the selected file as fileContent, but does not save the file.

save : used for saving the file with the current content

#### 3.3.1.13 Text Field

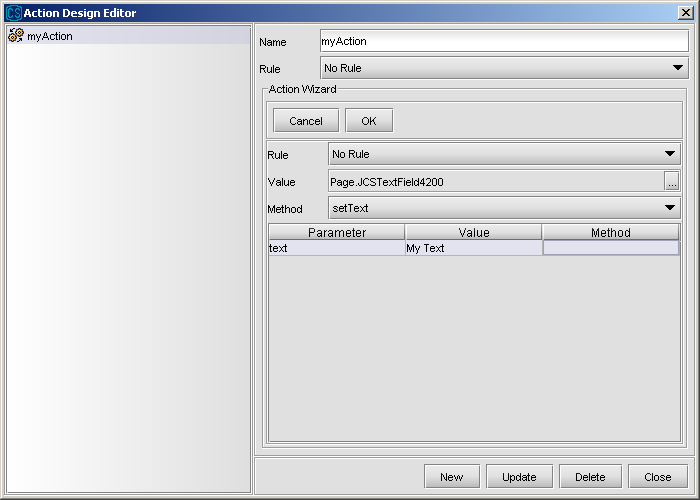
1. Select Text Field component from the toolbar and drop into page.
2. Set the properties of Text Field component.



##### 3.3.1.13.1 Text Field Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of TextField component.

Details about “Bean Action” are given in ”Action Design” section.



**appendText(text) :** appends specified text to the “text” property of JCSTextField

**getTextLength :** returns the length of text in the text field

**deleteLast :** used for deleting the last character of text field

#### 3.3.1.14 Handle Button Field

Handle Button Field is a special text field where you can find the data that you intend to write into textfield by making queries on a popup field which appears when the small button on the handle button field is clicked.

##### 3.3.1.14.1 Designing New Popup (Popup Template)

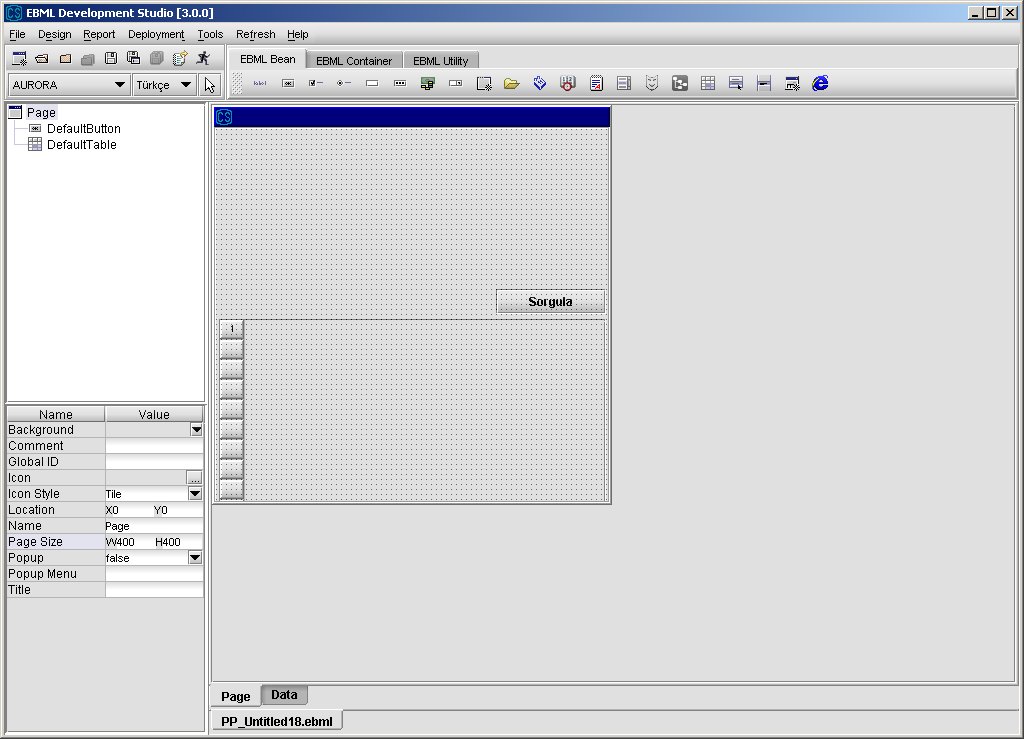
Design of a new popup has two basic steps:

* Popup Screen Design
* Popup Query Design

##### 

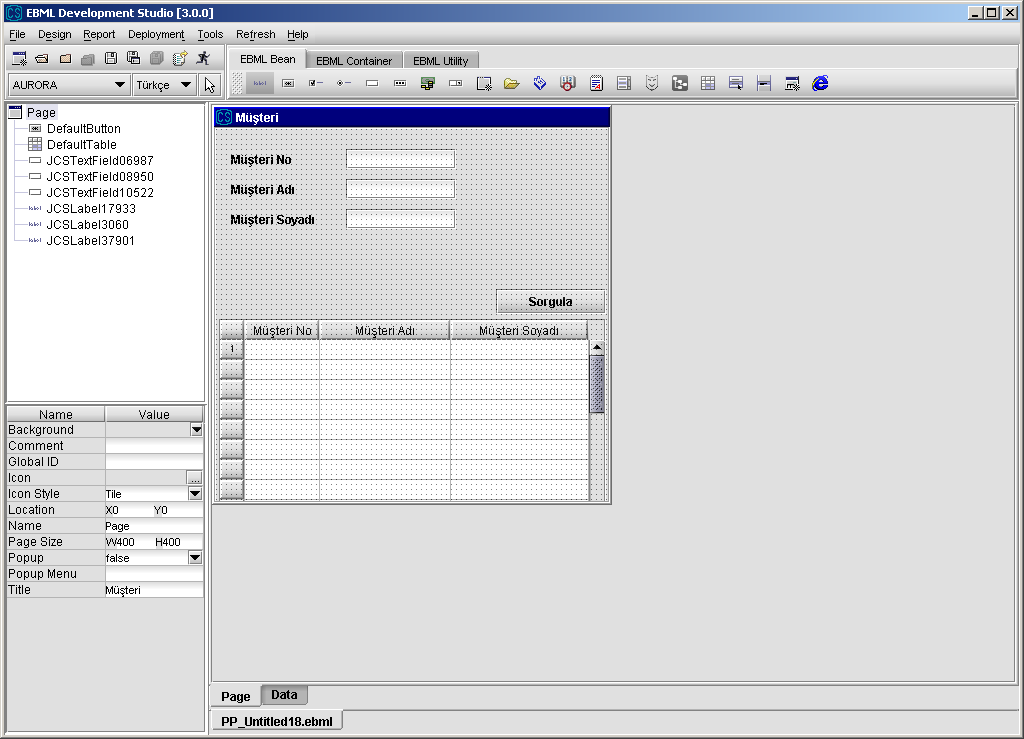
##### 3.3.1.14.2 Popup Screen Design

1. Select Popup under File\New menu item.
2. After selecting this menu item, a new popup design screen is displayed. All popups contain a search button (SORGULA) and a table by default.



1. Set the popup properties.
2. Design the popup by selecting the components from the toolbar and dropping into page. While designing a popup, developer can use any type of the component (label, text field, buttons, panel,...) from the toolbar. A region or another popup can also be placed in a popup.
3. The columns that will be displayed on the default table should be determined by using table customizer which will be shown by right-clicking on table and selecting *Customizer*.
4. Deploy the popup to the server.

The deployment process is the same as the deployment of any EBML file. The developer will be warned if there is another popup with the same name that it will be overridden. The developer must be very careful about naming and deploying a popup in order not to loose an already designed and operational popup.



##### 

##### 3.3.1.14.3 Popup Query Design

1. There are two ways to create the content of a popup. One is to prepare content by an SQL file, the other is to prepare content by a service.

To prepare by a SQL query, write down a valid SQL statement for the designed popup.

SELECT MBR\_ID, NAME, SURNAME

FROM MEMBERS

WHERE MBR\_ID LIKE ‘%’ and

NAME LIKE ‘%’ and

SURNAME LIKE ‘%’

STATUS = '1'

ORDER BY MBR\_ID, NAME

To prepare by an AURORA service, create a service which will put popup content in “CORE\_DATA” bagkey in the returned bag parameter, like :

public static CSBag testPopup(CSBag inBag) throws CSException {

CSBag returnBag = new CSBag();

….

….

….

returnBag.put(“CORE\_DATA”,0,”MBR\_ID”,”ID\_1”);

returnBag.put(“CORE\_DATA”,0,”NAME”,”NAME\_1”);

returnBag.put(“CORE\_DATA”,0,”SURNAME”,”SURNAME\_1”);

….

….

….

return returnBag;

}

1. Prepare an XML file that defines this content preparation operation.

In the case of a SQL query, this file contains this SQL with the proper search criteria

embedded into it. The following XML file is developed from the query above:

<popupdata type="sql">

<sql dataSource="AuroraDS">

SELECT MBR\_ID, NAME, SURNAME

FROM MEMBERS

WHERE MBR\_ID LIKE ? and

NAME LIKE ? and

SURNAME LIKE ?

STATUS = '1'

ORDER BY MBR\_ID, NAME

</sql>

<parameters>

<parameter prefix="" suffix="%" type="string">Page.txtMbrId</parameter>

<parameter prefix="" suffix="%" type="string">Page.txtMbrName</parameter>

<parameter prefix="" suffix="%" type="string">Page.txtMbrSurname</parameter>

</parameters>

</popupdata>

In a popup sql file, the column names of the result set of the executed query must match the column identifiers of the table in the popup.

As can be realized from the example, each search criterion must be embedded into the statement as “?” and the statements that will replace them are defined like below:

<parameter prefix=”” suffix=”%” type="string">Page.txtMbrId</parameter>

“prefix” and “suffix” attributes define prefix and suffix values that will be added to the specifed parameter during the execution of the query. “type” attribute defines whether the parameter will be enclosed with single quota or not. It can take three values; if it is stated as “string”, the parameter is enclosed with single quota, if it is stated as “numeric” or blank (or attribute “type” is not stated), the parameter isn’t enclosed with single quota.

In the example above, if the value of “Page.txtMbrID” is “A”; the value executed in the query will be “'A%'”. If the type attribute stated as “numeric” or left blank, the value executed int query will be “A%”.

The content of parameter tag must match the id of a component placed on the popup window.

If the popup content will be created by an AURORA service, the XML will be like :

<popupdata type="service">

<service>TEST\_POPUP\_SERVICE</service>

<parameters>

<parameter n="CORE\_ID">Page.txtMbrId</parameter>

<parameter n="CORE\_NAME">Page.txtMbrName</parameter>

<parameter n="CORE\_SURNAME">Page.txtMbrSurname</parameter>

</parameters>

</popupdata>

As can be realized, parameters are again defined under the tag “parameters” with a syntax :

<parameter n=”BagKey name”>value</parameter>

These parameters are passed into the input bag of the specified AURORA service. So in the case of the XML defination above, the specified AURORA service can get these parameters as;

public static CSBag testPopup(CSBag inBag) throws CSException {

String idParameter = inBag.get(“CORE\_ID”).toString();

String nameParameter = inBag.get(“CORE\_NAME”).toString();

String surnameParameter = inBag.get(“CORE\_SURNAME”).toString();

….

….

}

As stated before, popup content must be presented in the “CORE\_DATA” bag key of the return bag parameter of the AURORA service. Bagkeys names must match with the column identifiers of the popup table.

1. Then, this XML file must be named with the same name as the popup’s EBML file

except it has no “PP\_ “ prefix and it must have the “.sql” extension. For instance, if the name of the popup EBML is PP\_CUSTOMER then SQL XML file name must be CUSTOMER.sql. After naming the file as described, the file must be deployed to the .server by using Deployment\Deploy Popup SQL menu item.

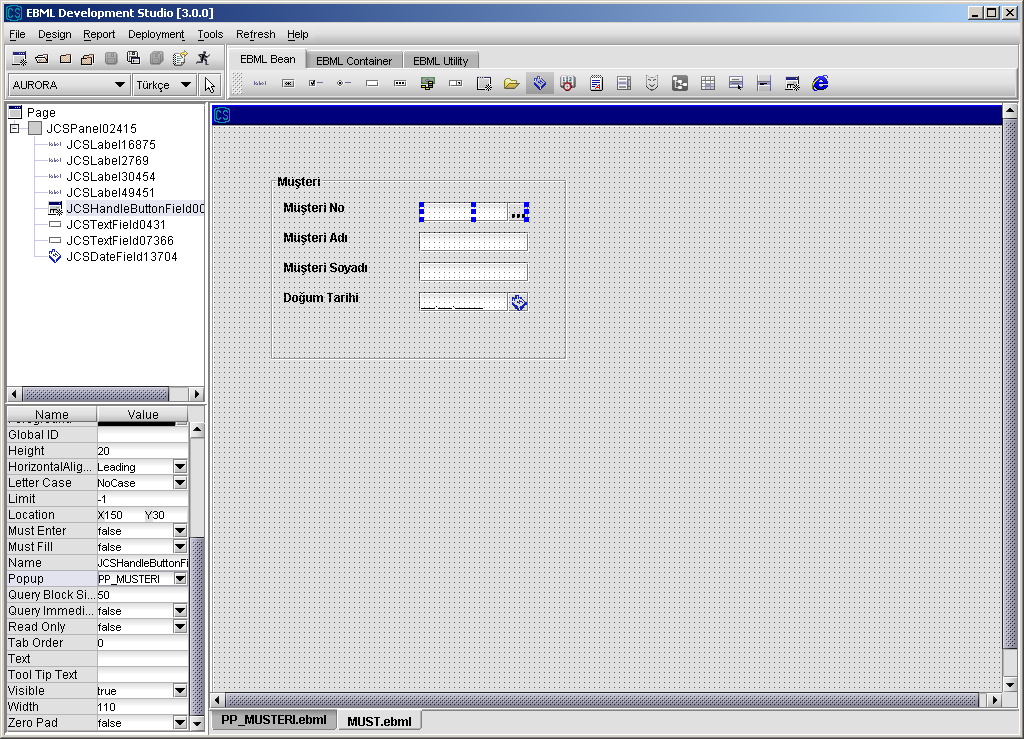
The developer must be very careful about the compatibility of the source field and the target field when defining outputs and inputs. For instance, if one tries to send value which contains non-numeric characters to a numeric field as input or output, only the numeric characters will be assigned and non-numeric ones will be ignored.

##### 

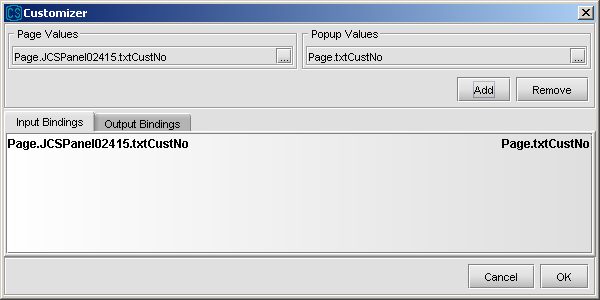
##### 3.3.1.14.4 Using Previously Designed Popup

Developer can use the previously defined popups while designing a screen.

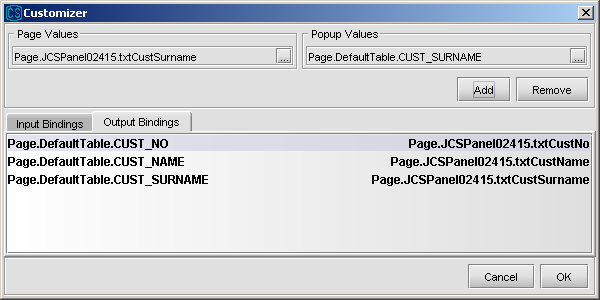
1. Select the popup component and drag into screen
2. Select a popup name from Popup Name property.



1. By right-clicking on the handle button field, select Customizer. In the customizer make the input/output bindings of the popup.



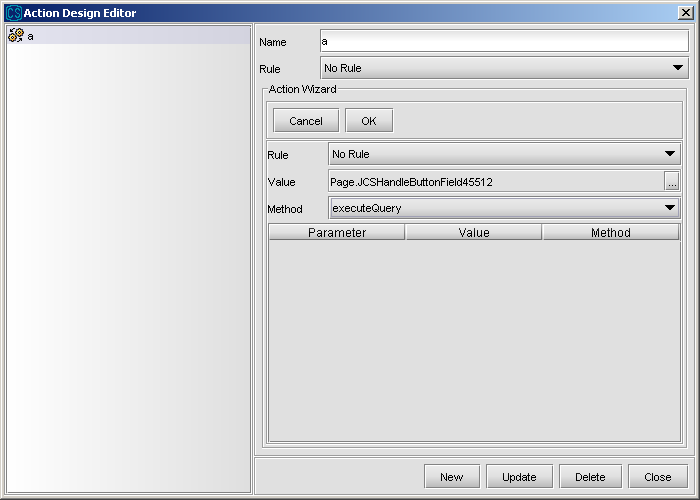
In the Page Values popup, only the beans on the page is seen. In the Popup Values part only the beans on the popup is seen. In input binding, you determined which values go from page to popup beans. In output binding, which columns of the selected row from the default table of popup will go to which beans of the page.



#### 3.3.1.14.4 Popup Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of HandleButtonField component.

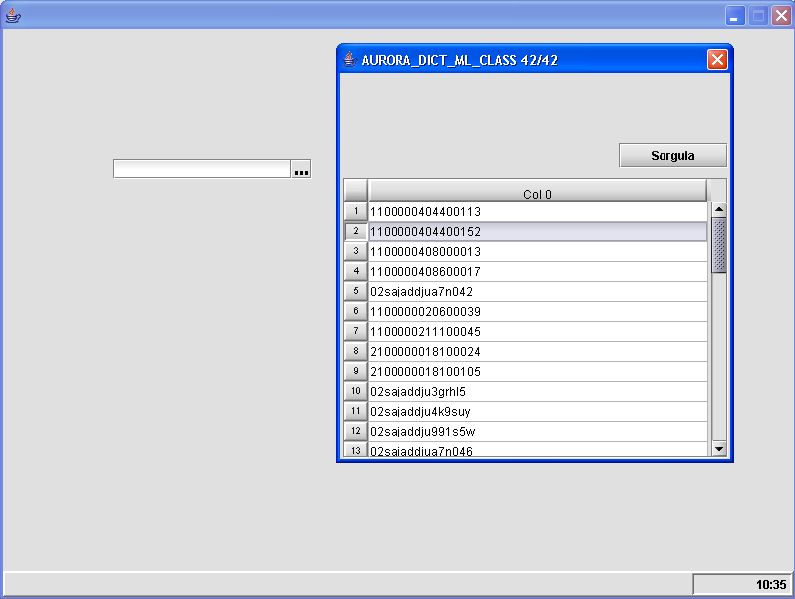
Details about “Bean Action” are given in ”Action Design” section.



renderPopup(popupName) : used for rendering a popup whose name popupName dynamically

executeQuery : used for executing popup query without showing the popup. If query returns one result, it behaves as if the result is selected from popup. If more than one or no result comes, it shows the popup.

Note : When user hits “Enter” key, popup dialog is calls executeQuery method automatically and dialog is shown. If SQL result is only one record, there is no need to show diloag and bindings is processed for one record. An open popup dialog is shown below.



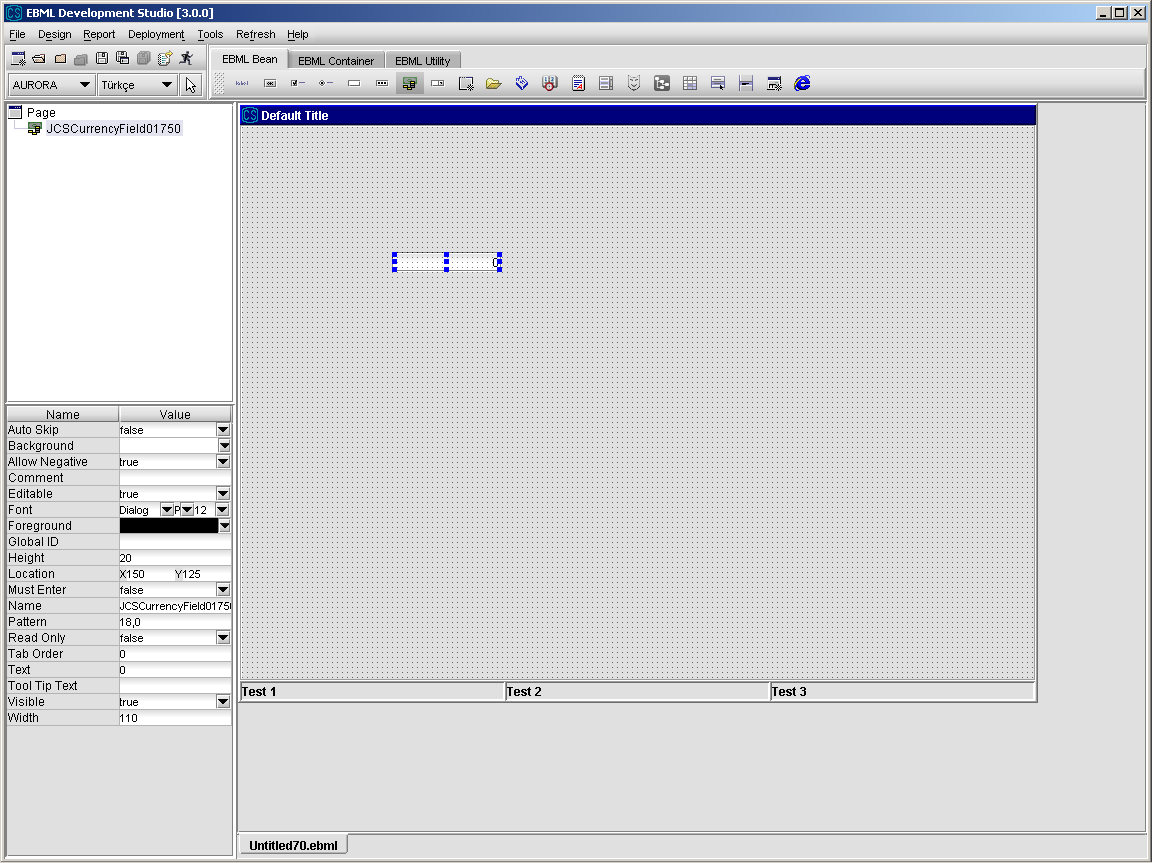
In this sample, popup table column “COL\_0” is binded to handle button on page. When a row double clicked or “Enter” key is pressed on any key, popup window disposes and bindgings are executed. As seen below, handle button field is filled with the value of selected row record.



#### 3.3.1.15 Currency Field

CurrencyField is used for representing currency.

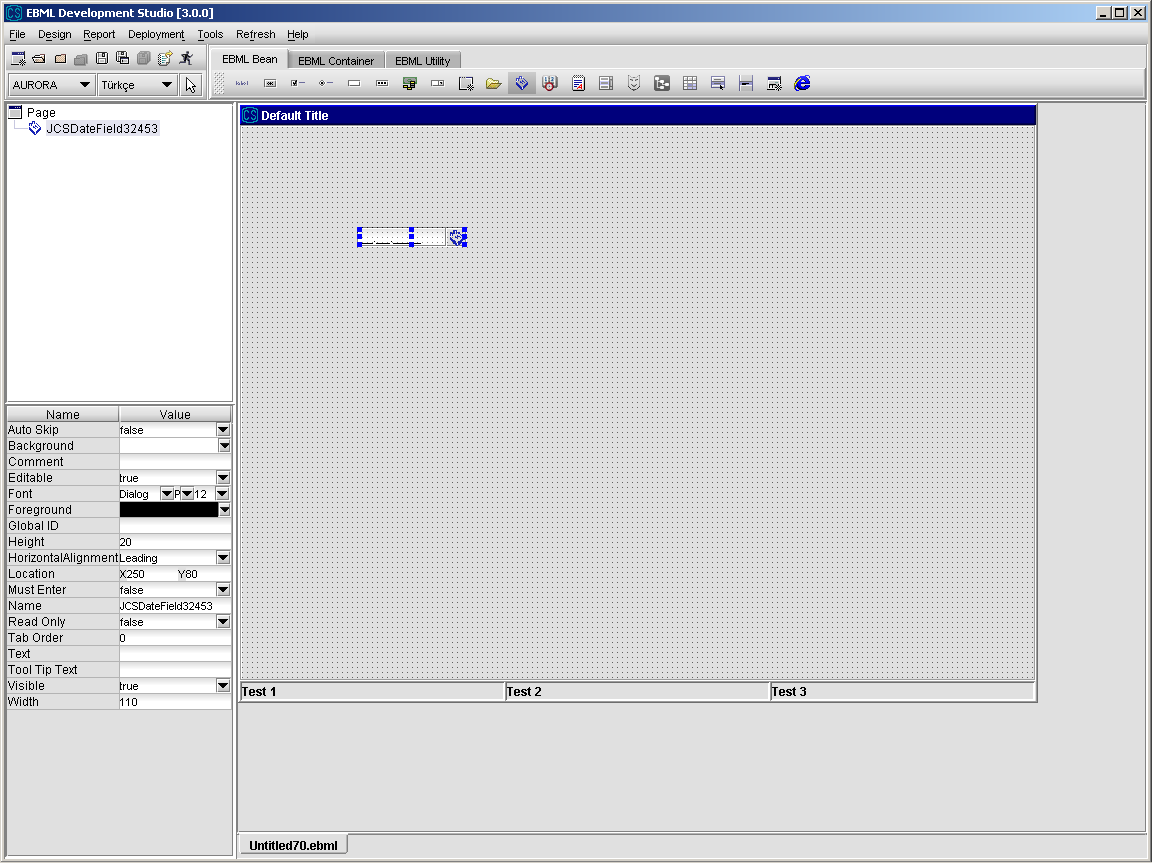
1. Select Currency Field component from the toolbar and drop into page.
2. Set the properties of Currency Field component.



#### 3.3.1.16 Date Field

DateField componenet is used for representing date.

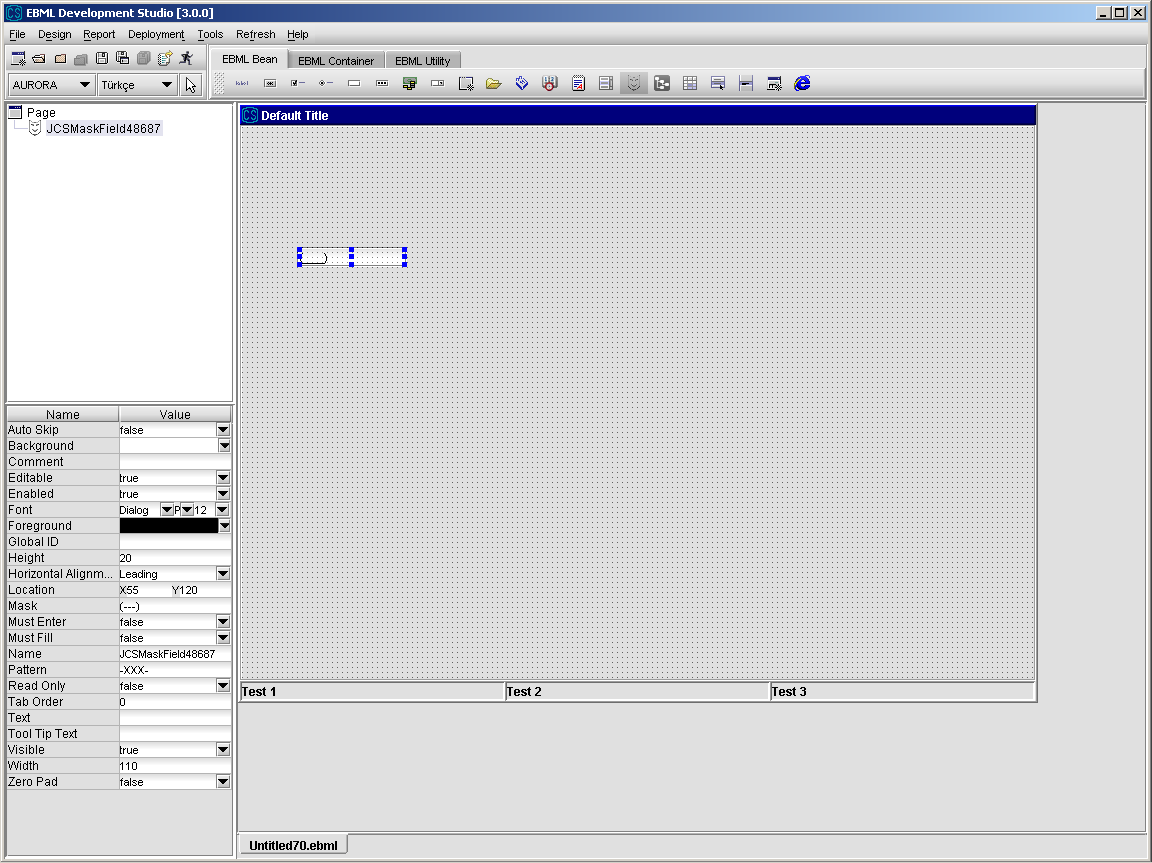
1. Select Date Field component from the toolbar and drop into page.
2. Set the properties of Date Field component.



#### 3.3.1.17 Mask Field

MaskField component is used for getting string values corresponding to the specified pattern.

1. Select Mask Field component from the toolbar and drop into page.
2. Set the properties of Mask Field component.



#### 3.3.1.17.1 Using Mask And Pattern Properties

Pattern property determines content type of text index. Pattern values must be one of the belows.

# : Digit

A : Letter

X : Alpha-numeric character

U : Upper-case character

L : Lower –case character

E : One of “exceptions” character

\* : Any character

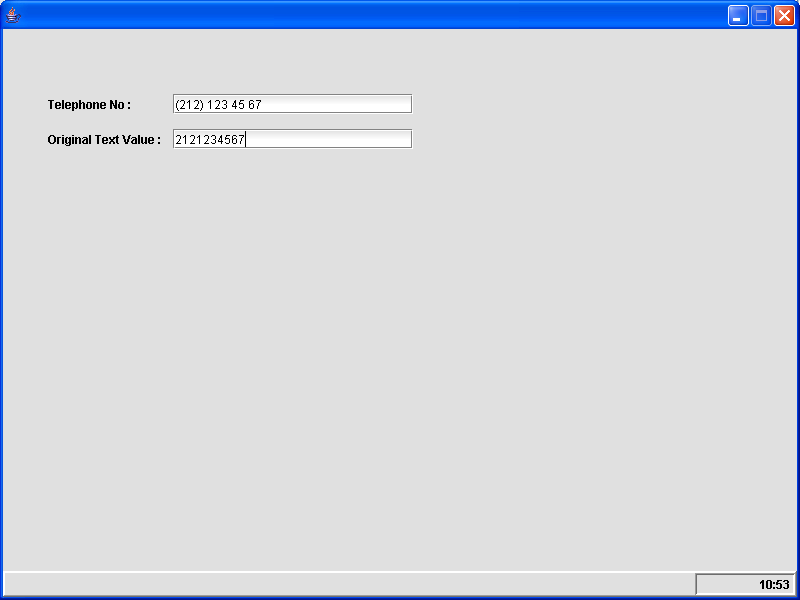
“-” means use the value defined in mask field

Example: If you one to make an telefon number mask field

Pattern -###--###-##-##

Mask (---) --- -- --

Next figure shows this example.



Other Examples;

Pattern : ###-###

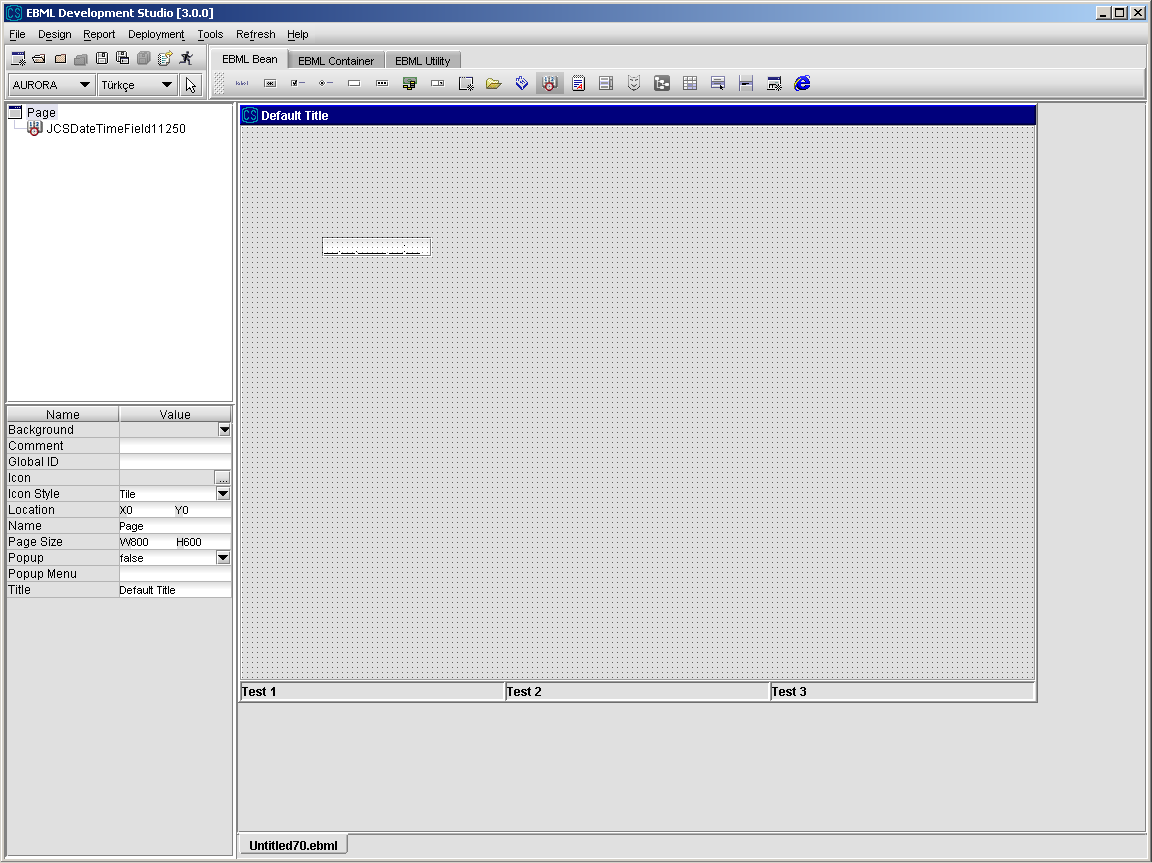
Mask : ---/---

Result : 123/456

#### 3.3.1.18 DateTime Field

DateTimeField component is used for representing date and time together.

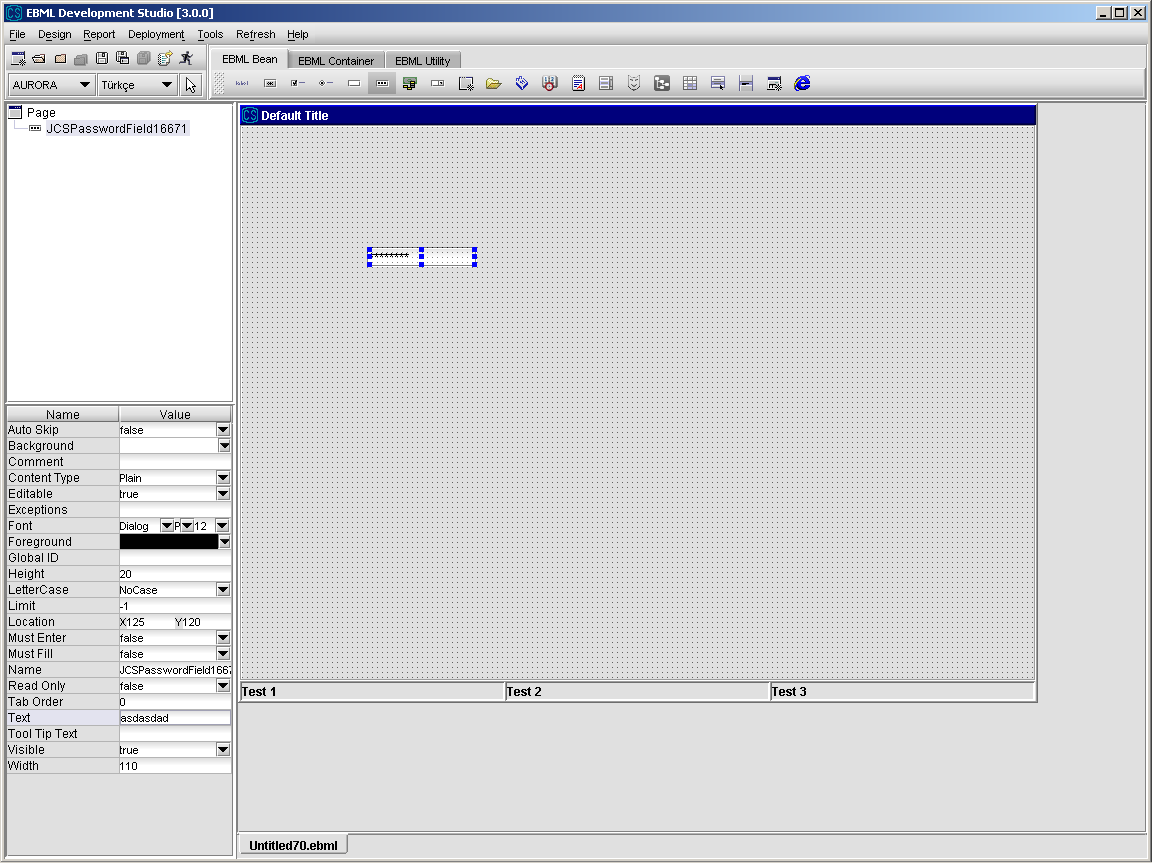
1. Select DateTimeField component from the toolbar and drop into page.
2. Set the properties of DateTimeField component.



#### 3.3.1.19 Password Field

PasswordField component is used for representing passwords.

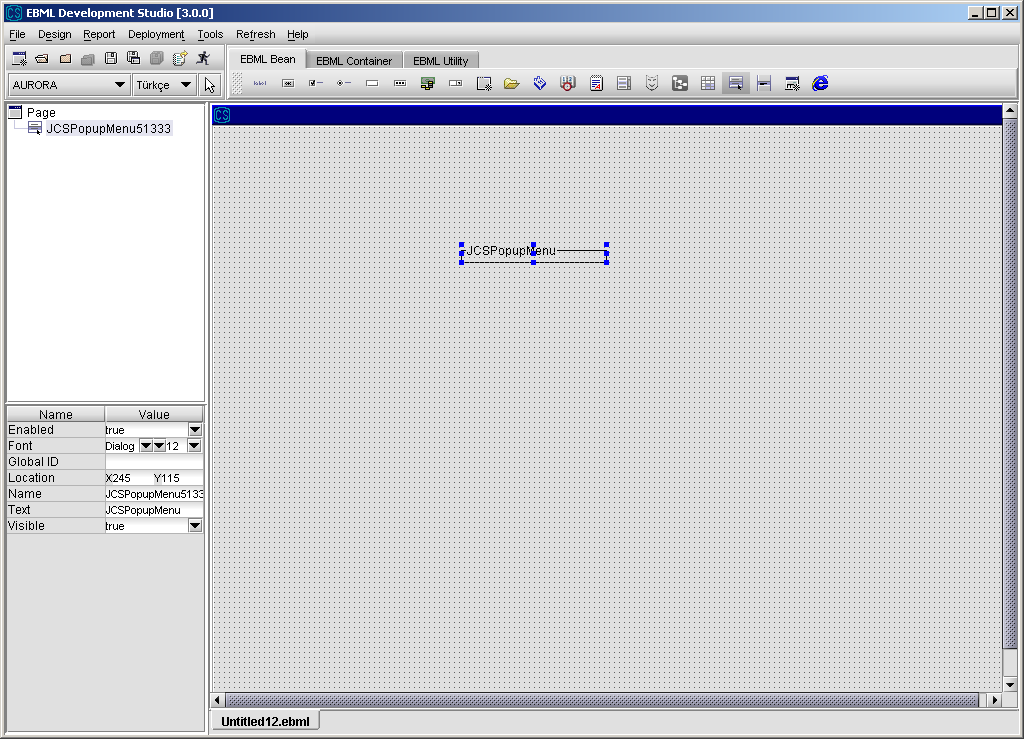
1. Select Password Field component from the toolbar and drop into page.
2. Set the properties of Password Field component.



#### 3.3.1.20 Popup Menu

Popup menu is used to show a popup menu when right mouse button is clicked on some components like JCSPage, JCSTree and JCSTable.

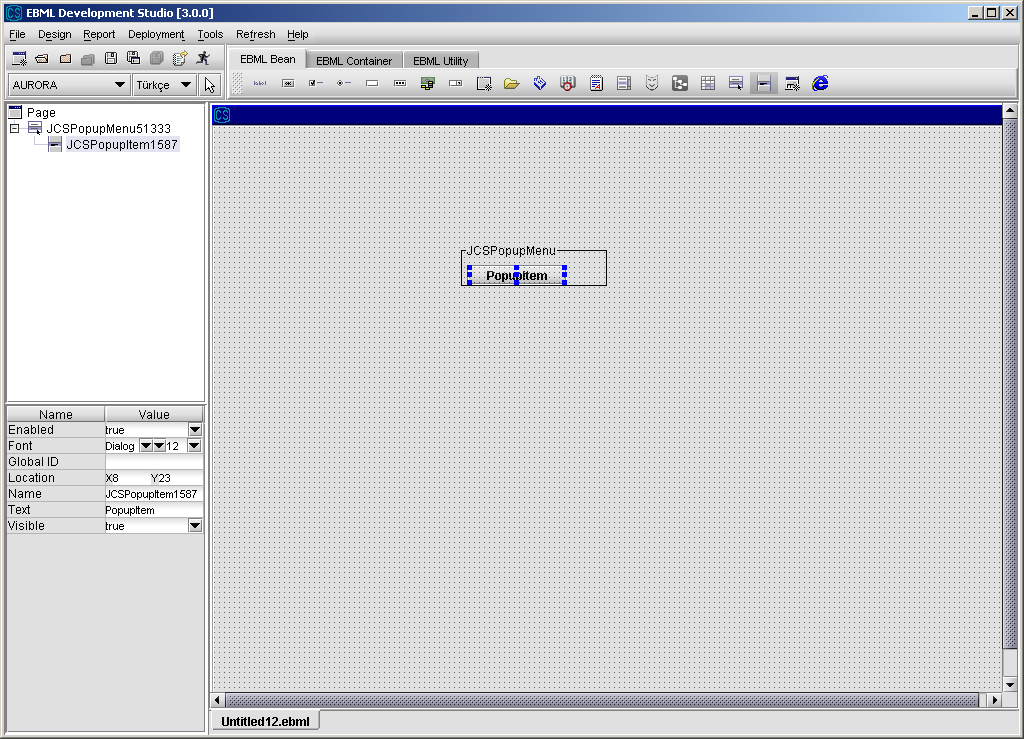
1. Select Popup Menu component from the toolbar and drop into page.
2. Set the properties of Popup Menu component.



#### 3.3.1.21 Popup Menu Item

Popup menu item is an element that can be clicked on a popup menu.

1. Select Popup Menu Item component from the toolbar and drop it into a popup menu component.
2. Set the properties of Popup Menu Item component.

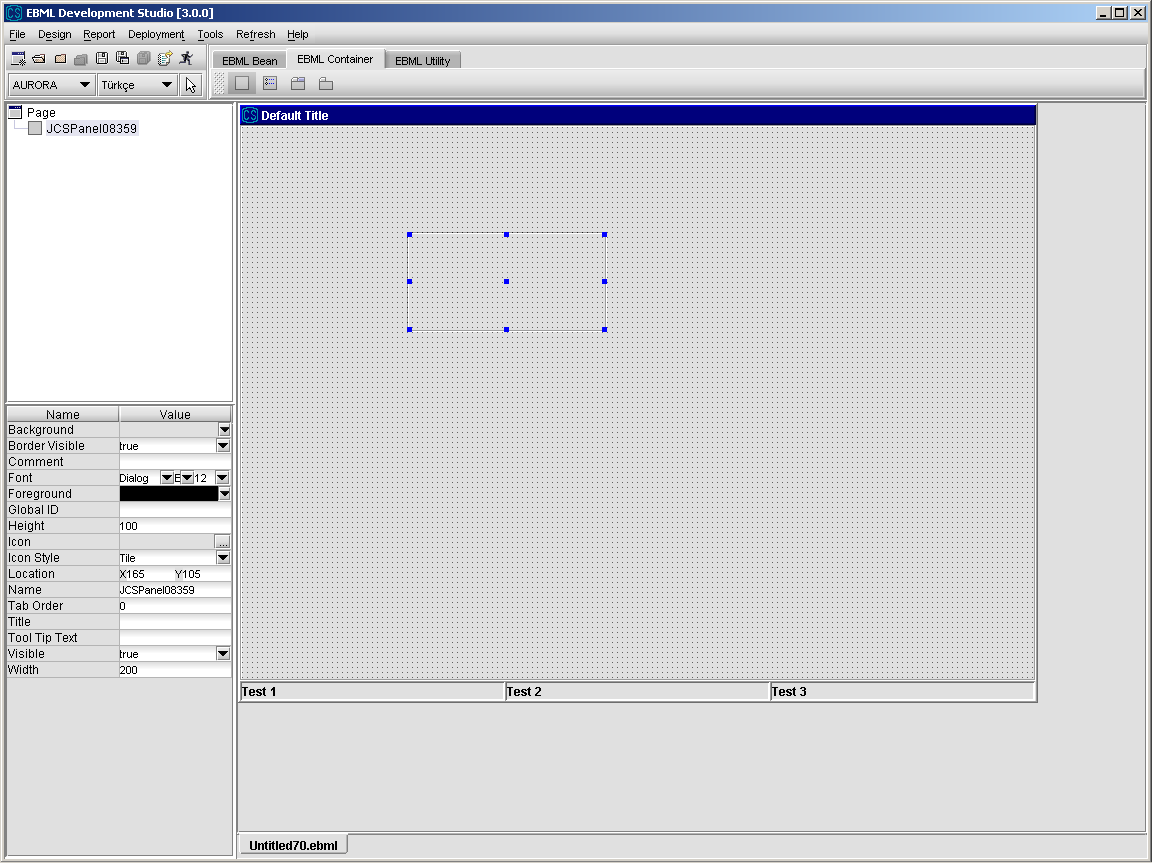


### 3.3.2 JCS CONTAINERS

#### 3.3.2.1 Panel

Panel is used for grouping the components .

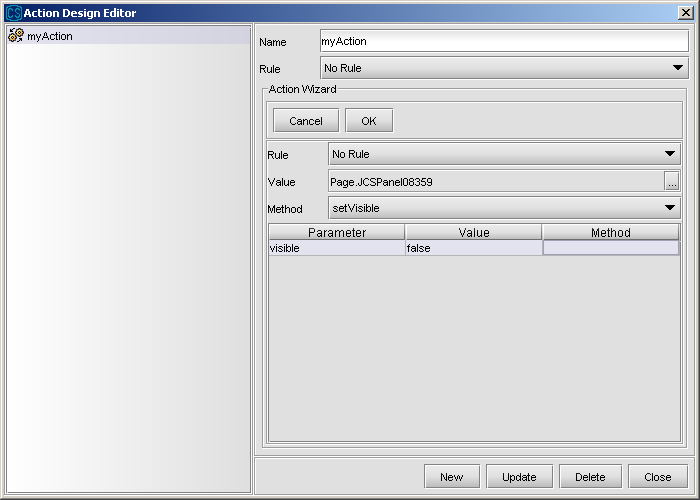
1. Select Panel component from the toolbar and drop into page.
2. Set the properties of Panel component.

****

##### 3.3.2.1.1 Panel Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Panel component.

Details about “Bean Action” are given in ”Action Design” section.



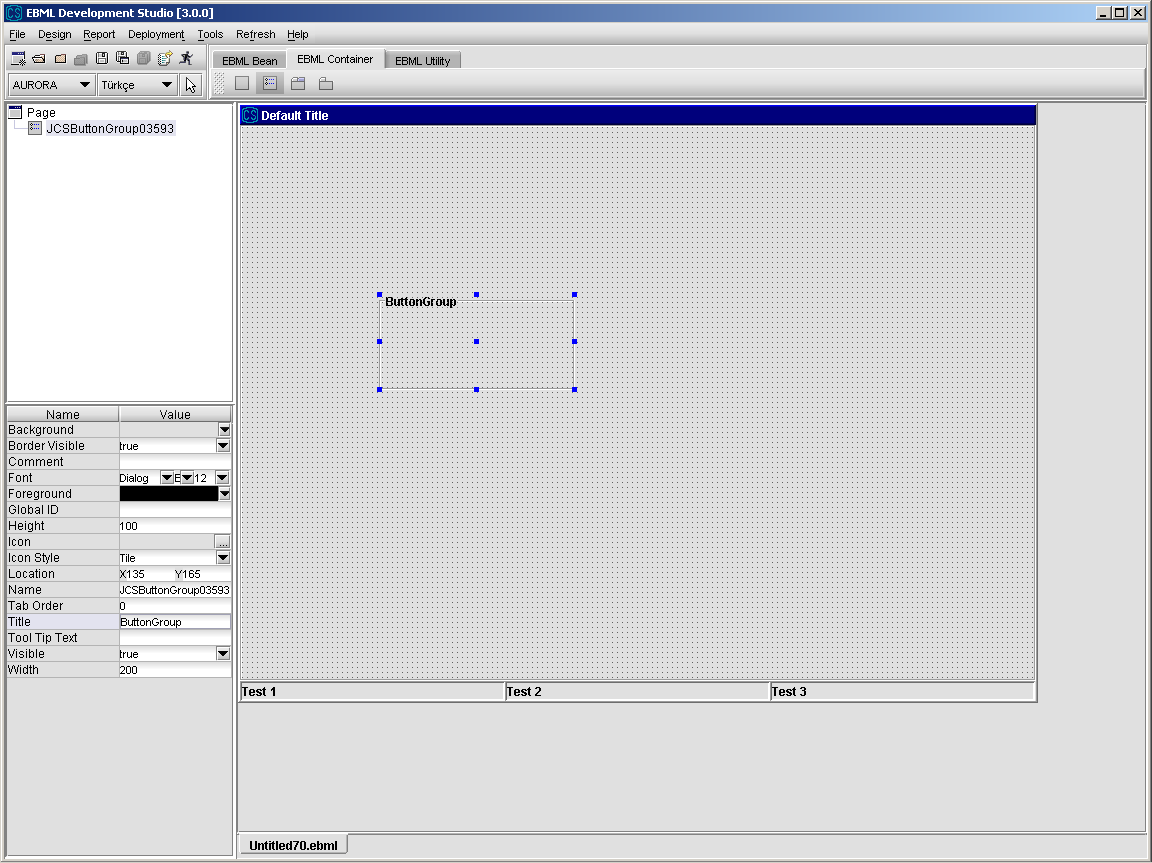
**cleanUp :** calls cleanup method for all the beans on the panel

**setEnableAll(enabled) :** used for setting ”enabled” property of all the components on the panel as *enabled* which is either true or false

#### 3.3.2.2 Button Group

Button group is used for grouping radio buttons.

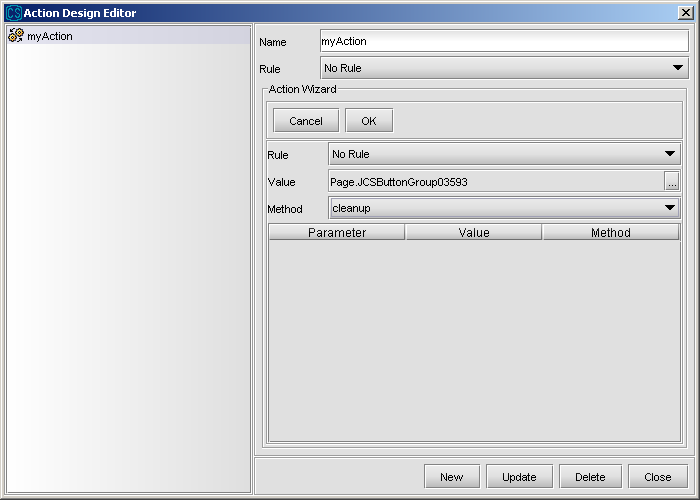
1. Select Button Group component from the toolbar and drop into page.
2. Set the properties of Button Group component.



##### 3.3.2.2.1 Button Group Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of ButtonGroup component.

Details about “Bean Action” are given in ”Action Design” section.



**setSelected(buttonName):** used for selecting radio button whose name is *buttonName* in button group

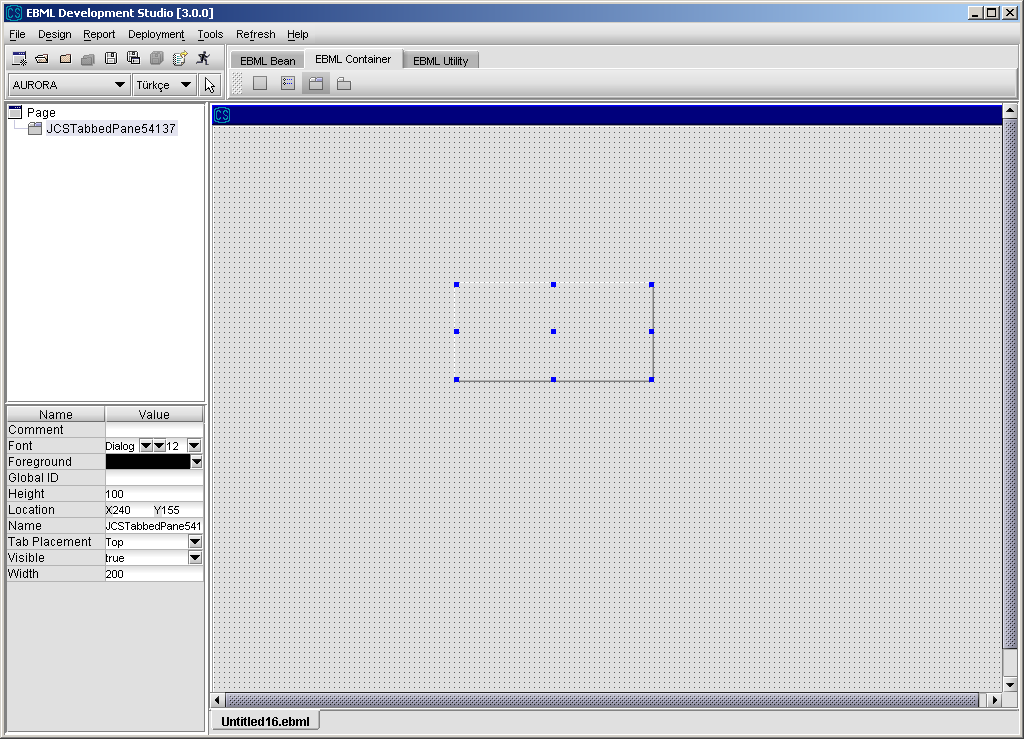
**cleanup:** deselects all the radion buttons in the button group

**setEnableAll(enabled) :** used for setting ”enabled” property of all the components on the panel as *enabled* which is either true or false

#### 3.3.2.3 Tabbed Pane

Tabbed Pane is used for placing Tab Pages.

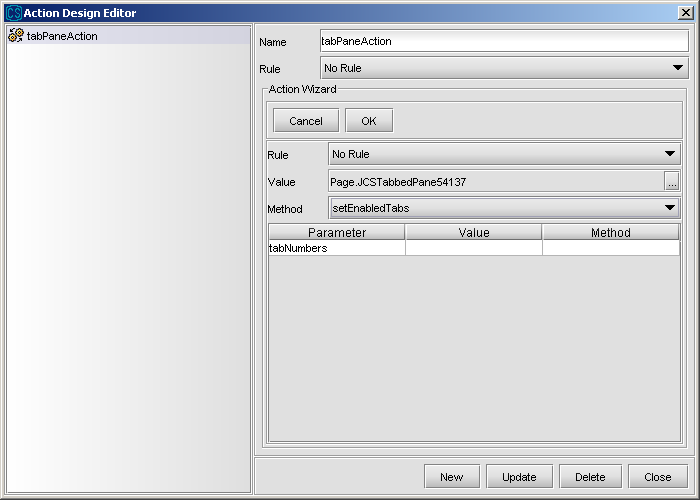
1. Select Tabbed Pane component from the toolbar and drop into page.
2. Set the properties of Tabbed Pane component.



##### 3.3.2.3.1 Tabbed Pane Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of TabbedPane component.

Details about “Bean Action” are given in ”Action Design” section.



getSelectedTab: used for getting the index of the selected tab

setDisabledTabs(tabNumbers): used for setting the disabled tabs. The indexes are given as input and they should be separated by commas

setEnabledTabs(tabNumbers): used for setting the disabled tabs. The indexes are given as input and they should be separated by commas

setSelectedTab(tabIndex): used for selecting the tab page whose index is tabIndex

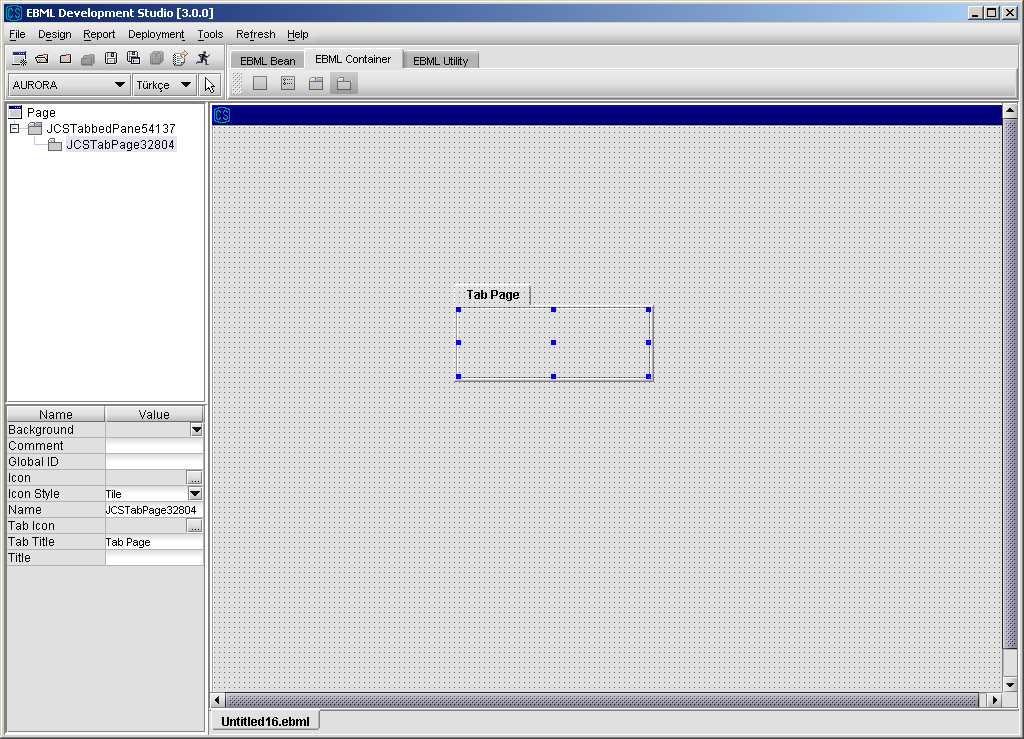
cleanup: calls the cleanup method for all tab pages in the tabbed pane

**setEnableAll(enabled) :** used for setting ”enabled” property of all the tab pages on the tabbed pane as *enabled* which is either true or false. Also, tab pages enables/diables all child components.

#### 3.3.2.4 Tab Page

Tab Page is used as a page in a Tabbed Pane.

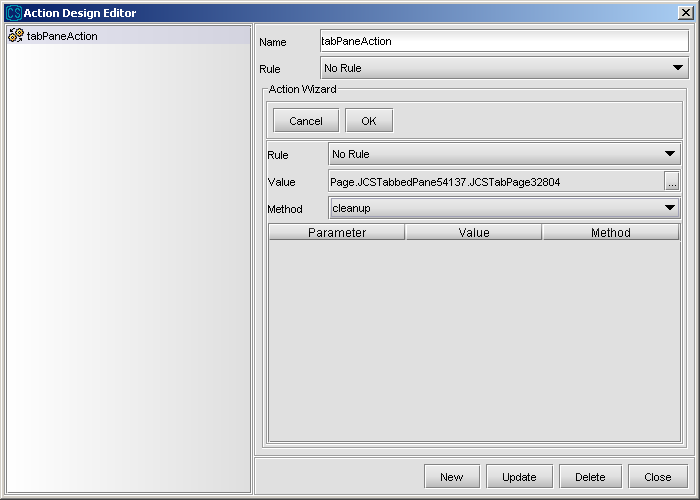
1. Select Tab Page component from the toolbar and drop into a TabPane component in a page. Tab Page component can not be placed anywhere else.
2. Set the properties of Tab Page component.



##### 3.3.2.4.1 Tab Page Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of TabPage component.

Details about “Bean Action” are given in ”Action Design” section.



cleanup: calls cleanup method for all the components in the tab page

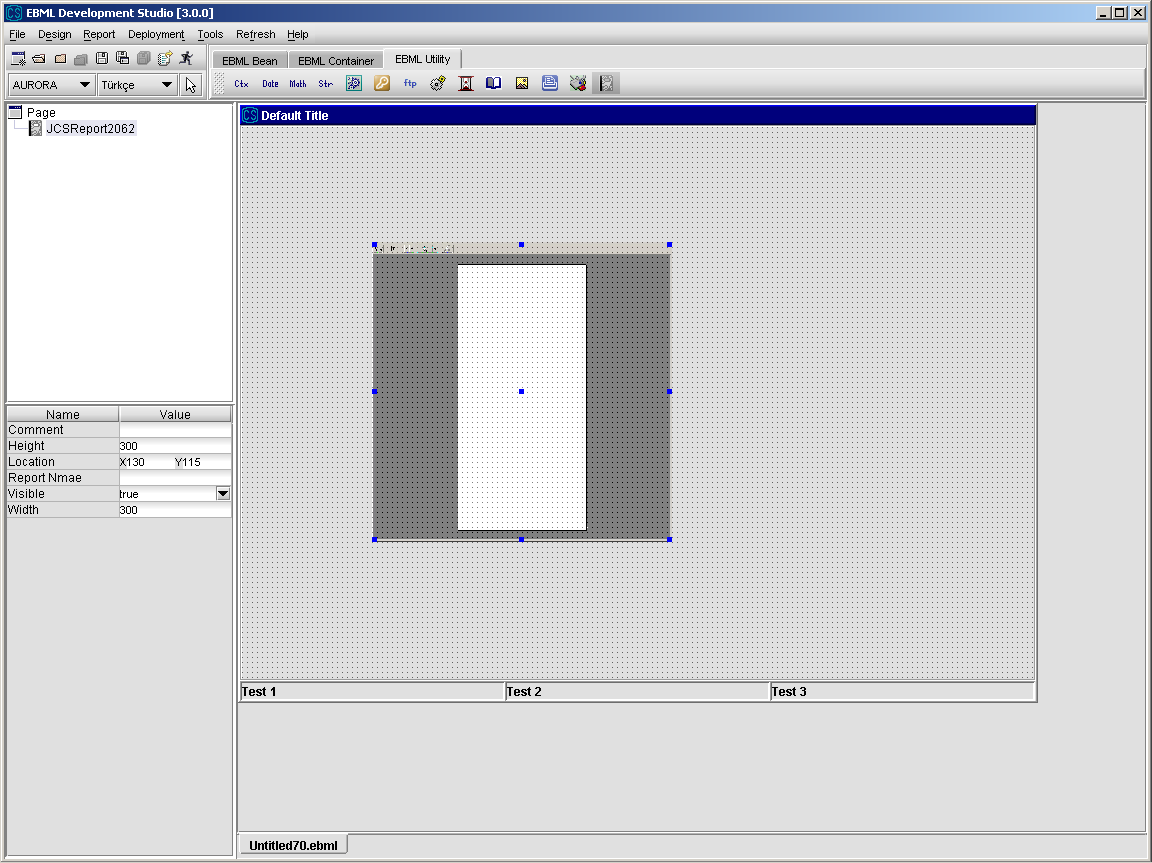
**setEnableAll(enabled) :** used for setting ”enabled” property of all the components on the tab page as *enabled* which is either true or false.

### 3.3.3 JCS UTILITY BEANS

#### 3.3.3.1 JCSReport

JCSReport component is used to display, print and save reports based on a given report template. Reports can be saved in different formats such as “pdf”, “html”, “txt”. Before printing and saving a report, it should be displayed on screen.

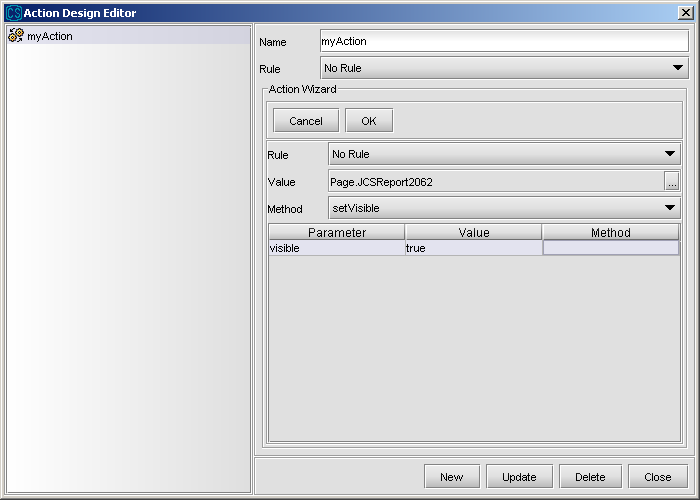
1. Select Report component from the toolbar and drop into page.
2. Set the properties of the Report component.



##### 3.3.3.1.1 Report Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Report component.

Details about “Bean Action” are given in ”Action Design” section.



**print :** prints the report

**printToFile :** prints the report to a file

**render :** renders the report on the screen

#### 3.3.3.2 JCSTextPrinter

JCSTextPrinter Component is used to print text data

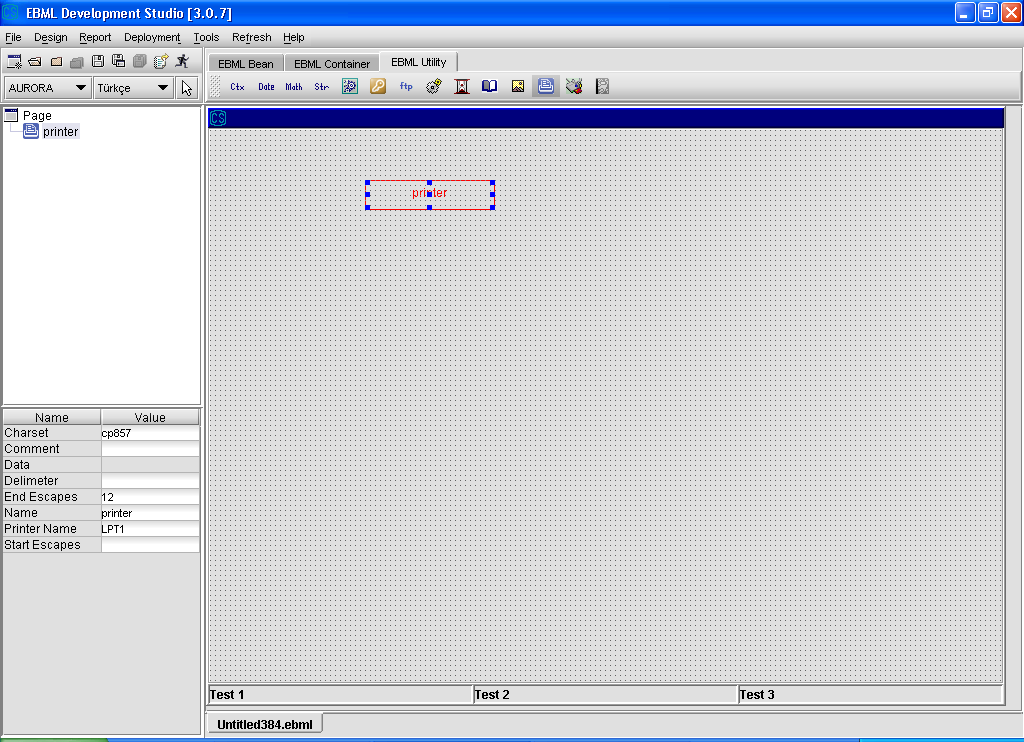
1. Select JCSTextPrinter component from the toolbar and drop into page.
2. Set “Printer Name” to the name of your printer. It is a hardware port, parallel or serial port.For instance, LPT1,COM1,etc.
3. Set “Start Escapes” and “End Escapes” to your printer’s escape characters if it has special escape characters. Some escape codes are
   1. 12 - Form Feed
   2. 27 69 - Set emphasized mode for bold printing
   3. 27 70 - Reset emphasized mode.
   4. Etc.

Note that escape codes must be added by semicolon.

Start Escape : 27,69,12

End Escape : 12

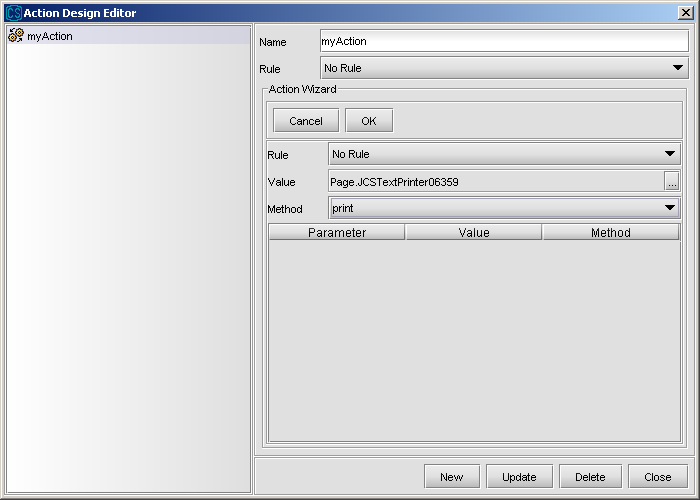
1. Call “setData” method for what you want to print.
2. Call “print” method to print data.



##### 3.3.3.2.1 TextPrinter Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of TextPrinter component.

Details about “Bean Action” are given in ”Action Design” section.

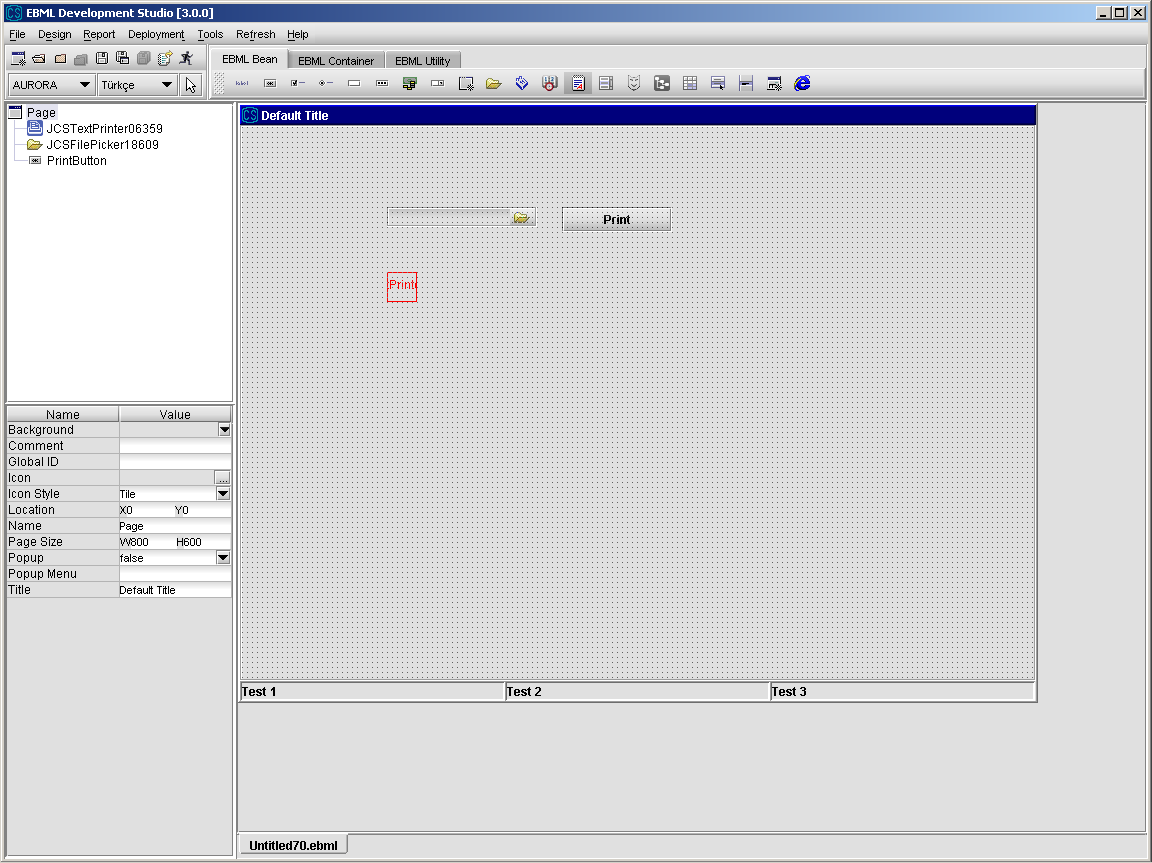
****

**setPrinter(printerName):** used for setting the printer name (e.g LTP1)

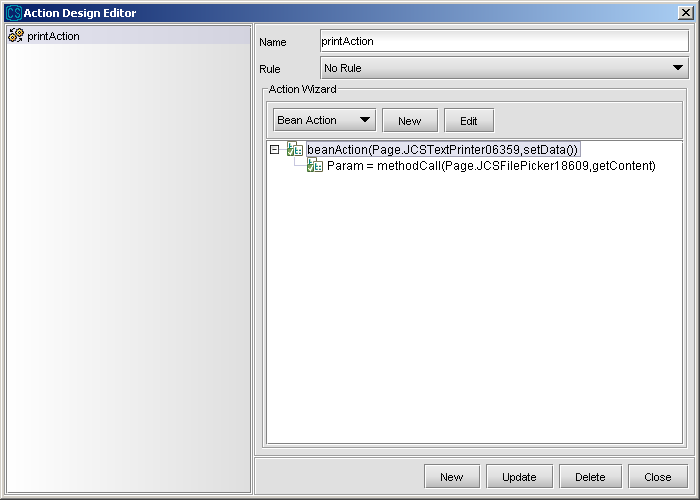
**setData :** Data for printing

**print :** used for printing data which is set by “setData” method.

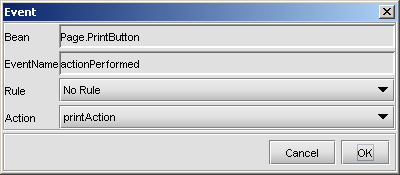
**Example:** Assume that we would like to design a screen enabling the user to print the contents of any file selected. Page should have a file picker and text printer component.



From Design\Action Design Editor, define a new action, assume it is “printAction”m add the required bean action to set the file content as the data of the text printer.



Close the action editor and right click on the Print button. Select actionPerformed event of the Button and relate this event with “printAction” which we defined above.



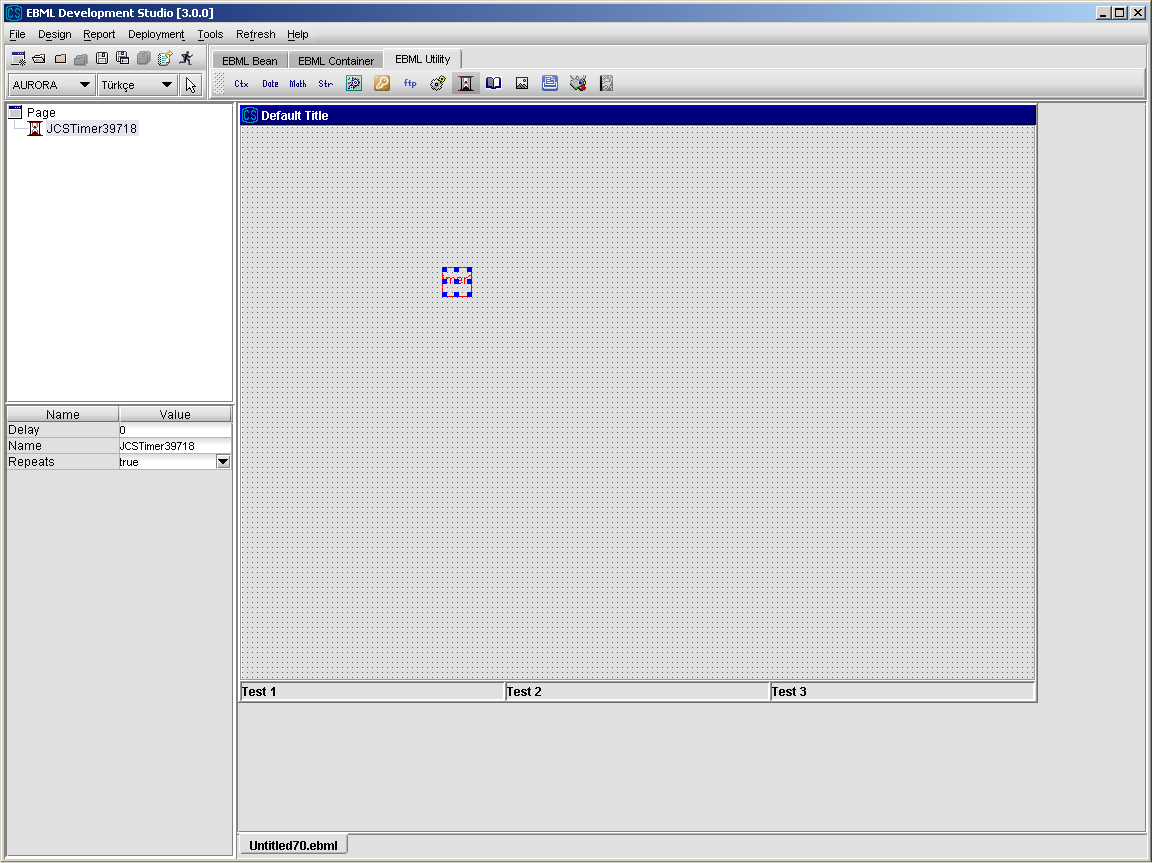
Now, when you test the screen with CMS, you firstly select a text file to print. When you click the Print button, the content of the file will be printed.

Note : It is not recommended to change Charset property except you you want to use a special code-page.

#### 3.3.3.3 JCSTimer

JCSTimer is used for trigger events for some time intervals from the client side.

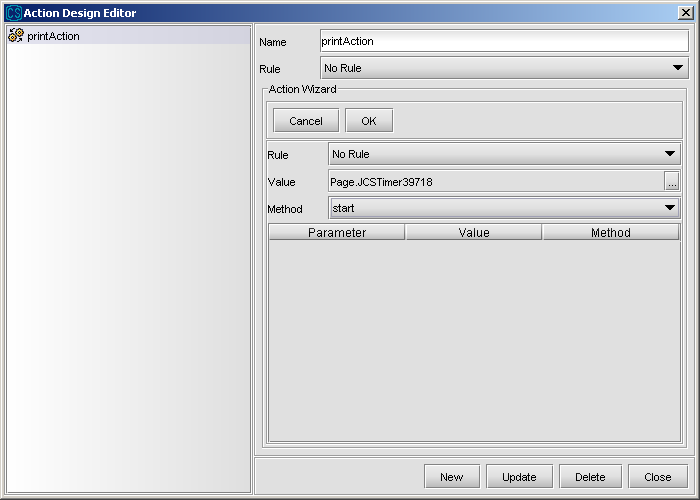
1. Select Timer component from the toolbar and drop into page.
2. Set the properties of Timer component.



##### 3.3.3.3.1 Timer Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Timer component.

Details about “Bean Action” are given in ”Action Design” section.

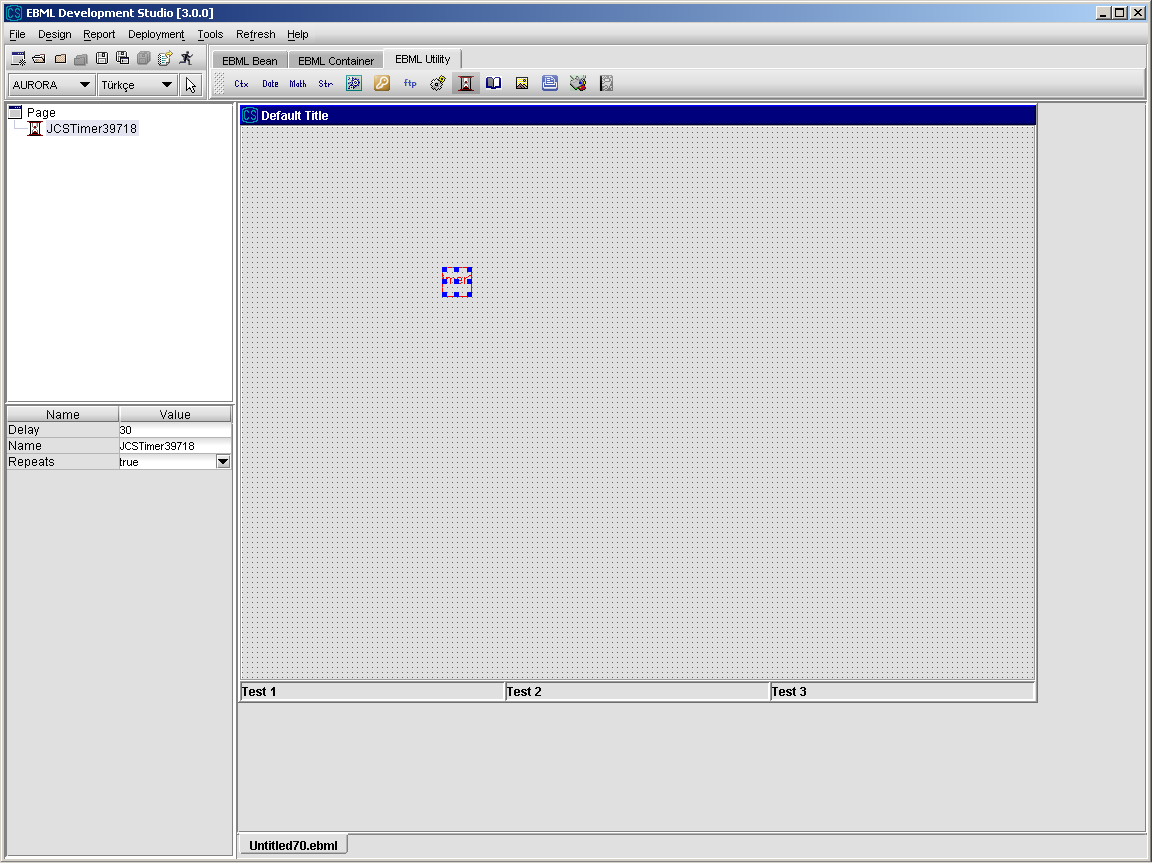


**start** **:** starts the timer

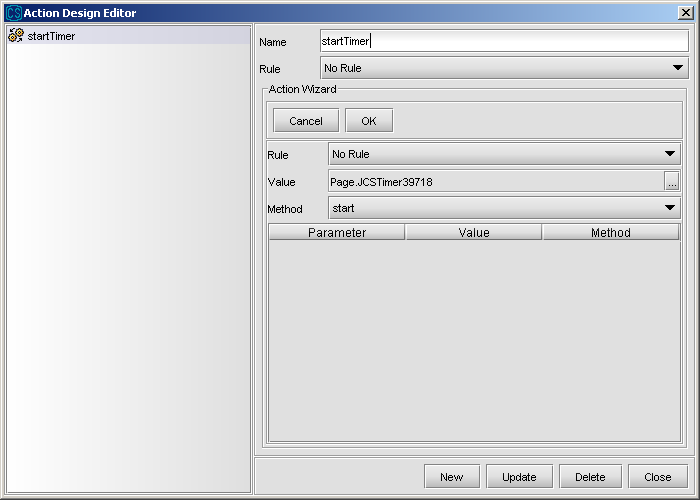
**stop :** stops the timer

**Example :** Assume that we would like to design a screen enables to call a service at every 30 seconds. The page should have a JCSTimer component.

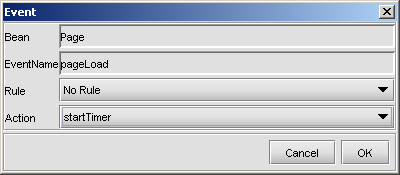
First we define a JCSTimer bean with “delay” property set to “30” and repeats property set to “true” (so it does not stop at the first time, it continues working).



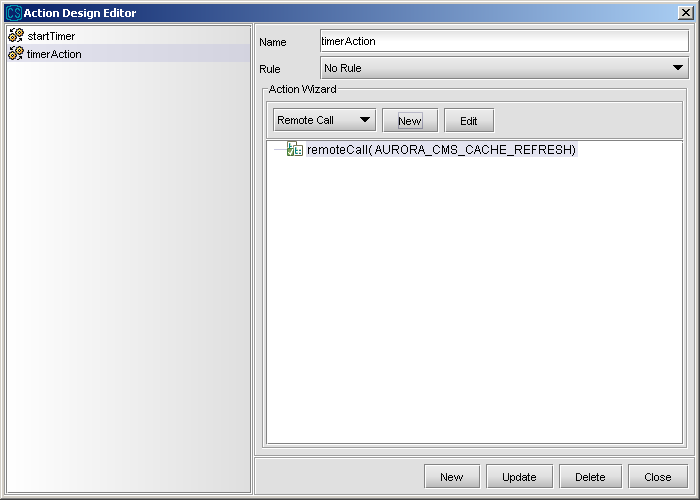
Then we define a new action from Design\Action Design Editor to start the timer. Call this action “startTimer”.



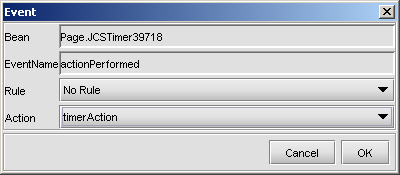
Relate this action with the pageLoad event of the Page component.



Now, define the action that will be repeated every 30 seconds. Call it as “timerAction”.



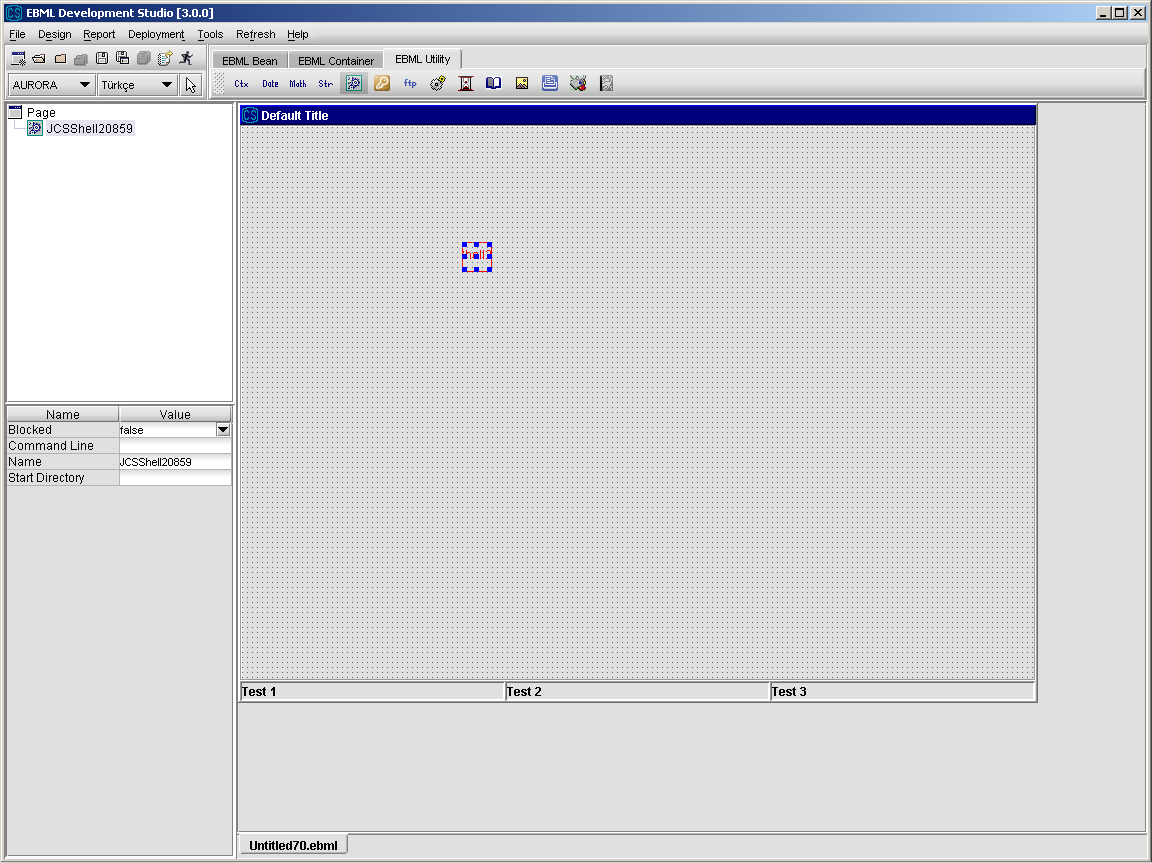
Relate this action with the actionPerformed event of Timer component.



#### 3.3.3.4 JCSShell

JCSShell is used for executing shell commands.

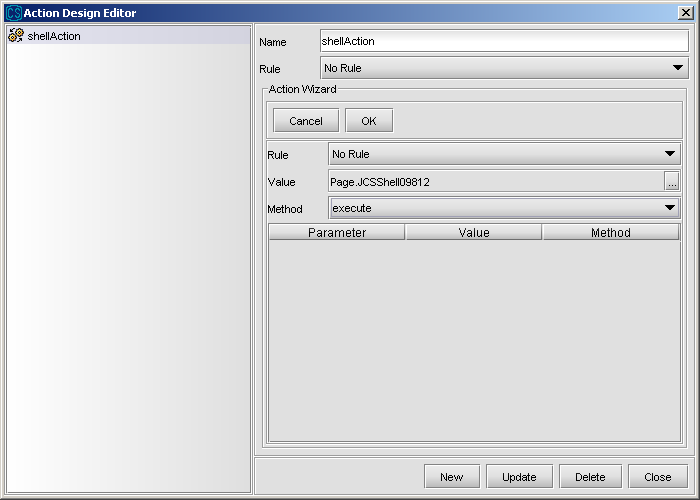
1. Select Shell component from the toolbar and drop into page.
2. Set the properties of Shell component.



##### 3.3.3.4.1 Shell Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Shell component.

Details about “Bean Action” are given in ”Action Design” section.



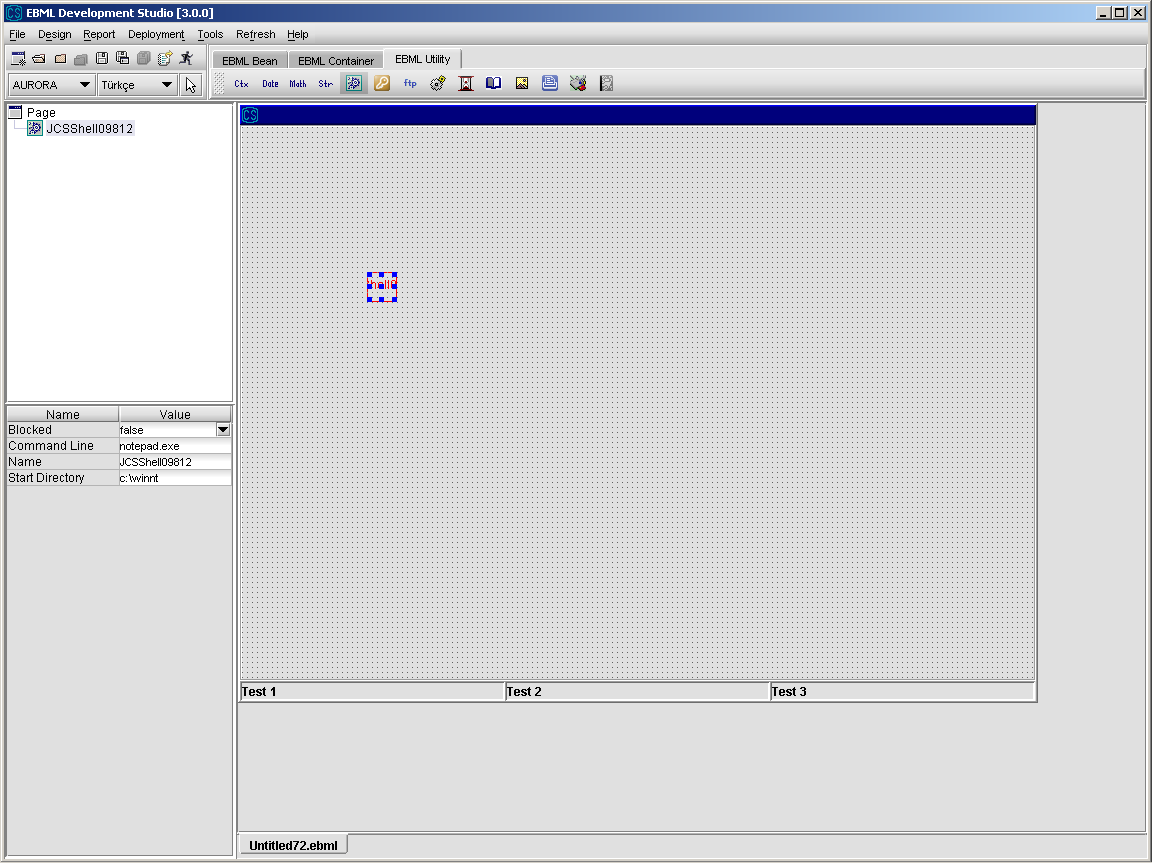
**execute:** Executes the command specified in the “command” property of the shell bean (if “start directory” is specified, the shell bean tries to execute the command from that directory)

**executeWithParameters(parameters):** used for executing commands that requires parameters

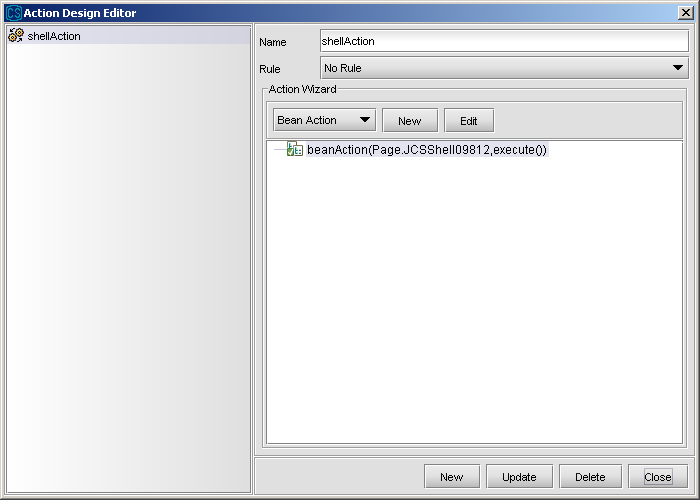
**getErrorValue:** used for getting the error value as a result of the executing a commmand

**Example:** Assume that we would like to design a screen that can execute Windows NotePad on pageLoad. The page should have a JCSShell component.

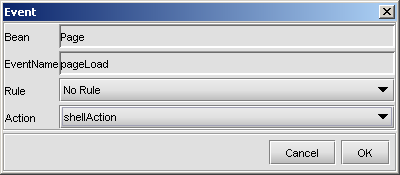
First we define a JCSShell bean with “command” property set to “CNOTEPAD.EXE” and “start directory” property set to “C:\WINNT”.



Then we define an action to call the execute method of JCSShell component.



Finally, relate the pageLoad event of the Page component with the action “shellAction”.

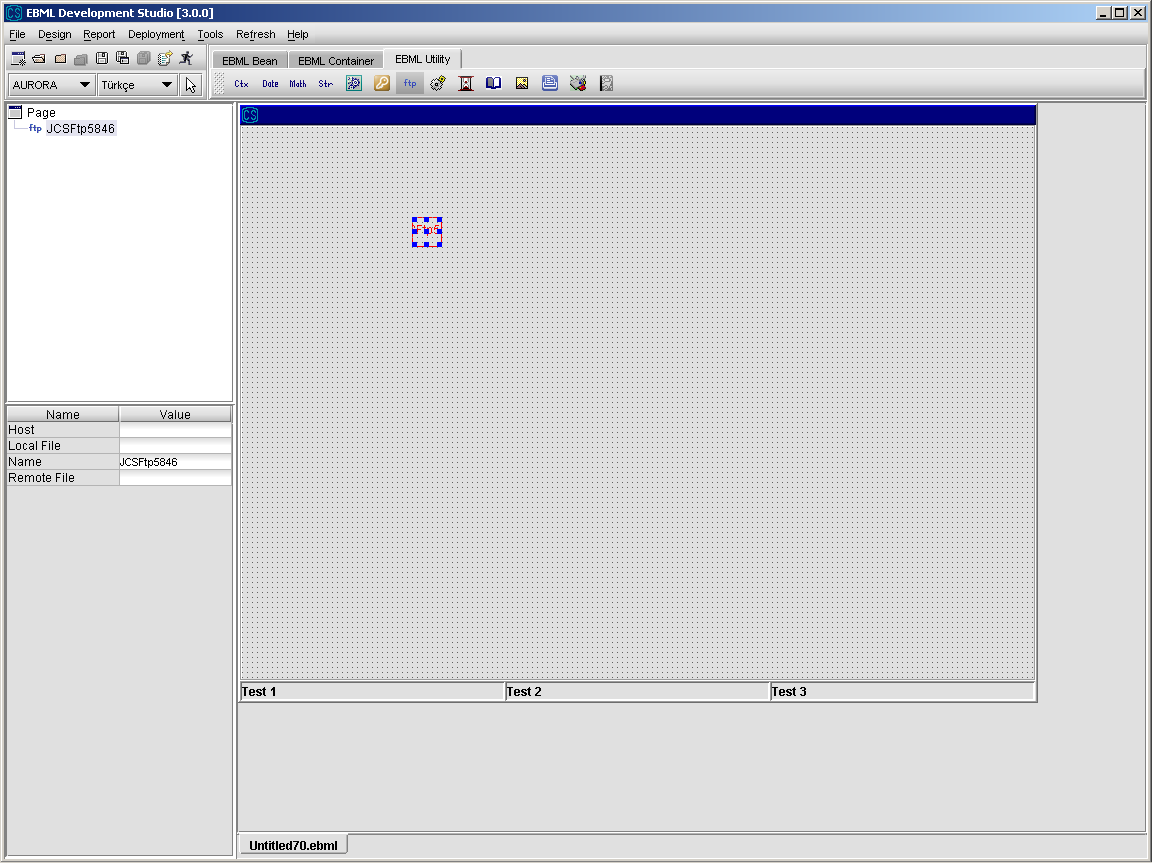


When you test the screen, you see that a Notepad program runs, when your screen is opened.

#### 3.3.3.5 JCSFtp

JCSFtp is used for doing FTP operations from the client side.

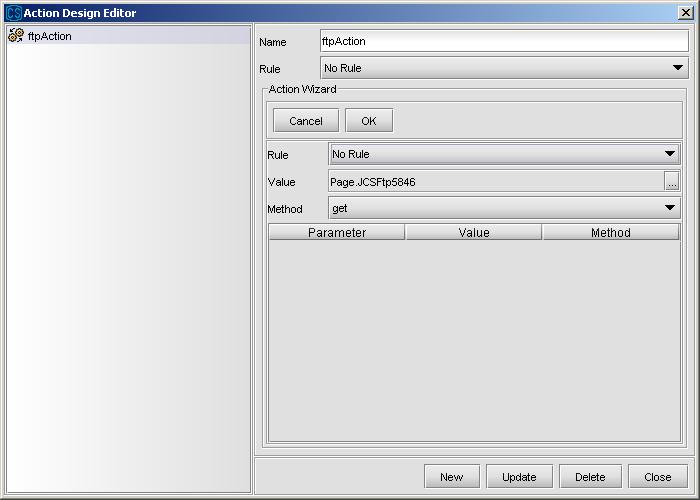
1. Select Ftp component from the toolbar and drop into page.
2. Set the properties of Ftp component.



##### 3.3.3.5.1 Ftp Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Ftp component.

Details about “Bean Action” are given in ”Action Design” section.



**get :** getting the file specified

**getPassword :** getting FTP password

**getUsername :** getting FTP user name

**put :** putting the file specified

**setPassword(password) :** setting FTP password

**setUsername(userName) :** setting FTP user name

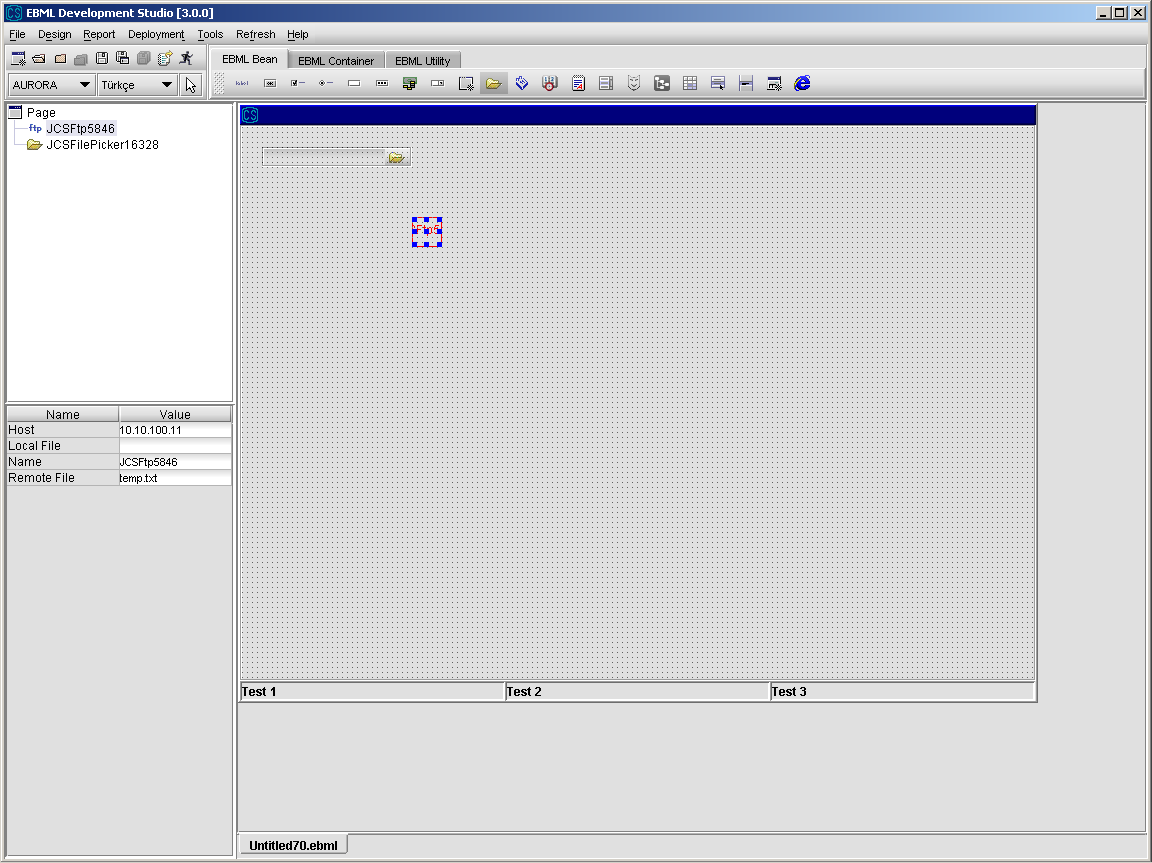
**makeDirs(directories) :** used for creating multiple directories, which are given as a list in directories, on server

**putMultipleFiles(fileArray):** used for putting multiple files given in *fileArray* to ftp server

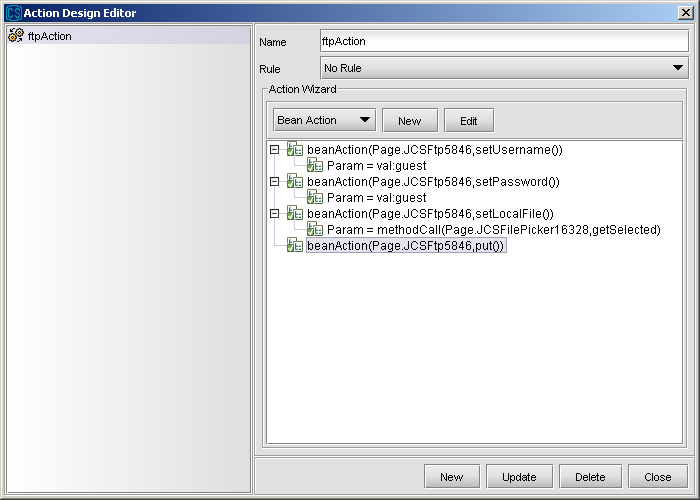
**getMultipleFiles(fileArray):** used for getting multiple files given in *fileArray* from ftp server

**Example**: Assume that we would like to design a screen that can send files selected from a file picker to a remote host as “temp.txt” via ftp. The page should have a JCSFTP component and a JCSFilePicker component.

First we define a JCSFTP bean with “host” property set to “10.10.100.11” and “remote file” property set to “temp.txt”.



We first define an action to put the selected file from FilePicker to the ftp server. In this action we set the both username and password to connect as “guest”. Then, we select the local file that will be uploaded to the server. It will be taken from the FilePicker. Then we call the “put” command of JCSFtp component.



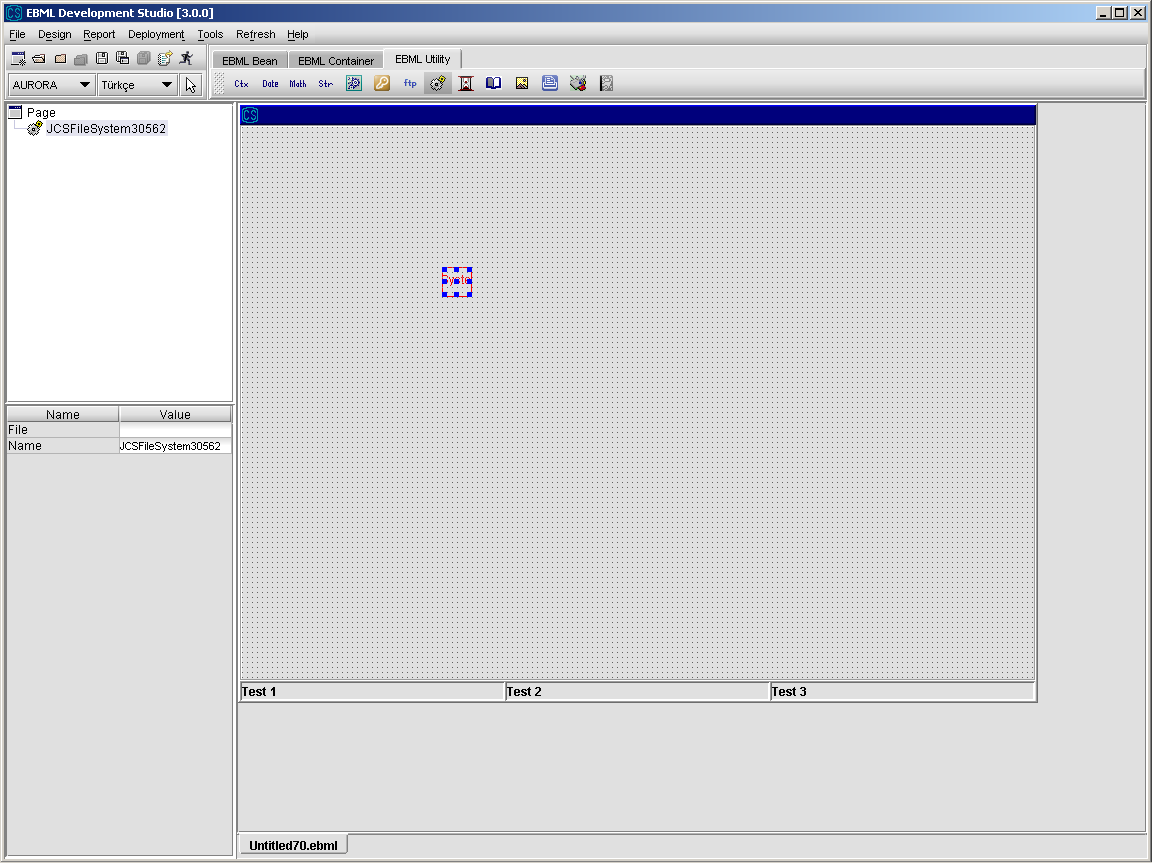
Finally, we relate this action with the “fileSelected” event of FilePicker component.



#### 3.3.3.6 JCSFileSystem

JCSFileSystem is used for file system operations like create file, delete file etc.

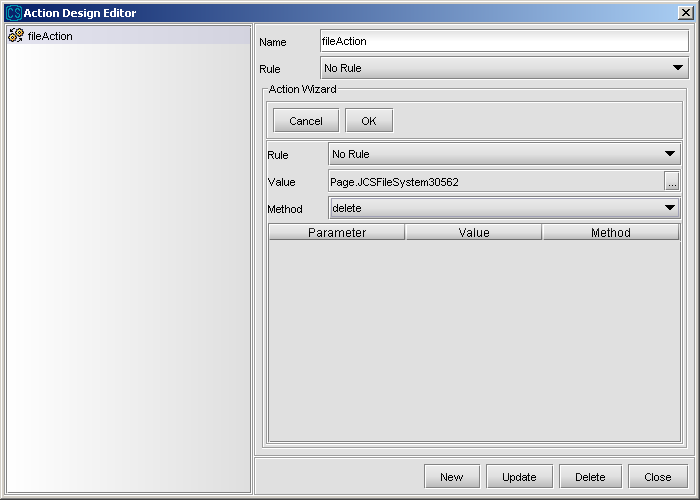
1. Select FileSystem component from the toolbar and drop into page.
2. Set the properties of FileSystem component.



##### 3.3.3.6.1 FileSystem Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of FileSystem component.

Details about “Bean Action” are given in ”Action Design” section.



**delete :** deletes the specified file stated in the “file” property

**exists :** checks if the specified file exists

**getLength :** gets length of the specified file

**isDirectory :** checks whether the specified file is a directory

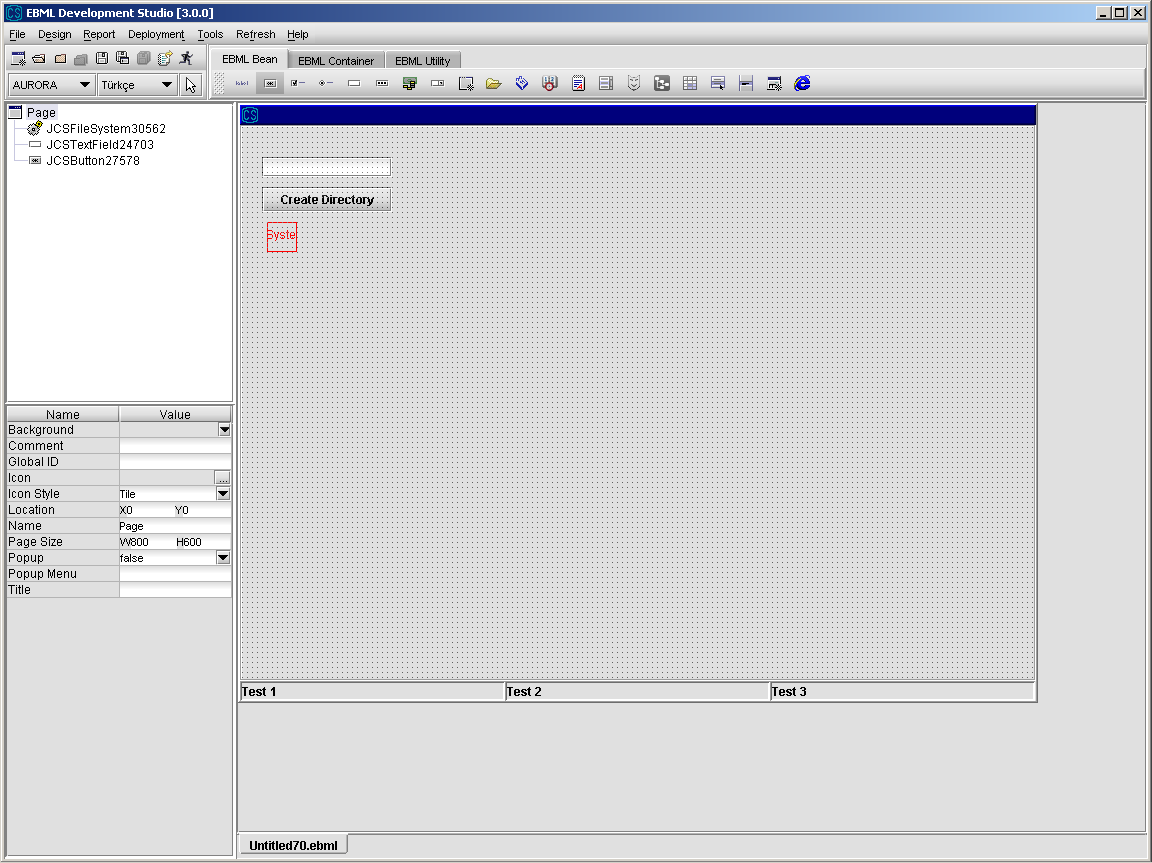
**makeDirectory :** creates a directory according to the specified file

**makeAllDirs(directories) :** creates directories for the list given in *directories*. An example for this methos is given below

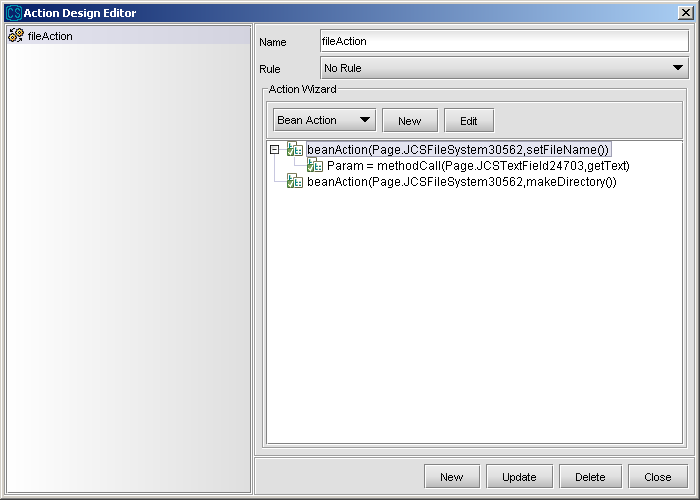
**getTempDirectory :** used for getting temporary directory of the JVM

**rename(name):** used for renaming the current file *name*

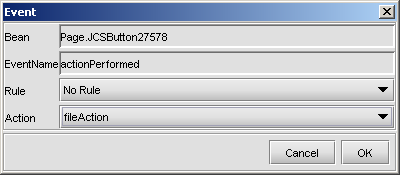
**Example:** Assume that we would like to design a screen that can create a directory we specified. The page should have a JCSFileSystem component, a JCSTextField component and a JCSButton component.

****

Firstly, we define an action to create directory. It gets the new directory name from the text field and sets it as the filename of the FileSystem component. Then it executes the makeDirectory method of the FileSystem component.

****

Then we relate this action with the actionPerformed event of the Button.



**makeAllDirs Example**: In the previous EDS version, this method belong to JCSDocTool component. Now this method has moved to here and JCSDocTool component has been removed.

Assume that we would like to design a screen that can create a list of directories. The page should have a JCSFileSystem component and a JCSButton component.



The makeAllDirs method of JCSFileSystem component accepts a list of directories as an input parameter. Thislist of directories must be presented as an xml-formated string like:

<array>

<value>directory1</value>

<value>directory2</value>

<value>directory3</value>

</array>

or can be given as constant string.

We can design a remote call like below that return such a string and we can give the output key “CORE\_ACTION” as input to makeAllDirs method of JCSFileSystem component:

**// SERVICE\_NAME : CADET\_TEMP\_SERVICE**

**public static CSbag tempService(CSBag inBag) throws CSException {**

**CSBag outBag = new CSBag();**

**try {**

**String xml = "<array><value>C:\\temp1</value><value>C:\\davut\\temp2</value></array>";**

**outBag.put("CORE\_ACTION",xml);**

**} catch (Exception e) {**

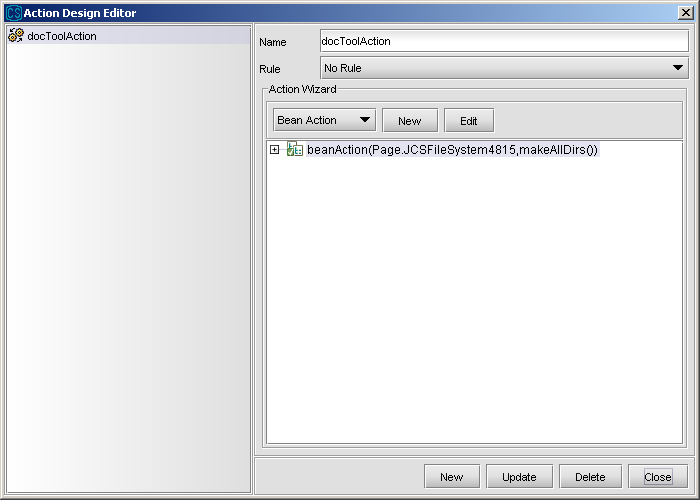
**e.printStackTrace();**

**}**

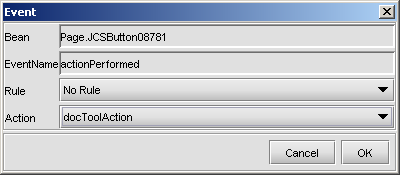
**return outBag;**

**}**

Another option is giving a constant string as input to makeAllDirs method.



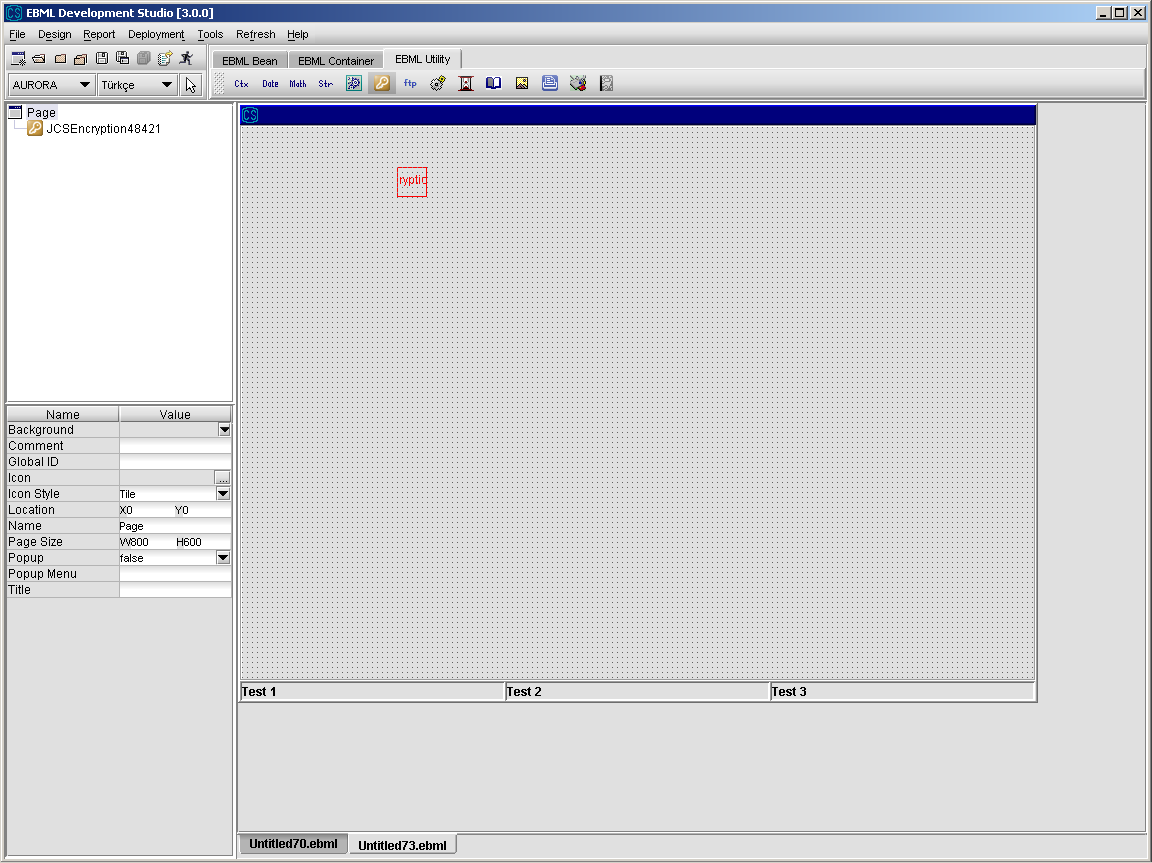
Then, we relate the actionPerformed event of the Button with this action.



#### 3.3.3.7 JCSEncryption

JCSEncryption component is used for encryption and decryption of files. It uses DES algorithm. Whıle encrypting, user states the input file to be encrpt, the output file (decrypted file) and key value. The JCSEncryption component creates a specific key from given key value, encryptes the imput file using this specific key, and creates the output file as an outcome. The decryption process is the reverse of encryption.

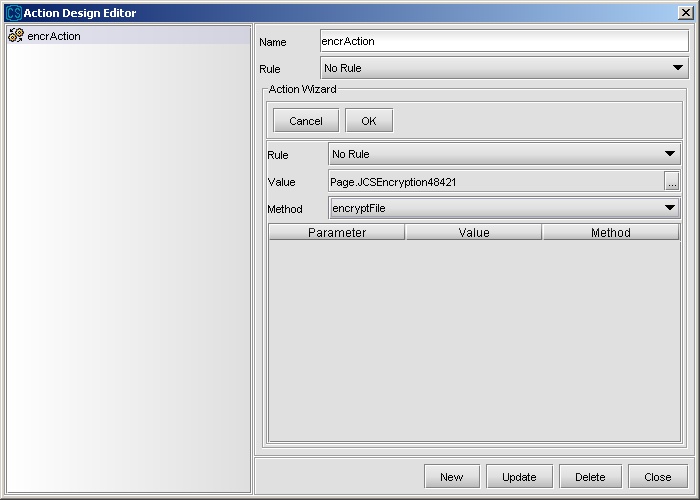
1. Select JCSDocTool component from the toolbar and drop into page.
2. Set the properties of JCSEncryption component.



##### 3.3.3.7.1 JCSEncryption Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Encryption component.

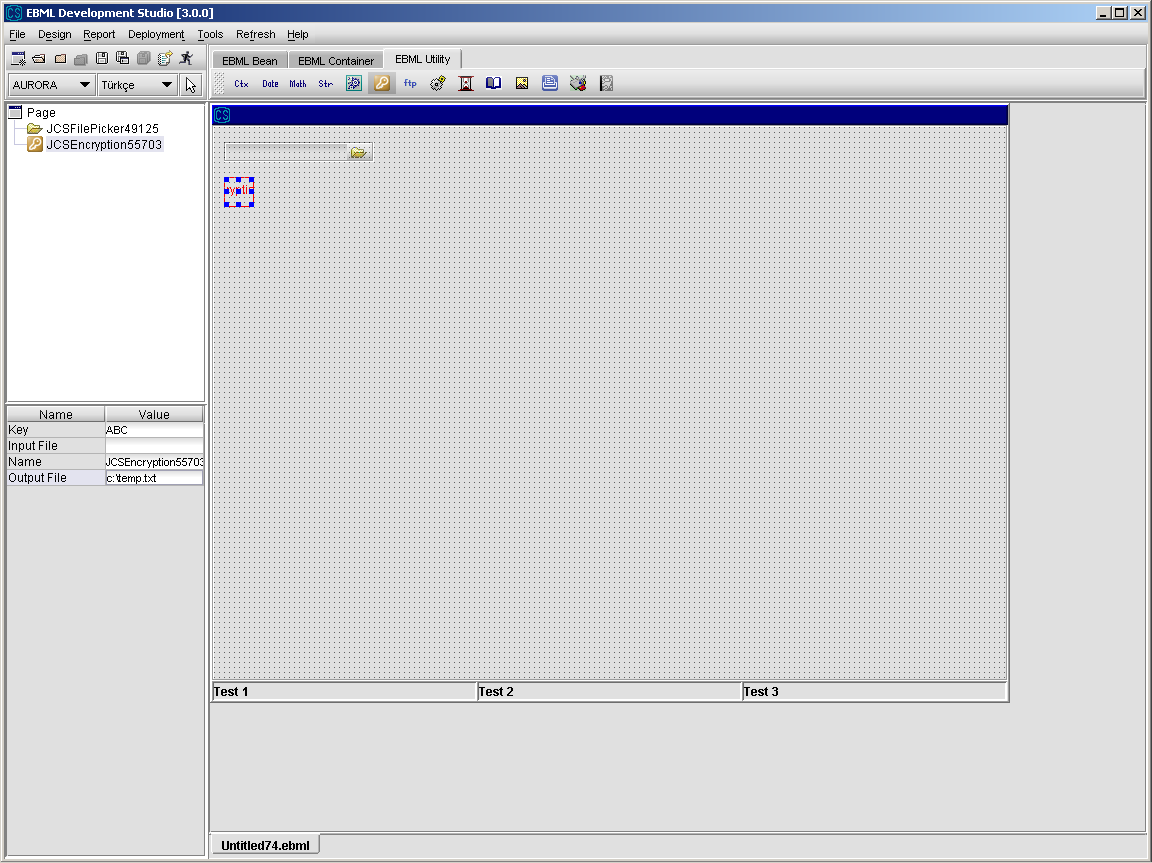
Details about “Bean Action” are given in ”Action Design” section.



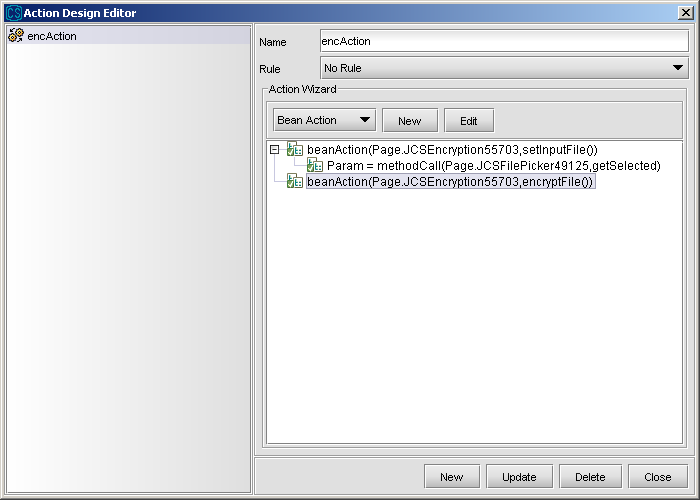
**decryptFile :** decrypts the input file with given key and results it in the output file

**encryptFile :** ecrypts the input file with given key and results it in the output file

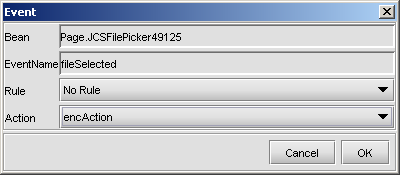
**Example:** Assume that we would like to design a screen that can encrypt a given file with key “ABC” and write to output file “C:\temp.txt”. The page should have a JCSEncryption component and a JCSFilePicker component.

****

When we select a file,we will set it as “input file” property of the JCSEncryption component. Then we will call the encryptFile method.



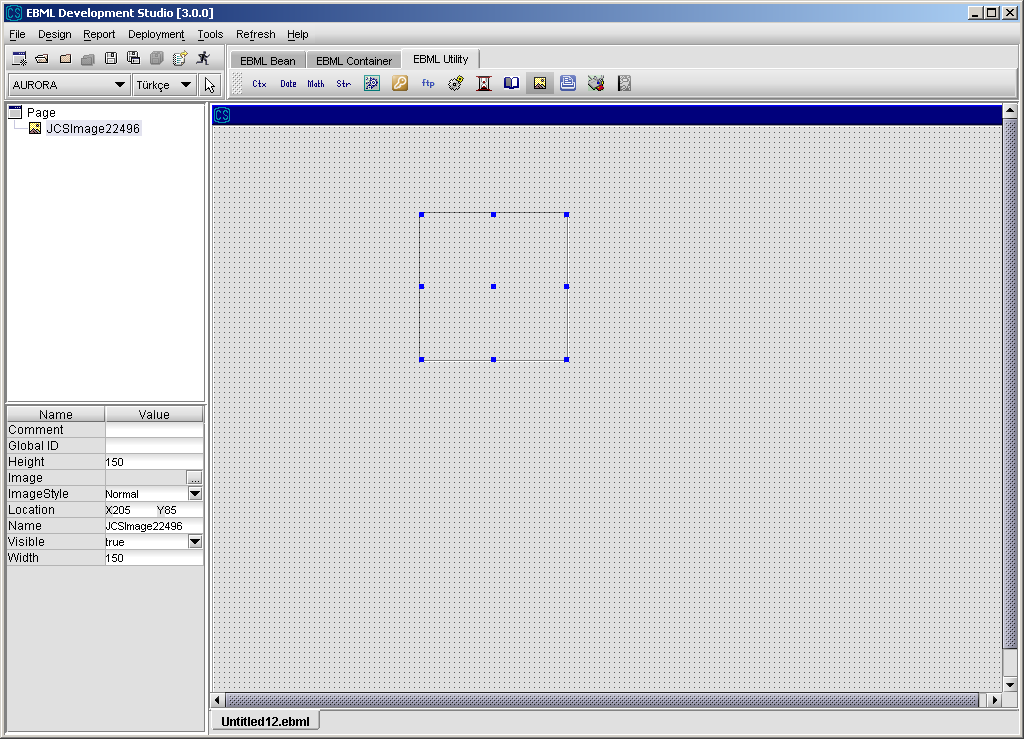
Finally, we will relate the fileSelected event of FilePicker with this action.



#### 3.3.3.8 JCSImage

JCSImage component is used to show images on the pages.

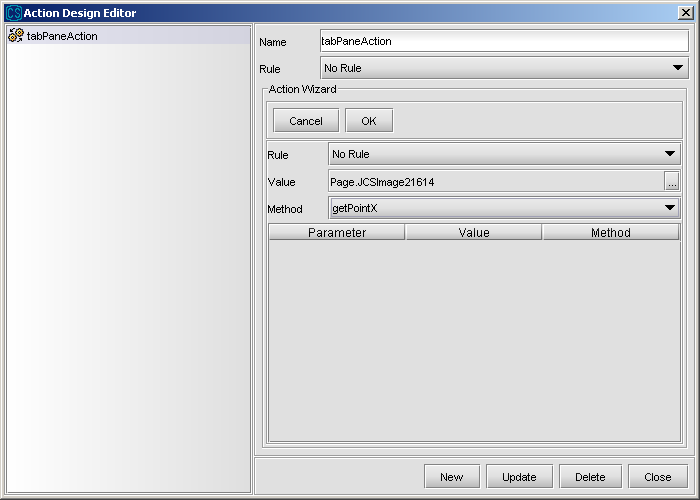
1. Select JCSImage component from the toolbar and drop into page.
2. Set the properties of JCSImage component.



##### 3.3.3.8.1 JCSImage Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of Image component.

Details about “Bean Action” are given in ”Action Design” section.

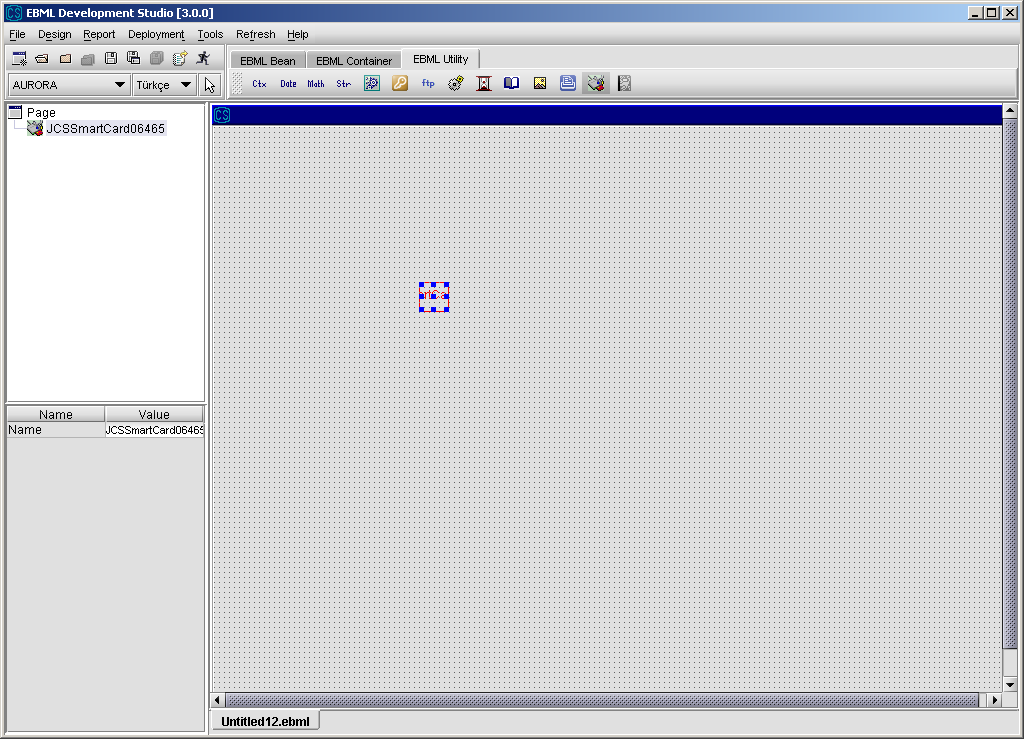


getPointX: the X coordinate of the last double-clicked point

getPointY: the Y coordinate of the last double-clicked point

setLocalImage(localImage): used for setting an image found locally on the client

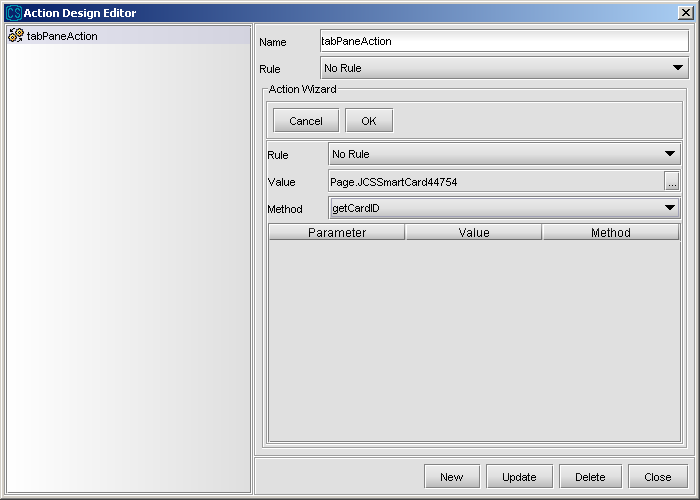
#### 3.3.3.9 JCSSmartCard



##### 3.3.3.9.1 JCSSmartCard Operations

From Design\Action Design Editor, select an action and select “Bean Action” option from Action Wizard to call methods of SmartCard component.

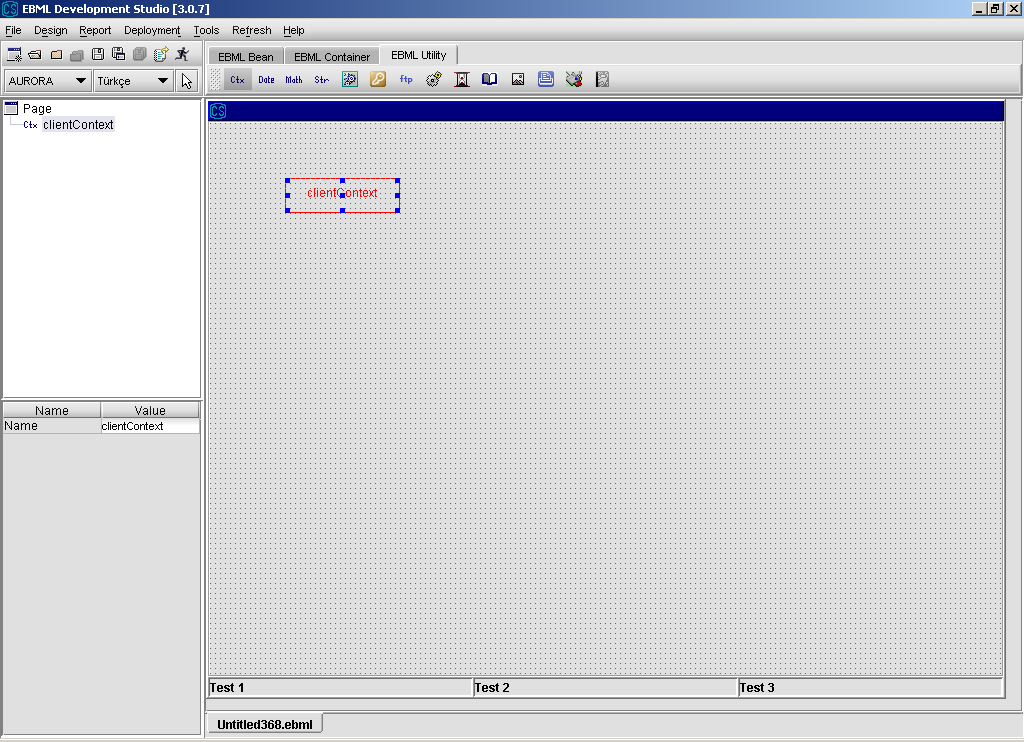
Details about “Bean Action” are given in ”Action Design” section.



getCardID: used for getting the ID of the smart card

#### 3.3.3.10 JCSClientContext

JCSClientContext is used for holding and transferring data between pages. Every data is stored and access with a unique key (just like a hashtable). JCSClientContext is designed to store simple and complex data elements. For instance, while you can store a string with a key, you can also store table columns with a key either. This component is single-instanced and any data can be accessed in any page in the same ERE. To use this bean click utility tab and choose it.



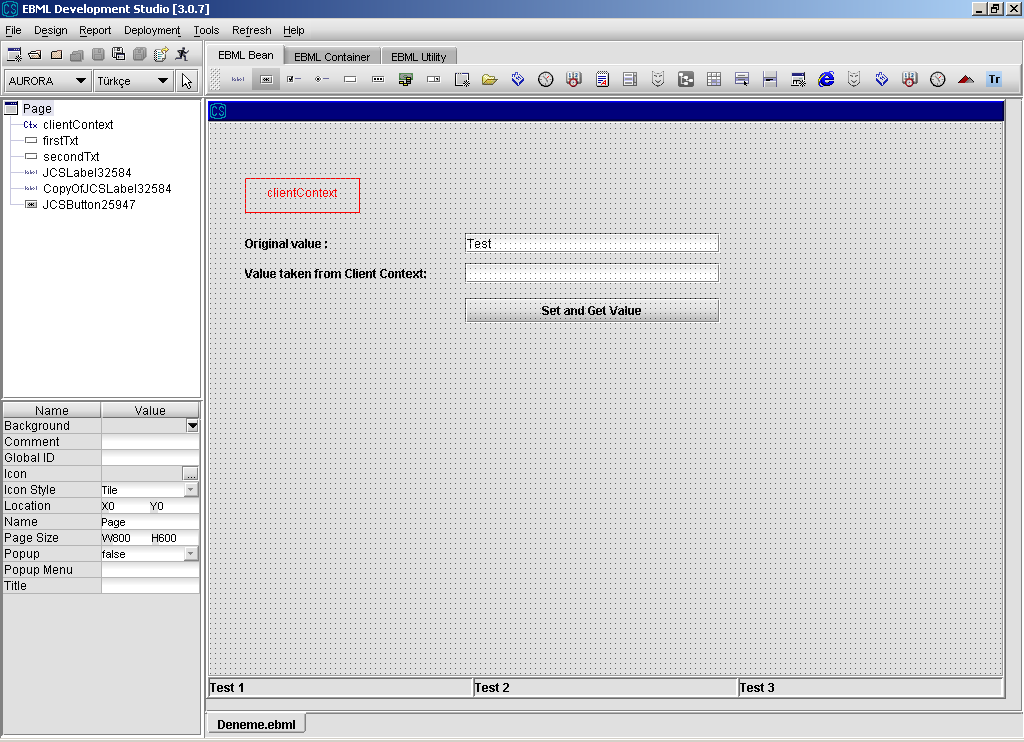
##### 3.3.3.10.1 JCSClientContext Operations

As seen in property editor, there is no property except name.

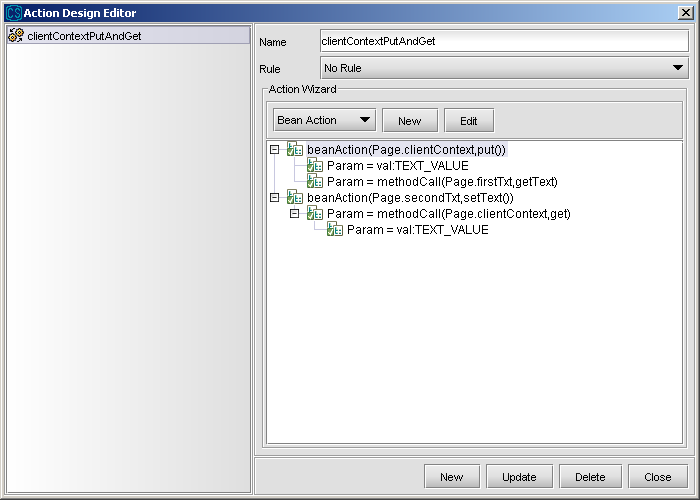
**clear :** Clears all data in JCSClientContext.

**put :** Puts a value with a specified key. This method accepts only two argument, key and value.

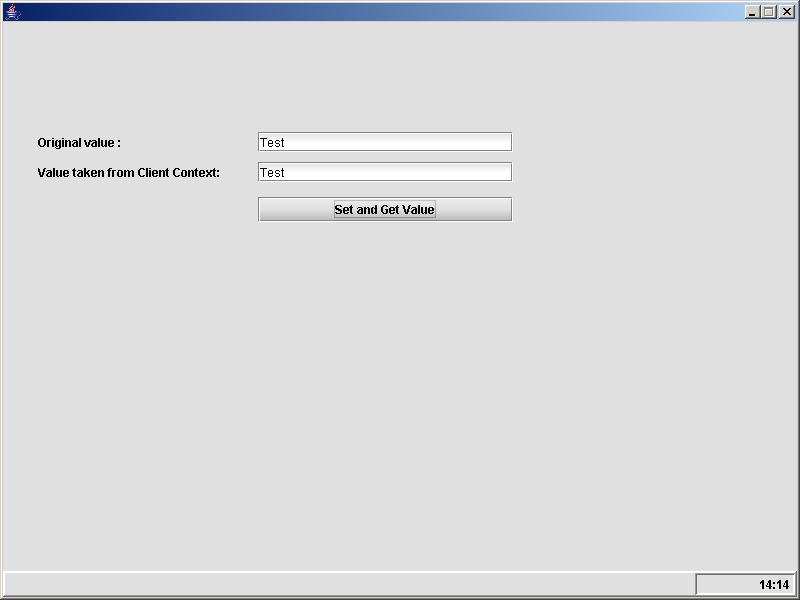
**get:** Gets a value with a spefied key. If no data available with this key, an empty string returns. This method only accepts only one argument, key.



In the example above, When button is pressed, value of top text field is stored in client context with key “TEXT\_VALUE” and read back with the same key and set to second text field. Relevant action design is show below.



This action is valid when a page is called. In the other word, client context is the same instance in every page in the same ERE. Next figure shows the result.



**put :** This method was designed for table component and accepts three argument. These are a key,a value and an index arguments. Index is used for table row. This means that “I store a value with the key “key” for row “index”.

**get :** This method is desgined for getting value with key and index. Unlike get method with one parameter(key), this method requires another parameter for row index. When you use put method with three parameters(key,index,value) you should read the same value with get method with parameters index and key.

**putItems :** This method is designed for only table operations(getItems, getItemsByName

and putItems). It requires two parameters, key and items which is xml representation of JCSTable. When you want to store whole table with a spefied key, you should call getItems method. This will return a xml string which describes a table. Any time, you can set that values back to the same table or another. If you want to set it back to another one, at least one column identifier must be the same.

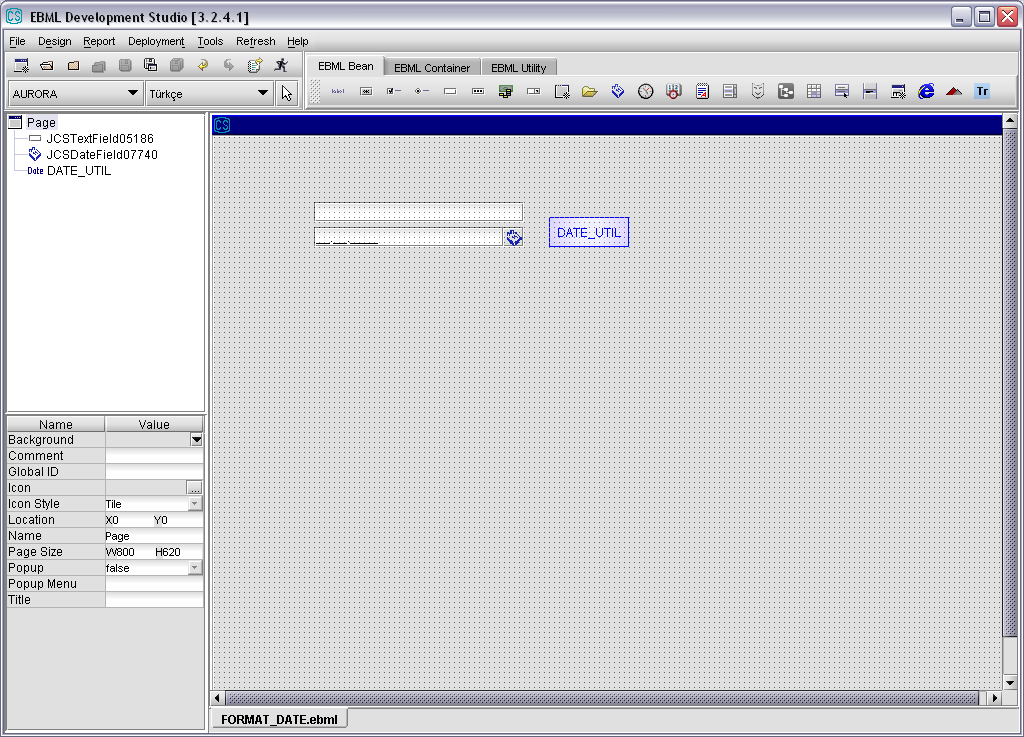
For instance, you have a table with column identifiers “NAME” ,“SURNAME” and “NUMBER”. Any table with (at least) one of these identifiers can be load with putItems method of the table.

**getItems :** This method returns xml data with a key.

**geItemsByName:** This method gets parameters of column identifiers to build an xml string with gven column identifiers. Identifiers are passed as “comma-separated string”.

#### 3.3.3.10 JCSDateUtility

JCSDateUtility is an utility class that manipulates on date strings. You add months,days or years to a date string. Or, you can difference two date. Also it is available to format a date for given input and output patterns.



##### 3.3.3.10.1 JCSDateUtility Operations

**addDay :** Adds number of days to date. Its calculates special cases.

**addMonth :** Adds number of months to date.

**addYear :** Adds number of years to date.

**difference :** Calculates how many days between two date

**formatDate** : Formats a given input date via given pattern for new output string via given output pattern. Patterns are very filexible to format output string. Date and time patterns are mentioned below.

#### Date and Time Patterns

Date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (') to avoid interpretation. "''" represents a single quote. All other characters are not interpreted; they're simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

|  |  |  |  |
| --- | --- | --- | --- |
| **Letter** | **Date or Time Component** | **Presentation** | **Examples** |
| G | Era designator | [Text](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#text) | AD |
| y | Year | [Year](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#year) | 1996; 96 |
| M | Month in year | [Month](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#month) | July; Jul; 07 |
| w | Week in year | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 27 |
| W | Week in month | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 2 |
| D | Day in year | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 189 |
| d | Day in month | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 10 |
| F | Day of week in month | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 2 |
| E | Day in week | [Text](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#text) | Tuesday; Tue |
| a | Am/pm marker | [Text](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#text) | PM |
| H | Hour in day (0-23) | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 0 |
| k | Hour in day (1-24) | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 24 |
| K | Hour in am/pm (0-11) | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 0 |
| h | Hour in am/pm (1-12) | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 12 |
| m | Minute in hour | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 30 |
| s | Second in minute | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 55 |
| S | Millisecond | [Number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number) | 978 |
| z | Time zone | [General time zone](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#timezone) | Pacific Standard Time; PST; GMT-08:00 |
| Z | Time zone | [RFC 822 time zone](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#rfc822timezone) | -0800 |

Pattern letters are usually repeated, as their number determines the exact presentation:

* **Text:** For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
* **Number:** For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
* **Year:** For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a [number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number).

For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.

* **Month:** If the number of pattern letters is 3 or more, the month is interpreted as [text](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#text); otherwise, it is interpreted as a [number](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#number).
* **General time zone:** Time zones are interpreted as [text](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#text) if they have names. For time zones representing a GMT offset value, the following syntax is used:
* *GMTOffsetTimeZone:*
* GMT *Sign* *Hours* : *Minutes*
* *Sign:* one of
* + -
* *Hours:*
* *Digit*
* *Digit* *Digit*
* *Minutes:*
* *Digit* *Digit*
* *Digit:* one of

0 1 2 3 4 5 6 7 8 9

*Hours* must be between 0 and 23, and *Minutes* must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

For parsing, [RFC 822 time zones](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#rfc822timezone) are also accepted.

* **RFC 822 time zone:** For formatting, the RFC 822 4-digit time zone format is used:
* *RFC822TimeZone:*
* *Sign* *TwoDigitHours* *Minutes*
* *TwoDigitHours:*

*Digit Digit*

*TwoDigitHours* must be between 00 and 23. Other definitions are as for [general time zones](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#timezone).

For parsing, [general time zones](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html#timezone) are also accepted.

SimpleDateFormat also supports localized date and time pattern strings. In these strings, the pattern letters described above may be replaced with other, locale dependent, pattern letters. SimpleDateFormat does not deal with the localization of text other than the pattern letters; that's up to the client of the class.

#### Examples

The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

|  |  |
| --- | --- |
| **Date and Time Pattern** | **Result** |
| yyyy.MM.dd G 'at' HH:mm:ss z | 2001.07.04 AD at 12:08:56 PDT |
| EEE, MMM d, ''yy | Wed, Jul 4, '01 |
| h:mm a | 12:08 PM |
| hh 'o''clock' a, zzzz | 12 o'clock PM, Pacific Daylight Time |
| K:mm a, z | 0:08 PM, PDT |
| yyyyy.MMMMM.dd GGG hh:mm aaa | 02001.July.04 AD 12:08 PM |
| EEE, d MMM yyyy HH:mm:ss Z | Wed, 4 Jul 2001 12:08:56 -0700 |
| yyMMddHHmmssZ | 010704120856-0700 |
| yyyyMMdd | 20041206 |
| dd / MM / yy | 06 / 12 / 04 |
| EEEE, MMM d, ''yy | Wednesday, Jul 4, '01 |
| EEEE, MMMM d, ''yy | Wednesday, July 4, '01 |
| ‘Today is ’ dd.MM.yyyy | Today is 12.06.2004 |

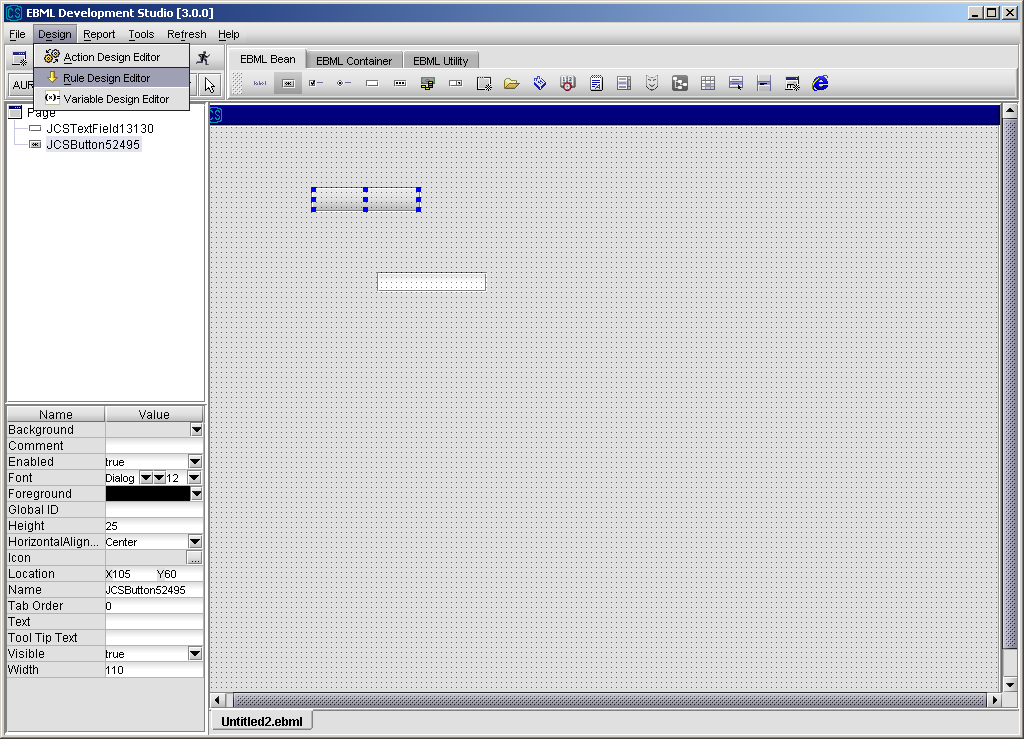
## 3.4 Rule Design

Rule is used to define different actions according to different controls on the page. For example, while a value chosen from combobox is *firstValue* we may want to go page A and we may want to go page B while another value is chosen from the combobox. With the use of Rule Based Engine (RBE), firing of events depends on fulfilling some conditions.

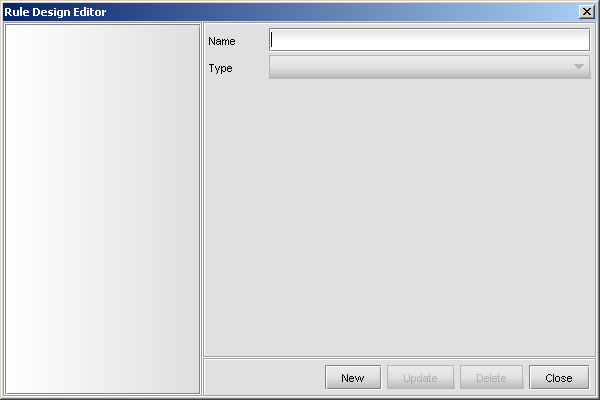
The rules in ERE may have two values, either true or false. The value of a rule is calculated when the event depending on this rule is fired. If the value of rule is true then the event is fired, otherwise event is not fired.

Developer can design a rule using EBML Development Studio – Design Menu.

1. Select Rule Design Editor.

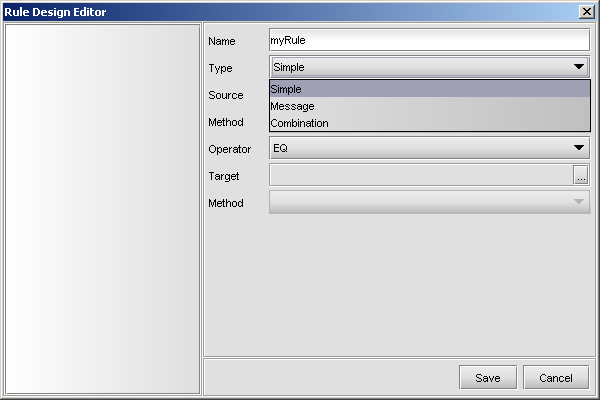


1. Press New button to define a new rule or select a predefined rule from the list on the leftside to update it.

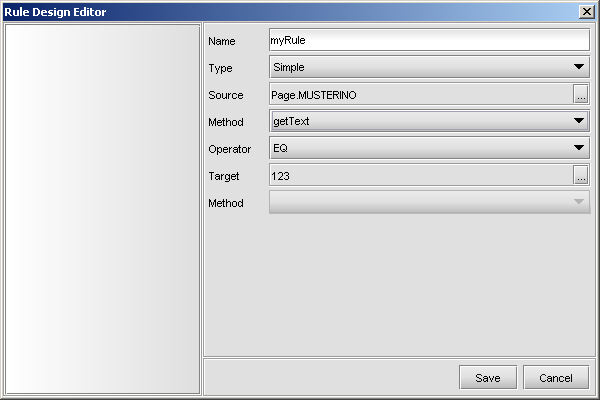


1. Enter the new rule name and select the rule type from Type combobox. There are 3

options as Simple, Message and Combination here.



**Simple** type is used to define a rule that depends on the value of a component or a variable. There are three parts to set in order to define this type of a rule.



Source part: In this part, the EBML name of the component or variable whose value will be checked is given. If a component is chosen, a method that belongs to that component and returns a value should also be selected. For the above examples getText method of component “Page.MUSTERINO” will be executed to evaluate the rule “myRule”. (\*EDS supports selecting table columns as source part)

Target part: In this part, a target value with whom the source component will be compared is given. For example, for the above example the target value is a constant string “123”. Target part, can be a constant, variable or component. As in source part, if a component is chosen, a method that belongs to that component should also be chosen. The value returned by getText method of component “Page.MUSTERINO” will be compared with the string “123” according to comparison operator for the above example.

Operator part: This part determines, how the comparison between source and target will be realized. There are several possible operators for making comparison here:

ISEMPTY : source is empty

GTE : source is greater than or equal to target

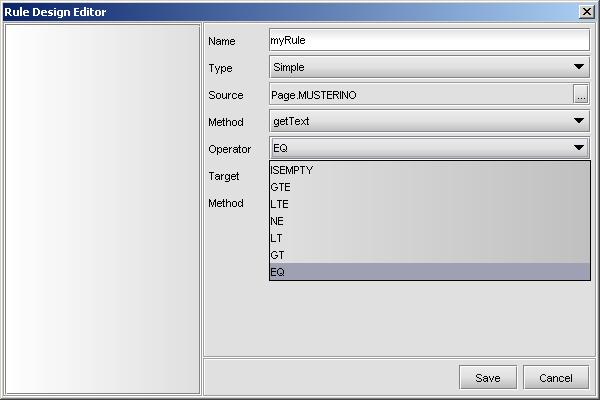
LTE : source is less than or equal to target

NE : source is not equal to target

LT : source is less than target

GT : source is greater than target

EQ : source is equal to target

**

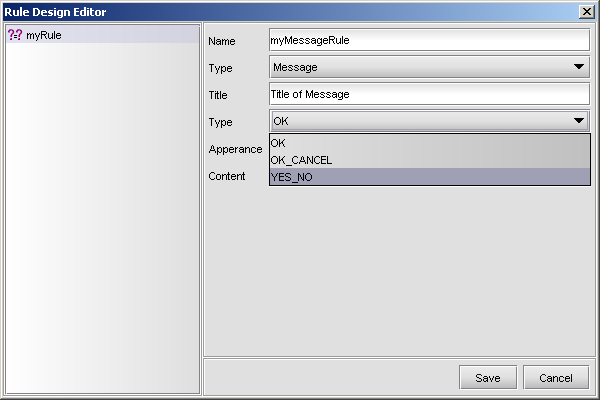
**Message** is used to define messages that are given to the user before making an operation for confirmation or information (or warning) purposes. There are four parts to set in this type of rule.

Title : The text to display as the title of the message dialog.

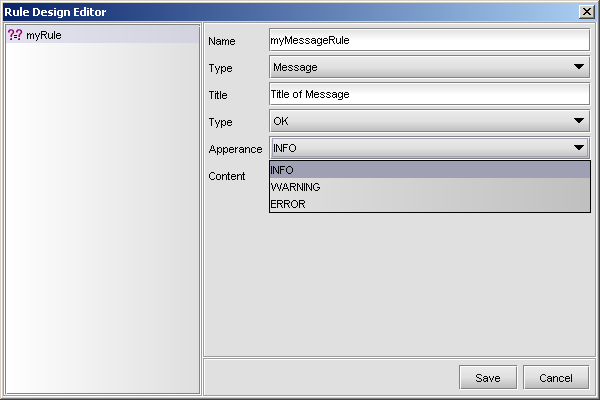
Type: There are three types of messages :

* **Information or Warning Messages:** In this type of messages, a message and an “OK” button appears and this type of messages are always evaluated as “True” when the button is clicked.

1. **Confirmation Messages:** In this type of messages, user is given a message and two choices “Ok” and “Cancel”. When the user clicks one of these two buttons, the rule is evaluated. If “Ok” button is clicked rule is evaluated as “True”, if “Cancel” button is clicked rule is evaluated as “False”.
2. **Question Messages:** For question messages, user is given a message and two choices, Yes or No. When the user clicks one of these two buttons, the rule is evaluated. If “Yes” button is clicked rule is evaluated as “True”, if “No” button is clicked rule is evaluated as “False”.

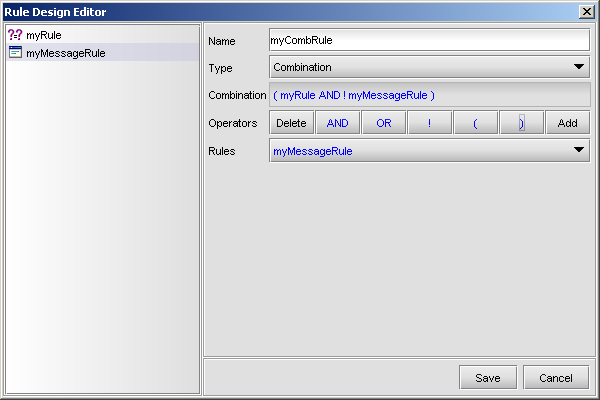


Appearance: The apperance choice determines the color and icon of the message box.

****

Content :The message content that will be displayed. It can be a string constant or can be taken from a variable.

**Combination** is used to define a rule that is a combination of other rules. The other rules can be chosen from all other type of rules. For this type of rules, the name of the rule and how the rule depends on other rules are given by using logical operators (AND, OR, !) and parenthesis when necessary.

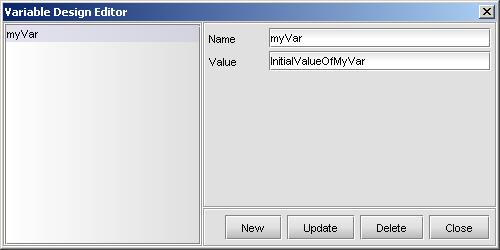


## 3.5 Variable Design

ERE Store-Variables are used for storing values in the page. Basically there can be two operations; assigning a value to a variable and getting the value from the variable. All types of values can be assigned to these variables (integer, string, etc.).

Developer can define a new variable to be used in the services using EBML Development Studio – Design Mode.

Select Variable Design Editor from Design menu.



* Variables can be used as an input for a service and can be used in rule definitions.
* Variables can be assigned to a component or a component value can be assigned to a variable.
* Variables can be used as input or output to a popup.
* Variables can be used in Remote Calls.

Also there are built-in variables in ERE from which developers can get information about user identity, branch id, branch name, cash desk, today’s date, and user name.

These variables are;

$today : Current date

$menuKey : menuKey property that is given during the menu definition

$clienthostname : Client’s host name, lt008

$clientosname : Operating System name, e.g. Windows XP

$securitylevel : Security level of the current user. Enabling/disabling some

fields of the screen according to the value of this variable is

possible.

$RC\_ERROR\_ID : If an exception is get from a remote call, the id of the exception is set to this variable. If the remote call is finished successfully, the value of it is set to –1.

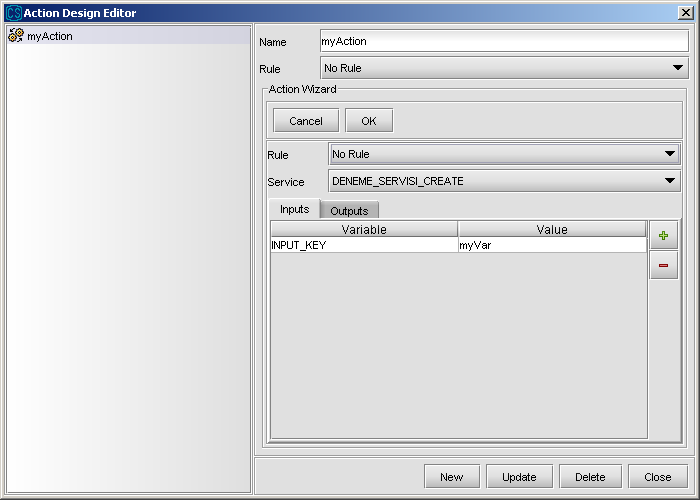
### 3.5.1 Variable Usage In Actions

Variables can be used as an input or output for a remote call, in rule definition, with components (assigning a value to a component and getting value of a component), with popups (as input/output).

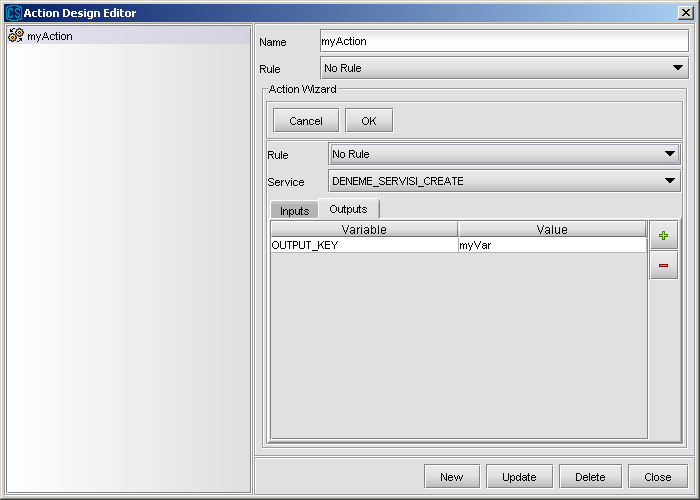
#### 3.5.1.1 Variable usage as an input/output for a remote call

The following example shows us how a variable can be used as an input for a remote call.

**Example:** DENEME\_SERVISI\_CREATE is selected as remote call service. A service specific variable INPUT\_KEY is used as an input. The value of the “INPUT\_KEY” is selected as the value of variable “myVar” which is defined in *Variable Design Editor*.

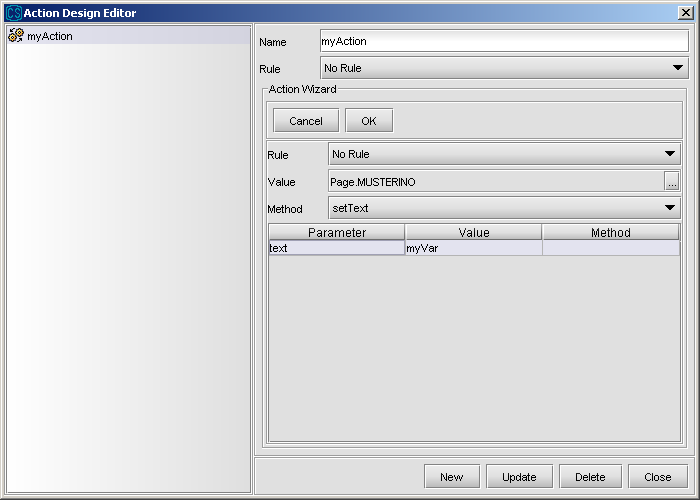


This service may return a value and it can also be assigned to “myVar” as shown below. In this example, value of “OUTPUT\_KEY” key is assigned to variable “myVar”.



#### 3.5.1.2 Variable usage with components

The following example shows using the value of a variable as an input to a method of component. The value of the variable “myVar” is used as input to the “setText” method of text component called “MUSTERINO”.

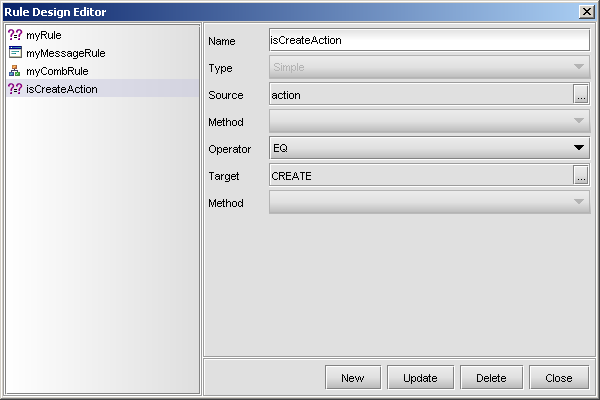


In a similar way, the output value of component method may also be assigned to a variable.



### 3.5.2 Variable usage in rule definition

**Example:** Rule “isCreateAction” is used for determining whether the action is a CREATE action. For this operation, variable “action” is used. The value of it compared with constant “CREATE”



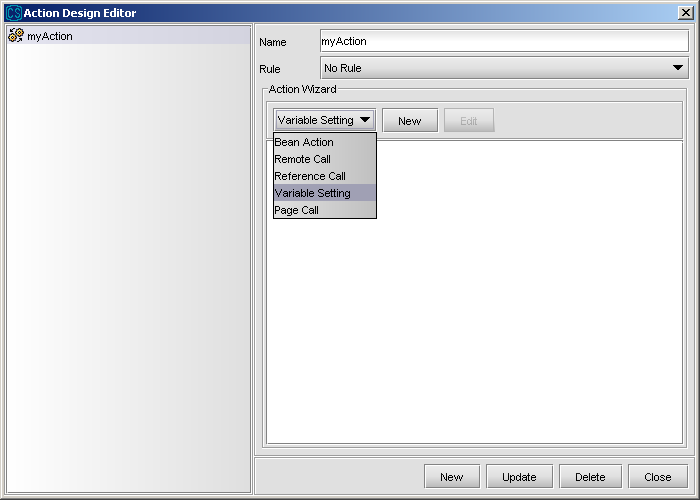
### 3.5.3 Variable usage with popups (as input/output)

The variables can be passed as parameters to popups and the output of popups can be assigned to variables.

## 3.6 Action Design

Developer can define a new action using EBML Development Studio – Design Menu. Then these actions can be linked to events.

1. Select Action Design Editor from Design menu. If you want to design a new action press “New” button at the bottom of the page, give a name for the action and press “Save” button. If you want to change a predefined action, select it from action list from left side.



Select the related option from Action Wizard panel. There are five type of sub-actions that can be added to an action.

**Bean Action :** used for defining method calls for the beans that are in current page.

**Remote Call :** used for defining a new remote call (remote event).

**Reference Call :** used for executing previously defined action.

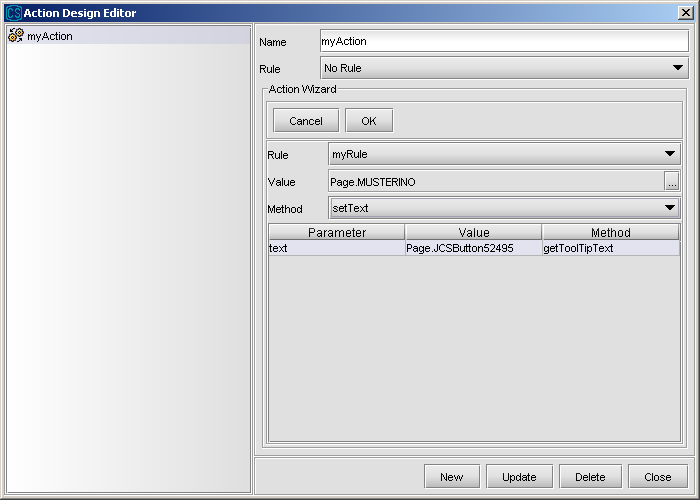
**Variable Setting :** used for assigning values to previously defined variables. Variables are defined from Variable Design Wizard.

**PageCall :** used for defining a page call, namely passing to another page.

### 3.6.1 Bean Action

This type of action is used for defining method calls for the beans that are in current page. As in other sub-actions, execution of it can be dependent on a rule.

In order to create a new Bean Action, select Bean Action from combobox in Action Wizard panel from Action Design Editor and click New button.

****

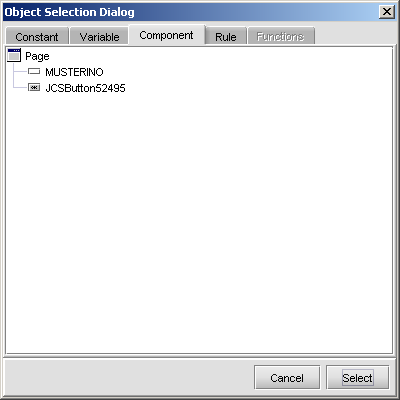
While defining a bean action, the following properties should be determined:

Rule: If the execution of bean action depends on a rule, dependent rule should be selected from rule combo.

Value: Which bean’s method will be called.

Method: Which method of the bean will be called.

Parameters: What will be the input parameters’ value. When a method is selected the name of its input parameters are listed. The value of parameter is chosen from following dialog. The value can be a constant, variable, component or rule. If it is a component, then a method for it should be selected. The output of this method will be the value of the parameter.

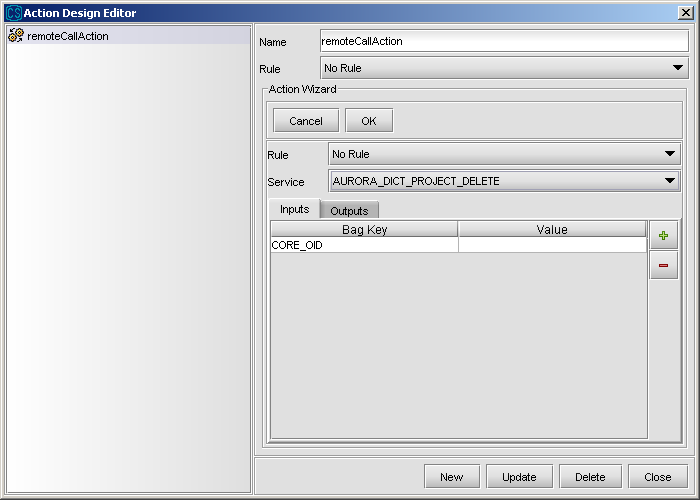


### 

### 3.6.2 Remote Call Action

This type of sub-action is used to call services defined on the server.

In order to create a new Remote Call Action, select Remote Call from combobox in Action Wizard Panel from Action Design Editor and click New button.



While defining a remote call action, the following properties should be determined:

Rule: If the execution of remote call action depends on a rule, dependent rule should be selected from rule combo.

Service:used for selecting the service to be called in remote event. All of the services deployed to the server are listed in service combobox.

Input/Output parameters: Selected service may require some inputs to work correctly. Also, it may return some outputs that you may want to use. Input and output parameters are referenced via bag keys. For example, if a service requires a bag key like, CUSTOMER\_NO, you should give the value of CUSTOMER\_NO key to the service as input. If input/output parameters are determined during service definition, they are listed in this screen when the service is selected. You only determine what will the values of these keys be. On the other hand, you can also add new bag keys by clicking “+” button, in that case you should write bag key name, or remove a predefined bag key by clicking “-“ button.

**Example :** Let’s define a very simple remote call. In this remote call, “Project” combobox will be filled with data by calling a related Data Dictionary service, when this action executed as a result of an event.

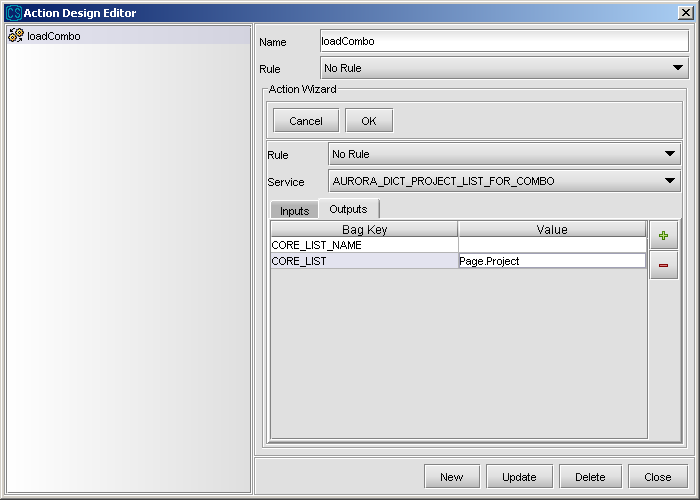
1. After defining an action, add a remote call action under this action. Select the service to be called. In this case we select a Data Dictionary service

Service Name : AURORA\_DICT\_PROJECT\_LIST\_FOR\_COMBO

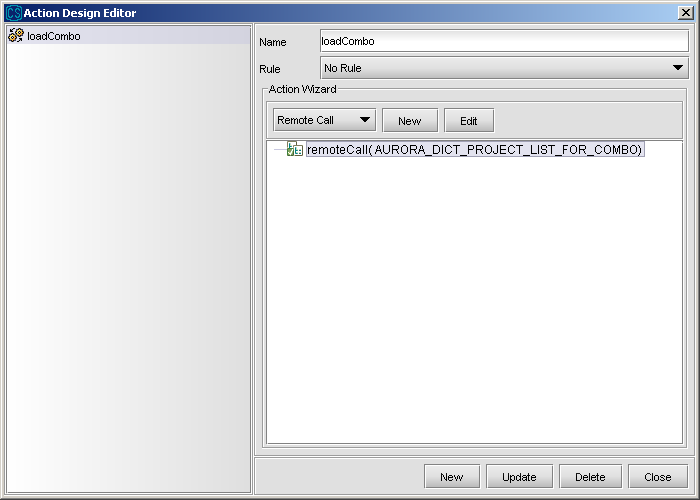
Input Parameters : none

Output Parameters : CORE\_LIST and CORE\_LIST\_NAME

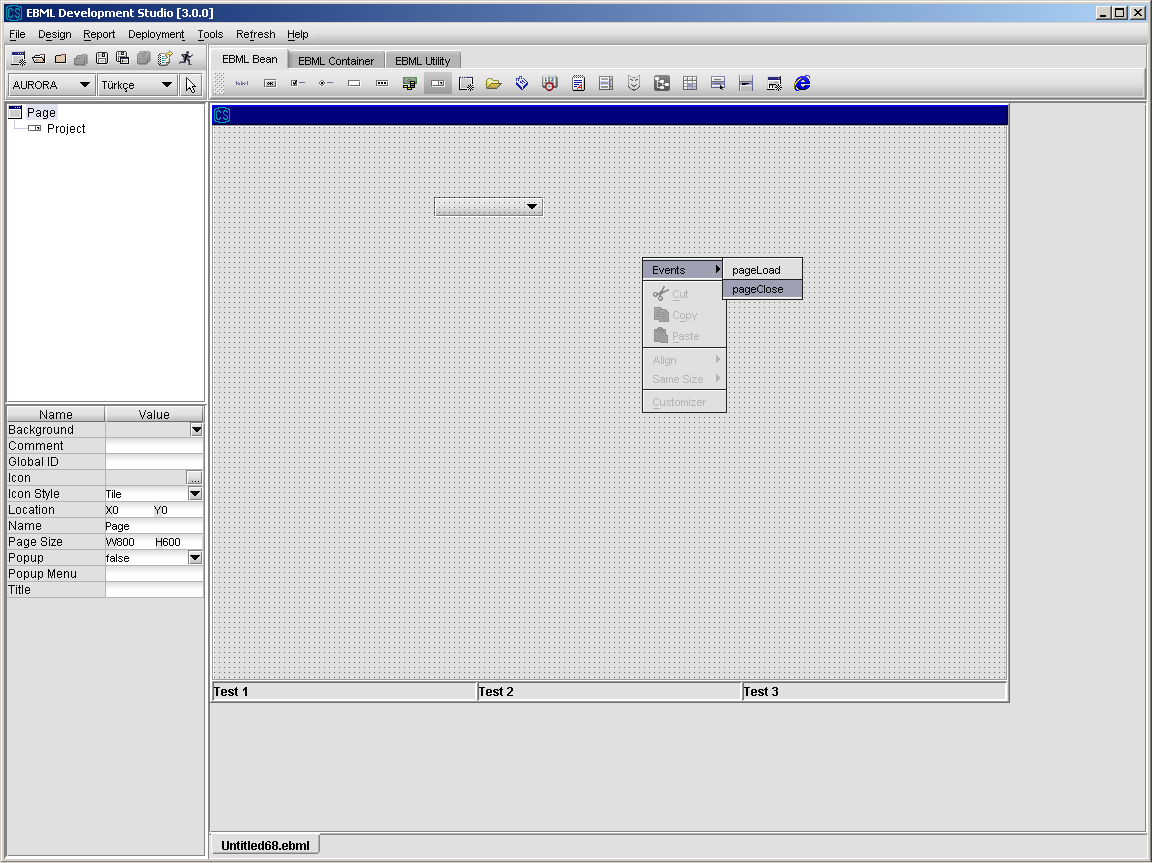
When we select the service, its keys for output parameters are listed. We use only CORE\_LIST parameter, since service returns the project name with this parameter. We assign the value of “CORE\_LIST” output key, to the “Page.Project” bean.



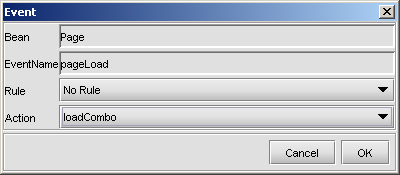
1. Click the Ok button to save the defined remote call.

****

1. Close the action dialog and click right mouse on the design environment. Select **Events-pageLoad** from appearing popup menu.



4. After Event dialog appears, select “loadCombo” action from Action combobox. By doing that, you say loadCombo action should run when page is loaded.

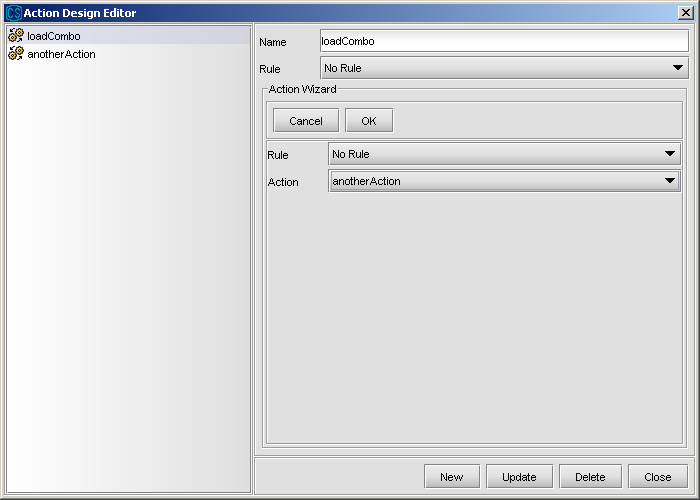


### 

### 3.6.3 Reference Call Action

This type of sub-action is used to call predefined actions.

In order to create a new Reference Call Action, select Reference Call from combobox in Action Wizard Panel from Action Design Editor and click New button.



While defining a reference call action, the following properties should be determined:

Rule: If the execution of reference call action depends on a rule, dependent rule should be selected from rule combo.

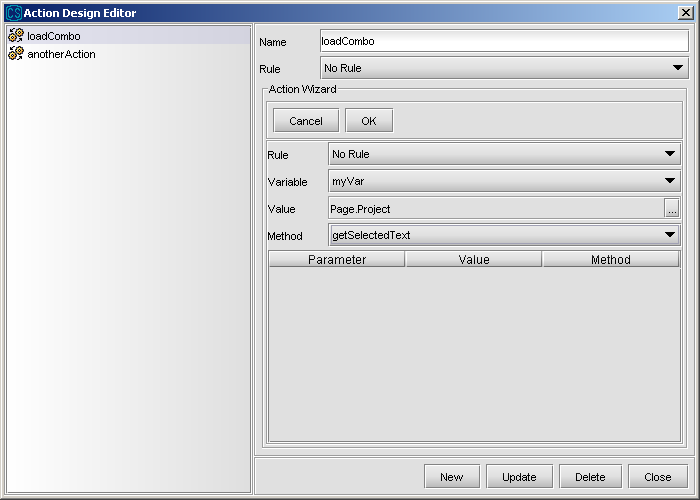
Action: The action to be called should be selected.

In the above example, we add a reference call action to action “loadCombo”. We have another action which is “anotherAction”. When the execution sequence of this sub-action comes, it calls “anotherAction” and it takes the turn. When “anotherAction” finishes its execution, “loadCombo” action continues from where it stays.

### 3.6.4 Variable Setting Action

This type of sub-action is used to change the values of variables. Only the value of user defined variables can be changed.

In order to create a new Variable Setting Action, select Variable Setting from combobox in Action Wizard Panel from Action Design Editor and click New button.



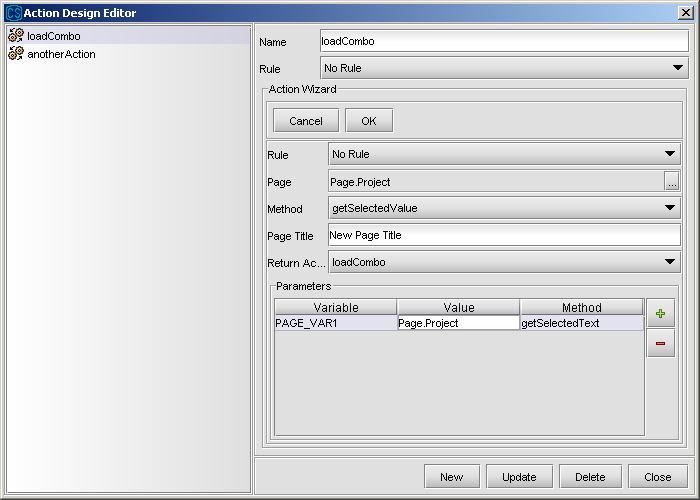
The new value of the selected variable can be a contant, value of another variable, result of a rule or result of a method of a component. If a component is selected in value field, one of its method and its input parameters must also be selected.

In the above example, we want to change the value of myVar. While determining the new value of it, we execute getSelectedText method of bean Page.Project and assign the result of this method to the variable.

### 3.6.5 Page Call Action

This type of sub-action is used to pass to another screen.

In order to create a new Page Call Action, select Page Call from combobox in Action Wizard Panel from Action Design Editor and click New button.



While defining a page call action, the following properties should be determined:

Rule: If the execution of page call action depends on a rule, dependent rule should be selected from rule combo.

Page: The name of the page that will be called is assigned here. It can be assigned from a constant, a variable or a method call of a component.

Method: If a component is selected for Page field, a method belonging to this component should be selected. The result of that method will be assigned as page name.

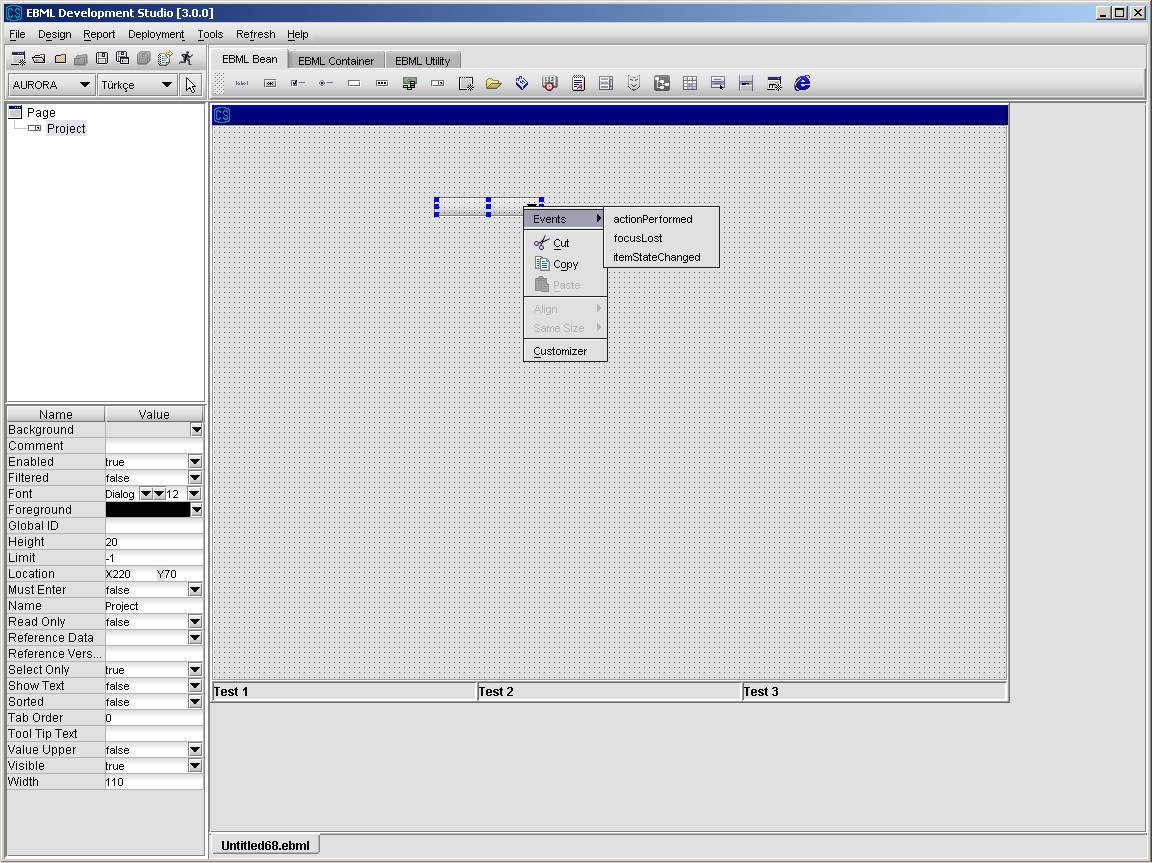
Page Title: A new title other than the original title of the page can be given here.

Return Action: After the called screen is closed, an action can be called. It is determined in this combobox.

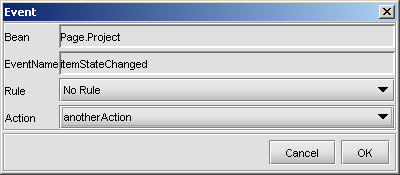
## 3.7 Event Design

After actions to be taken are defined, they should be connected with events. Namely, we should determine in which cases defined actions should be executed.

Events occur either due to some user interaction or due to some internal processes. However, an event always belongs to a bean. In general, we can see the events of a component by selecting and right mouse clicking on it. In that case, a popup menu appears. If you select Events menu, events belonging to the bean can be seen.

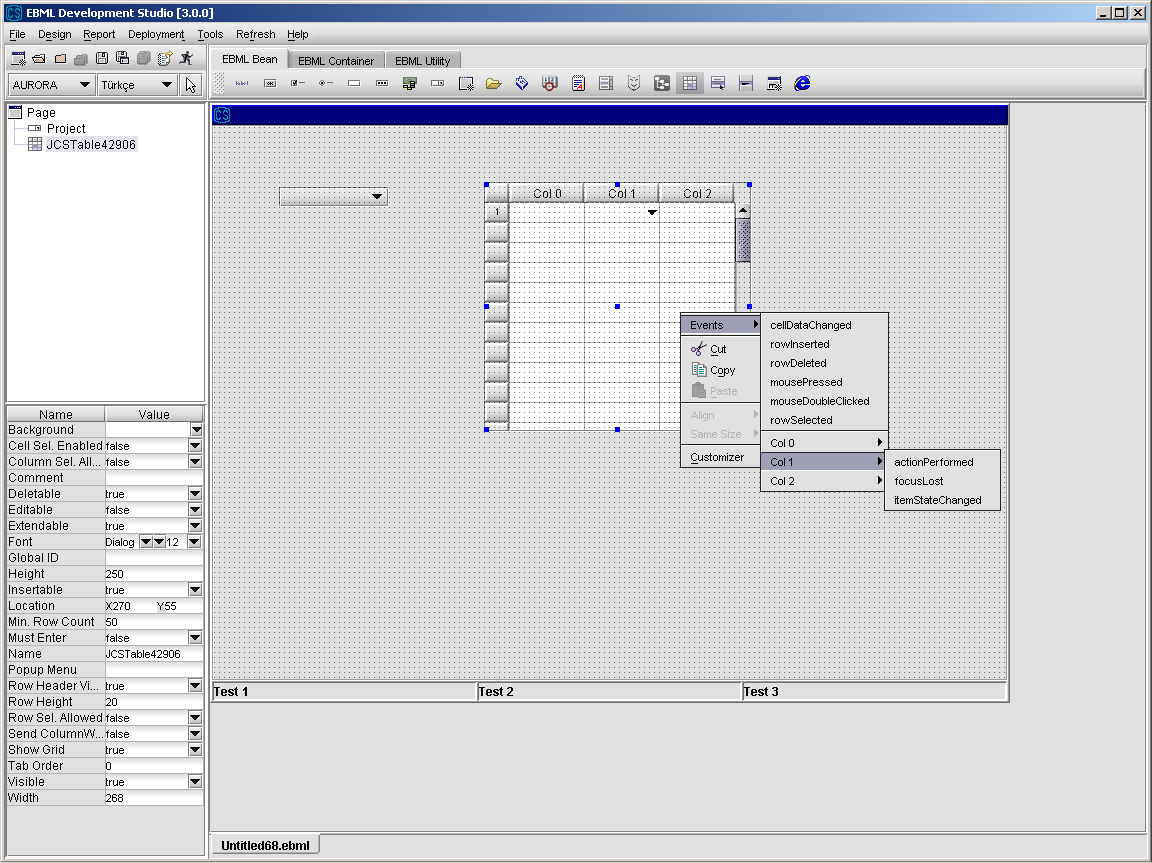


In the above example, events of a combobox is listed. If you want to perform an action when an event of this combobox occurs, select the rule that event execution is dependent, the event and determine what action to perform using below dialog. If execution of action is wanted to depend on a rule, this rule should be defined firstly and selected from rule combobox in the event dialog. In that case, when the event is fired, before the action is executed, the value of the rule is calculated. If rule returns true, action is executed, otherwise nothing is done.

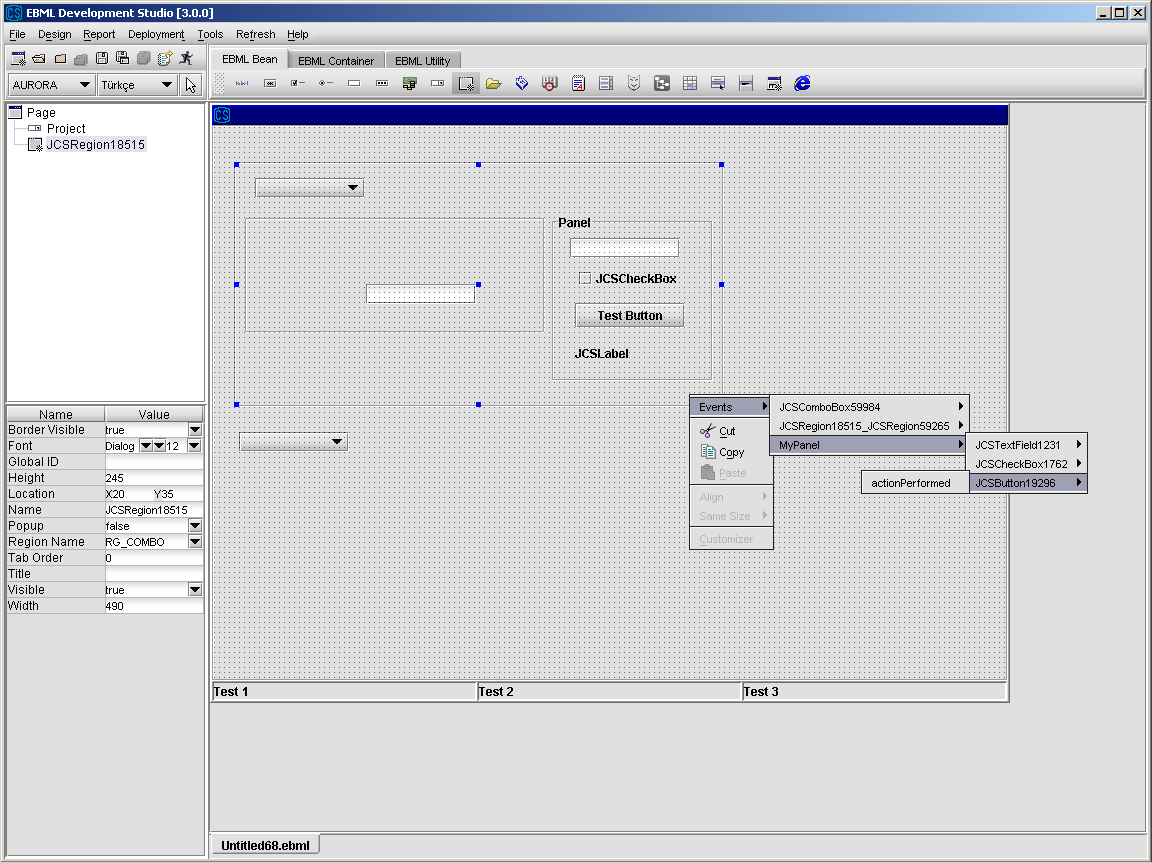


Events are combined with actions in a similar way for all components having an event. However, some components like JCSTable and JCSRegion have also some additional offerings.

As can be seen from the below screen, you can assign some actions to table events like cellDataChanged, rowInserted, etc. Also, since the table column named “Col2” is a combobox, you can define actions for events occurring for this column. The events that you can use change according to the type of the column.



Like tables, regions also have a speciality. You can reach all the components in the region by right clicking on the region and assigning actions to their events. In the below example, we are reaching to “Test Button” which is in the selected region. In order to do this, we select the region and right click on it. When popup appears, we select Events menu and then find the target component. The button is in a panel, so we first select the panel and then the button. When we select the button, its events appears. When we click on the event that we want to assign action, event-action assignment dialog appears.

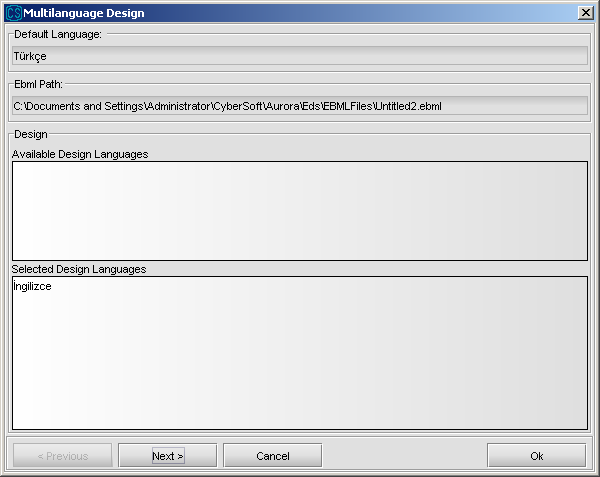


## 3.8 Multilingual Design

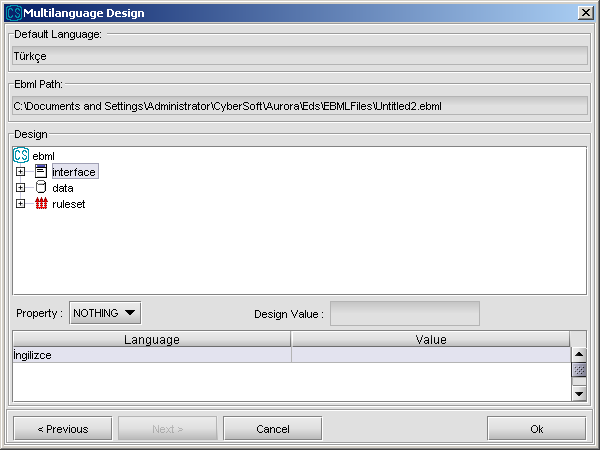
EDS provides designing screens for several languages. Therefore, instead of designing different pages for every language, it is possible to design a screen and set only the parts that change according to the language. Multilingual Design Wizard can be reached from menu Tools\MultiLingual Design.

The page is saved in default language and parts changing according to language are added to end of “ebml” file. When the file is deployed, it produces an “ebml” file for every language on the fly. For example, if the default language is Turkish and the page is also designed for English; EDS produces two different ebml files, one is for Turkish and one is for English. These files includes only the details that are necessary for them, namely Turkish page does not include details for English page. After this fork operation, pages are deployed to the correct places in the server according to their languages. While a user connects from ERE, it chooses the language that he wants to use. After that, when he makes a menu request, the pages in his selected languages are brought from server. If page does not exist in that language, the page in the default language of the server is searched. If it exists, it is served. If not, an error message is given.

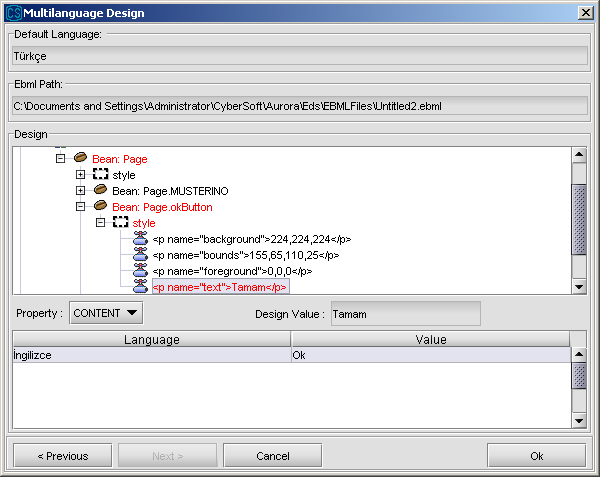
In the first page of Multilingual Design Wizard the available languages for EDS is shown. The languages that you want to use in your design should be selected in this page. After determining languages that you want to use, you can pass to second page by clicking Next button. Default language is not listed here.



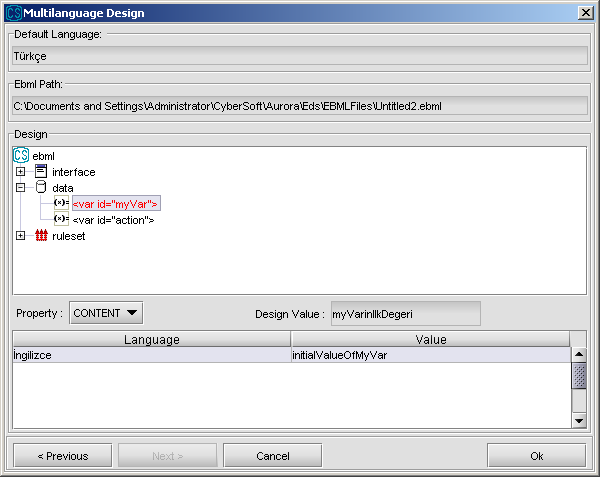
The items whose values can be multilingual, *the properties of beans, the initial values of variables, the contents of the comboboxes and properties of message rules*, are listed in the Multilingual Design dialog as a tree. The items and their ancestors up to some level, that are designed as multilingual is written in red while others in black. Therefore, by just looking at the tree, you may see whether there are multilingually designed beans under a node. However, as said above, only some ancestors has this property. The upmost level nodes, “interface”, “data” and “ruleset” always appears in black.



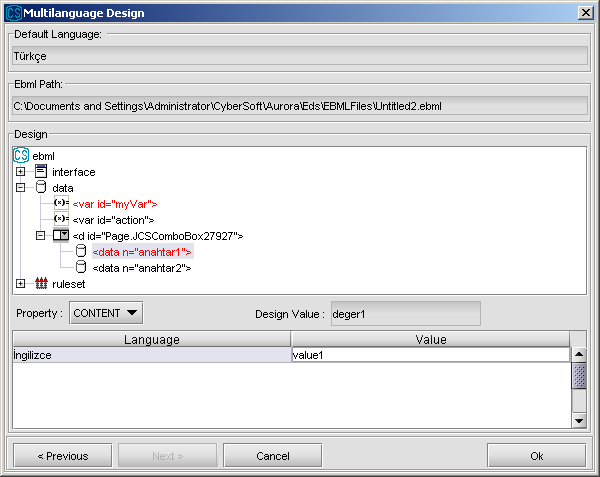
In order to change the property of a bean, the bean should firstly be found under the *interface* node. Beans are organized in a hierarchical manner. Namely if a component is in a container, it is found in the tree under the node belonging to that container. In the following example, we want to determine the properties of “Page.okButton”. It is placed in the main container “Page”, so it is found under it in the tree. When the “style” node is expanded, the properties that can be changed are also listed as tree nodes. In the example, “text” property of the button is determined as “Ok” for English, while its value is “Tamam” for default language, that isTurkish.



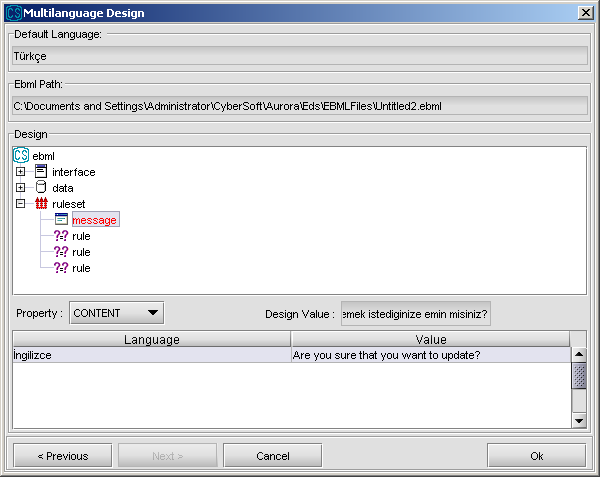
In order to change the initial value of a variable, the variable should be found under the “data” node. By selecting the variable from the tree, the initial value of the variable can be determined for every language other than default language. In the following example, the initial value of “myVar” is determined as “initialValueOfMyVar” for English, while it is “myVarinIlkDegeri” for the default language, that is Turkish.



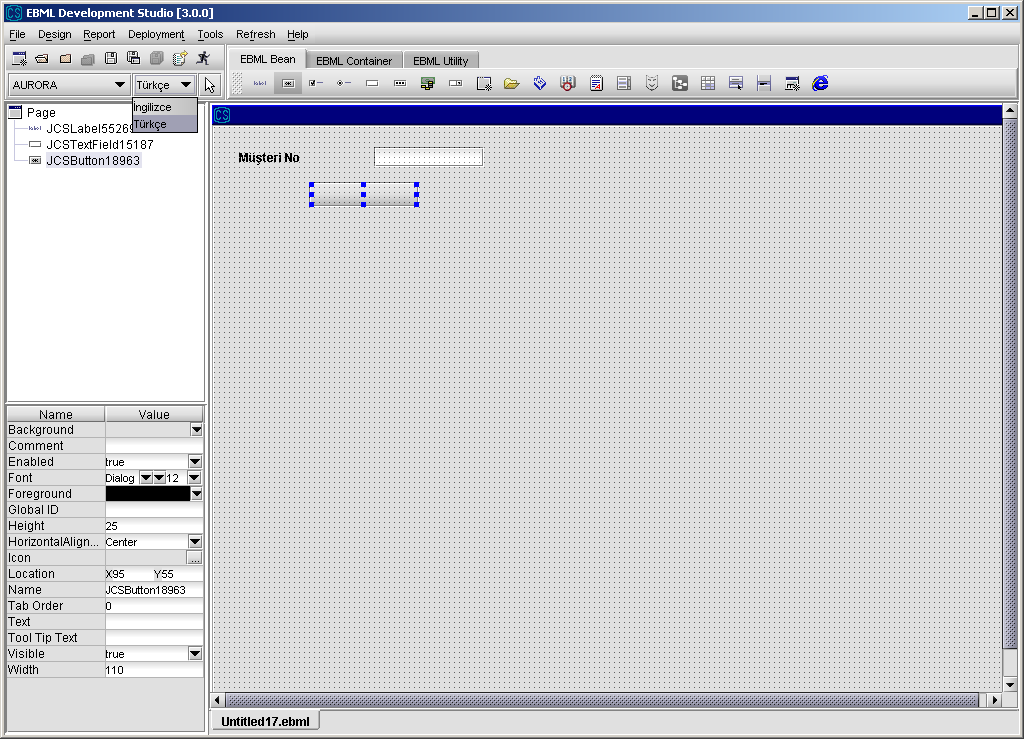
In order to change the content of the combobox, it should be found under the “data” node. The items added for combobox is listed under combobox node. In order to change the value of an item, select it and determine its value for languages that you want. In the following example, the combobox has two items and the value of first item whose key is “anahtar1” is determined as “value1” for English while the value of key is “deger1” for default language, that is Turkish. The value of second item is same for both Turkish and English.



In order to change the properties of message rules, it should be found under the “ruleset” node. Beside the content of the message, messageType and title properties of the message rule can also be changed. After the message is selected from the tree, the property that you want to change is selected from the Property combobox. For the following example, the content of the message is determined as “Are you sure that you want to update?” for English, while its value is “Guncellemek istediginize emin misiniz” for the default language, that is Turkish.



While multilingual pages can be tested by ERE, they can also be tested by EDS. When “Test” button is clicked from toolbar, EDS brings the page from CMS according to the selected language in the toolbar. Language selection combobox is just under the “Test” button. By that way, after that you design a page whose default language is Turkish, also in English multilingually, you can test how it will be seen for English content by selecting language as “English” and clicking “Test” button.



## 3.9 Utilities

### 

### 3.9.1 Tab Order

“Tab Order” property of each component should be set in right order to have a proper focus operation on the screen. However, in the case of tab orders are not determined by designer, a default tab order is given by ERE according to position of the beans. The strategy is simple; the above components comes in front in the tab order. For the components in the same level, the component in the left side comes in front. Focus cycle includes only visible components.

While giving Tab Order number, it is better to use numbers like 10, 20, 30,… so that if a new component is added later on, giving a new tab order number will not be a problem.

### 3.9.2 Miscellaneous Functions

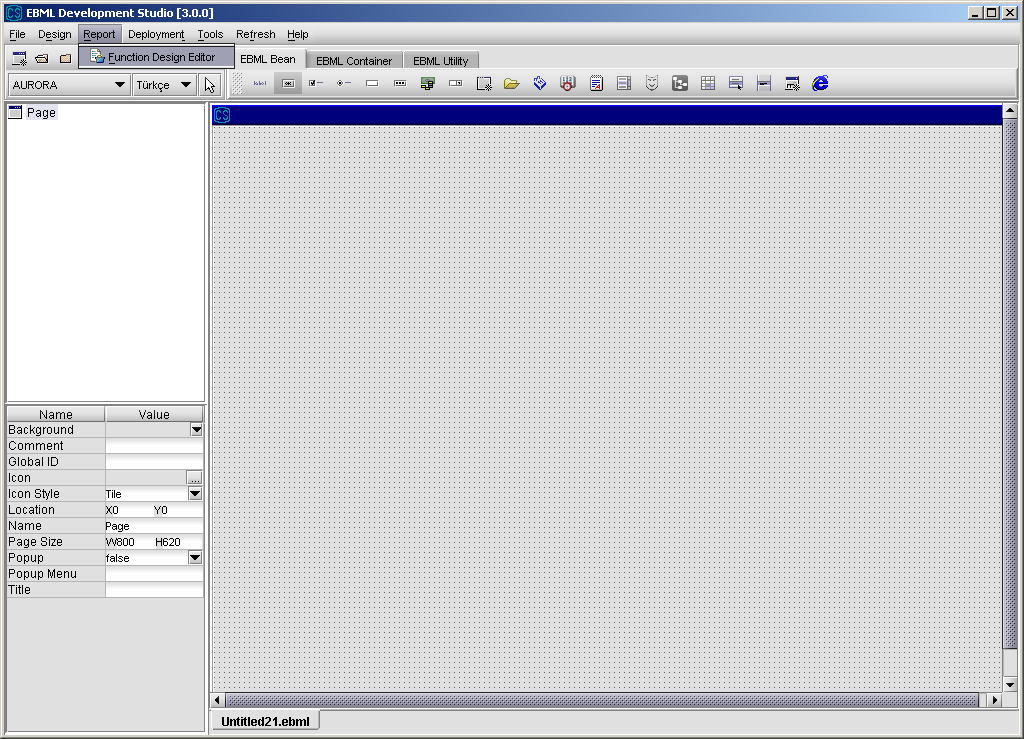
* EDS provides a bean called “Page” to perform “exit” and “giveUp” functions. With the “giveup” function call, page return to previous page. Using “exit” method, developer can provide his/her applications to exit.
* EDS provides “renaming” functionality for all of the beans in design environment. When a developer changes the name of the bean, all of the occurrences of that bean is replaced by the new name.
* In “Action Design Editor”, sub-actions can be moved up and down

# 4. Reporting Framework

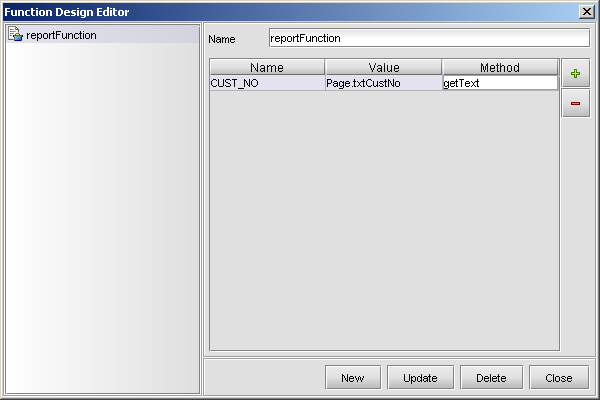
## 4.1 Designing Report Visually in EDS

Developer can design a report using EBML Development Studio – Design Mode.

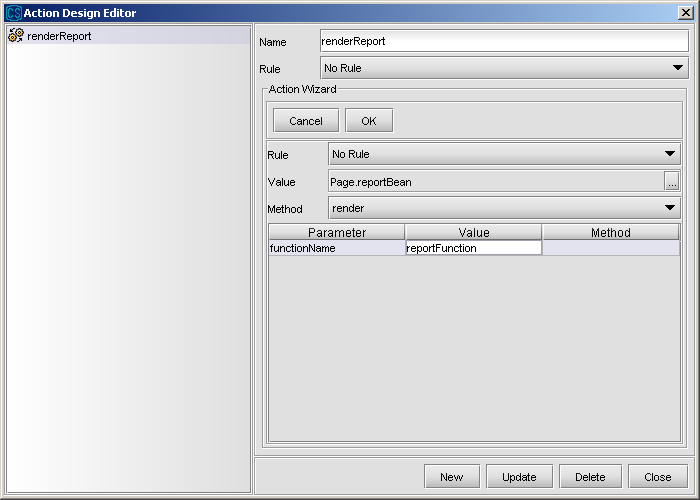
1. Select Function Design Wizard from Report menu item

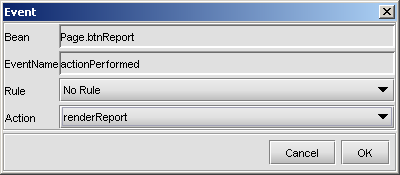


1. After running Method Design Wizard the following screen is displayed. In Method Design part, the parameters that will be used in report SQL file are mapped to the components, variables on the page or constants.



1. Now an action to render or print the report should be defined. In this action, we will define a bean action. In this action, we will call the related method of the report.

****4. Finally, we should associate this action with an event. Assume that we want to render the action when “Report” button is clicked.



## 4.2 Designing Report Template Using VSReport

Reports are rendered using VSReport ActiveX. The ActiveX is distributed along with the ERE. Report Design is accomplished through the design environment of the VSReport. The environment could be started using windows start menu. *(Start->Programs-> ComponentOne ActiveX Controls->VSView 7.0->Reporting Edition->Report Designer).*

The following point should be taken into account while designing a report:

* Each report file should contain only one report
* The report name should be same as the report file name except the “.xml” extension
* The runtime report query file name should be the same with the report name except that the query file should have “.sql” extension.
* The runtime report queries are two kinds: direct SQL queries and the service call queries.

**Direct SQL queries** have the following format:

<report type="sql" columns="3" column1="NAME" column2="AGE" column3="SEX">

<sql dataSource="AuroraDS">

SELECT \* FROM CUSTOMER WHERE NAME LIKE ?

</sql>

<parameters>

<p prefix="" suffix="%" type="string">NAME</p>

</parameters>

</report>

Here columns attribute specifies the number of output columns from the query. Each column name should be specified as column# attributes and they should be the same as column name used in the report design. The query (designed as a prepared statement but execution is different) that will prepare the report and the data source that query will be executed is written in sql tag. The values that will replace question marks are defined in parameters tag.

Parameter definations have a syntax like below:

<parameter prefix=”” suffix=”%” type="string">NAME</parameter>

“prefix” and “suffix” attributes define prefix and suffix values that will be added to the specifed parameter during the execution of the query. “type” attribute defines whether the parameter will be enclosed with single quota or not. It can take three values; if it is stated as “string”, the parameter is enclosed with single quota, if it is stated as “numeric” or blank (or attribute “type” is not stated), the parameter isn’t enclosed with single quota.

In the example above, if the value of “NAME” is “S”; the value executed in the query will be “'S%'”. If the type attribute stated as “numeric” or left blank, the value executed int query will be “S%”.

**Service call queries** have the following format:

<report type="service" columns="3" column1="A" key1="KEY1" column2="B"

key2="KEY2" column3="C" key3="KEY3">

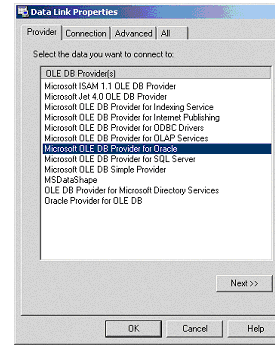
<service>SERVICE\_NAME</service>

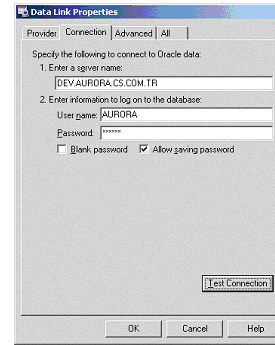
</report>

Here columns attribute specifies the number of output columns from the query. Each column name should be specified column# attributes and they should be the same as column name used in the report design. The report result set is obtained from the CORE\_DATA bag key with the column bag keys defined key# attributes for each column. The service that will prepare the report is given in service tag. Unlike direct sql queries, service call queries don’t have a parameters tag.

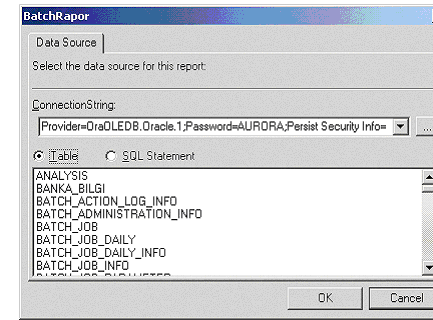
## 4.3 Sample Report Design

1. Run Report Designer



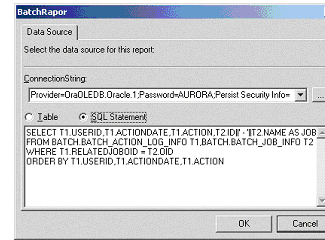


1. Set Data Link using the “ConnectionString” field
2. Tables connected to this data link are listed automaticaly



1. Prepare the report query without any parameters, choosing the “SQL Statement”

option



**Example Query:**

SELECT T1.USERID,T1.ACTIONDATE,T1.ACTION,T2.ID||' - '||T2.NAME AS

JOB,T2.DESCRIPTION

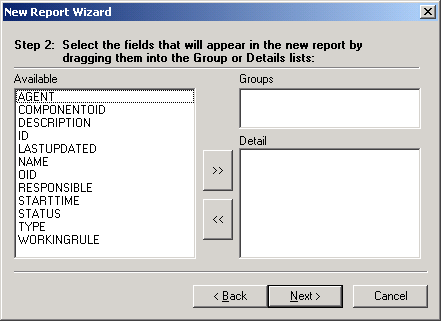
FROM BATCH.BATCH\_ACTION\_LOG\_INFO T1,BATCH.BATCH\_JOB\_INFO T2

WHERE T1.RELATEDJOBOID = T2.OID

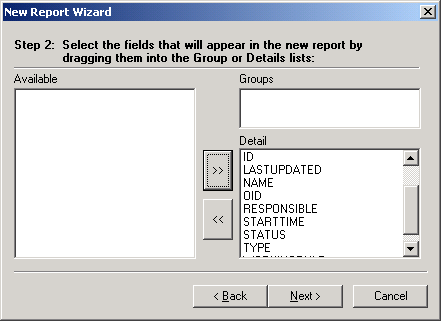
ORDER BY T1.USERID,T1.ACTIONDATE,T1.ACTION

1. Select the fields that will appear in the new report by dragging and dropping them

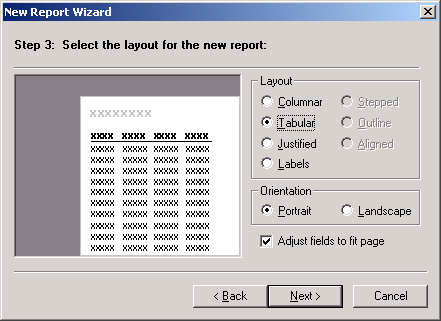
into the Group or Details list



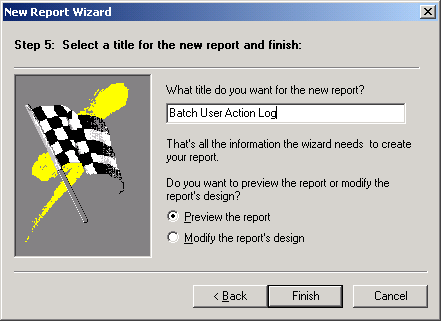
1. Selected fields are listed on Group or Details section



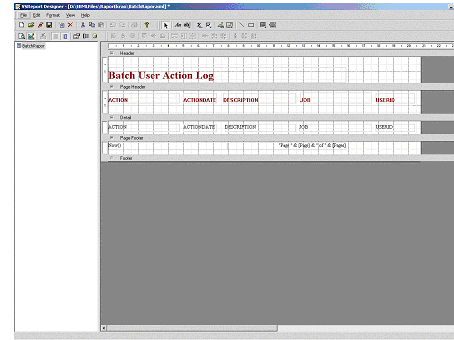
1. Choose type of the report from Layout section



1. Set the report title using Report Wizard

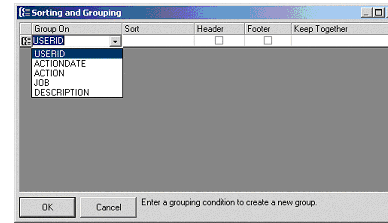


1. Using “Preview the report” option, Report Design Environment is viewed

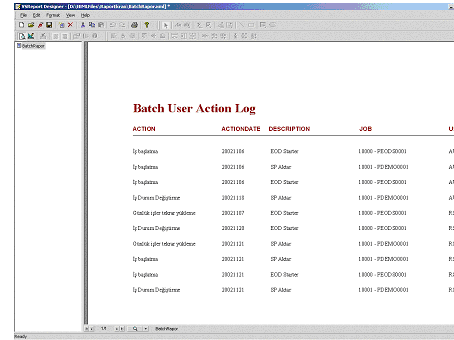


1. Add new fields and labels using Label and DataField buttons on the Report toolbar.

In order to add a new group, choose the Grouping from View.



1. Report is displayed in run-time environment by pressing the Preview button



1. Prepare the file “*reportName*.sql”. This file is prepared using number of columns

and column names in the order like in query. Parameters are added as defined in the popup.

<report type="sql" columns="5" column1="USERID" column2="ACTIONDATE" column3="ACTION" column4="JOB"

column5="DESCRIPTION" >

<sql dataSource="AuroraDS">SELECT T1.USERID,T1.ACTIONDATE,T1.ACTION,T2.ID||' - '||T2.NAME AS

JOB,T2.DESCRIPTION

FROM BATCH.BATCH\_ACTION\_LOG\_INFO T1,BATCH.BATCH\_JOB\_INFO T2

WHERE T1.RELATEDJOBOID = T2.OID AND T1.USERID LIKE ?

AND T1.ACTION LIKE ?

ORDER BY T1.USERID,T1.ACTIONDATE,T1.ACTION

</sql>

<parameters>

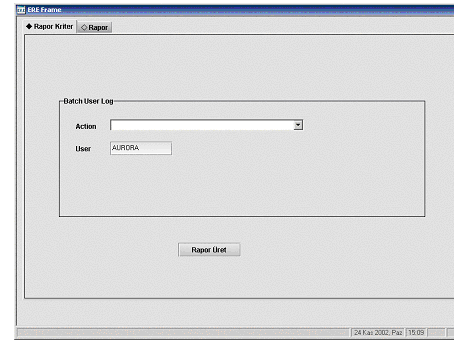
<p prefix="" suffix="%"> CORE\_USERNAME</p>

<p prefix="" suffix="%"> CORE\_ACTION</p>

</parameters>

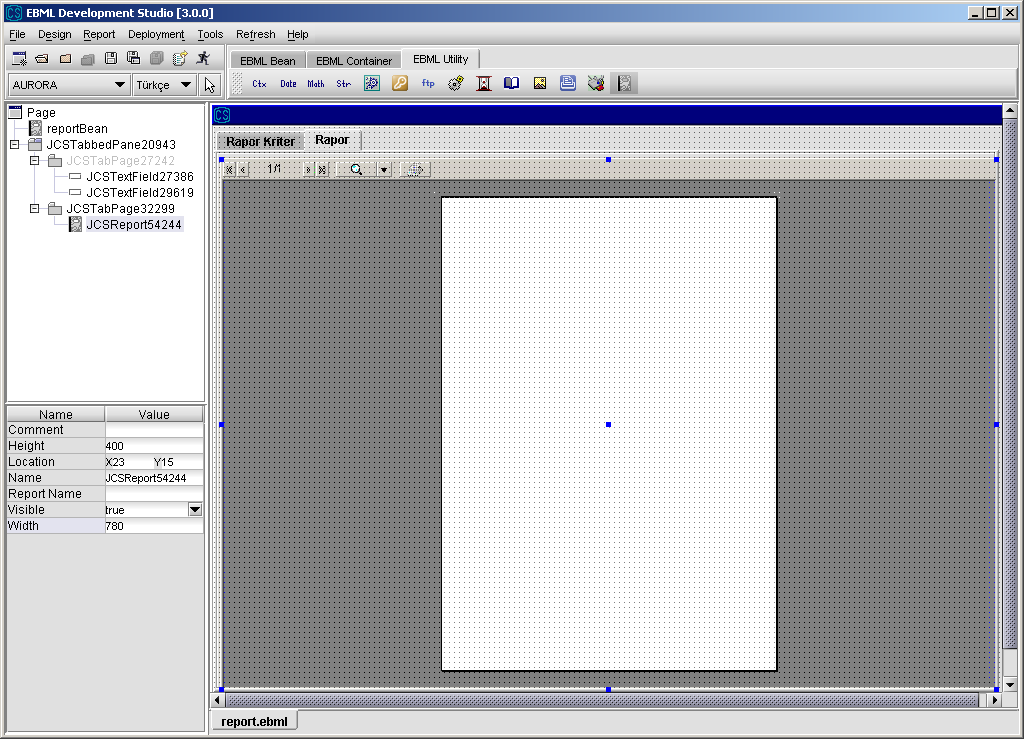
</report>

1. Run EDS environment and define the fields for the parameters defined previously



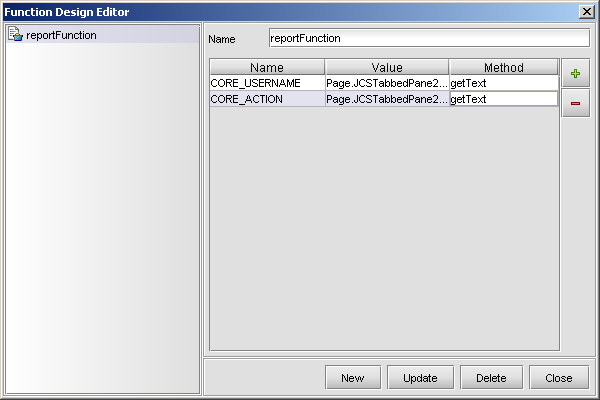
1. Drag and drop the report bean in EDS environment on a new page. Name the report

with the name given in Report Designer environment.

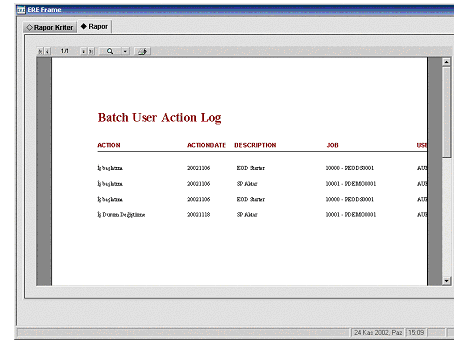


1. In EDS environment, add the parameters to send to the report by “Method Design

Wizard” from Design Wizard



1. Run the report to see the result



## 4.4 Sample Special Report (Statement of Account) Design

Special reports like “Statement of Account (Dekont)” are printed using JCSTextPrinter component.

1. Prepare document template in EDS Design Environment
2. Write an AURORA service for preparing the contents of “Statement of Account” document. This service will be used to construct a string as in the following sample text data. This string will be used to fill the document template with data.

**Example**: Following is a sample code of a service for preparing document’s contents. Service is named as “AURORA\_REPORTWIDGET\_SENDPARAMETER”. As it is seen from the following code, a bag having the data of the document is prepared and a **“format”** method of a special Utility class named as “**PrintFormatter”** is called with this bag.

**static** CSBag bag = CSBagFactory.createBag();

**public** **static** **void** main(String[] args) {

TextPrinter t = **new** TextPrinter();

**try** {

t.sendParameter(bag);

} **catch** (CSException e) {

}

}

**public** **static** CSBag sendParameter(CSBag inBag) **throws** CSException

{

ParameterFormatter pf = **new** ParameterFormatter();

CSBag outBag = CSBagFactory.createBag();

CSBag bag = CSBagFactory.createBag();

bag.put(0, "DEKONT");

bag.put(1, "TARIH");

bag.put(2, "10.01.2003");

bag.put(3, "SAAT");

bag.put(4, "16:40");

bag.put(5, "SUBE");

bag.put(6, "BOSTANCI");

bag.put(7, "HESAPNO");

bag.put(8, "1405466");

String parameter = pf.format(bag);

outBag.put(CORE\_PARAMETER,parameter);

**return** outBag;

}

This service returns a bag having string CORE\_PARAMETER which will be set to the “text” property of a TextPrinter component to print the document.

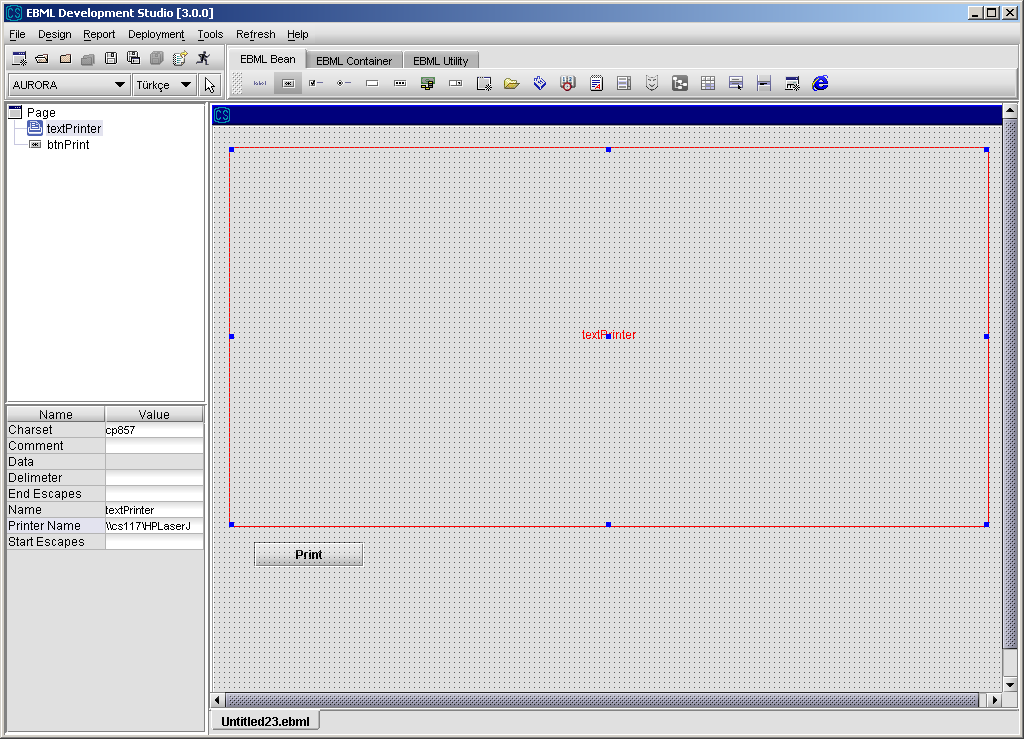
**Sample Text Data:**

DEKONT#TARIH#10.01.2003#SAAT#16:40#SUBE#BOSTANCI#HESAPNO#1405466#

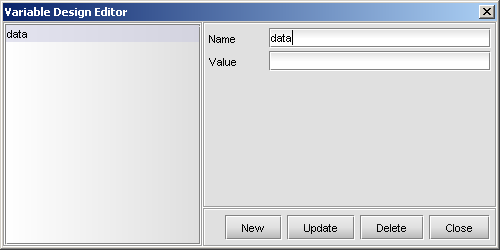
1. Design a new screen in EDS Design Environment that will be used to print the document. JCSTextPrinter component will be used to print the document’s content.

With the text data above, JCSTextPrinter’s “printFormatted” method will write “10.01.2003” as “TARIH” field, “16:40” as “SAAT” field in the template document.

* 1. Select JCSTextPrinter component from toolbar and drop into your page and set properties of JCSTextPrinter component



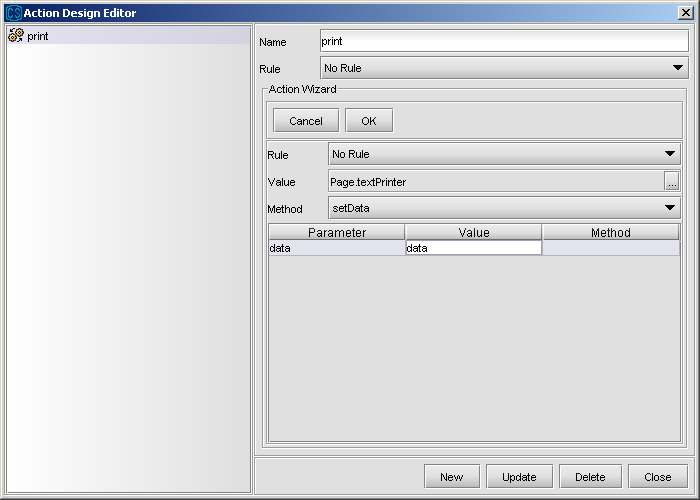
* 1. Design a new variable to get the result of the service “AURORA\_REPORTWIDGET\_SENDPARAMETER”.



* 1. Create an action to call the service “AURORA\_REPORTWIDGET\_SENDPARAMETER”

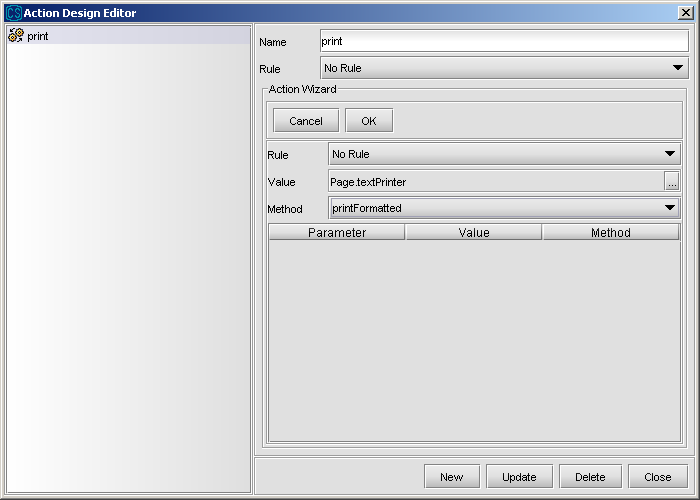
and assign the returned string from service to the variable you designed before.

* 1. Create a new bean action to set JCSTextPrinter’s data property to the value of VARIABLE:data.



* 1. Create another bean action to call JCSTextPrinter’s “printFormatted” method with. JCSTextPrinter’s “printFormatted” method prints the data set by its “data” property.

Since document is a special formatted one, JCSTextPrinter’s “printFormatted” method will be used to print the data.

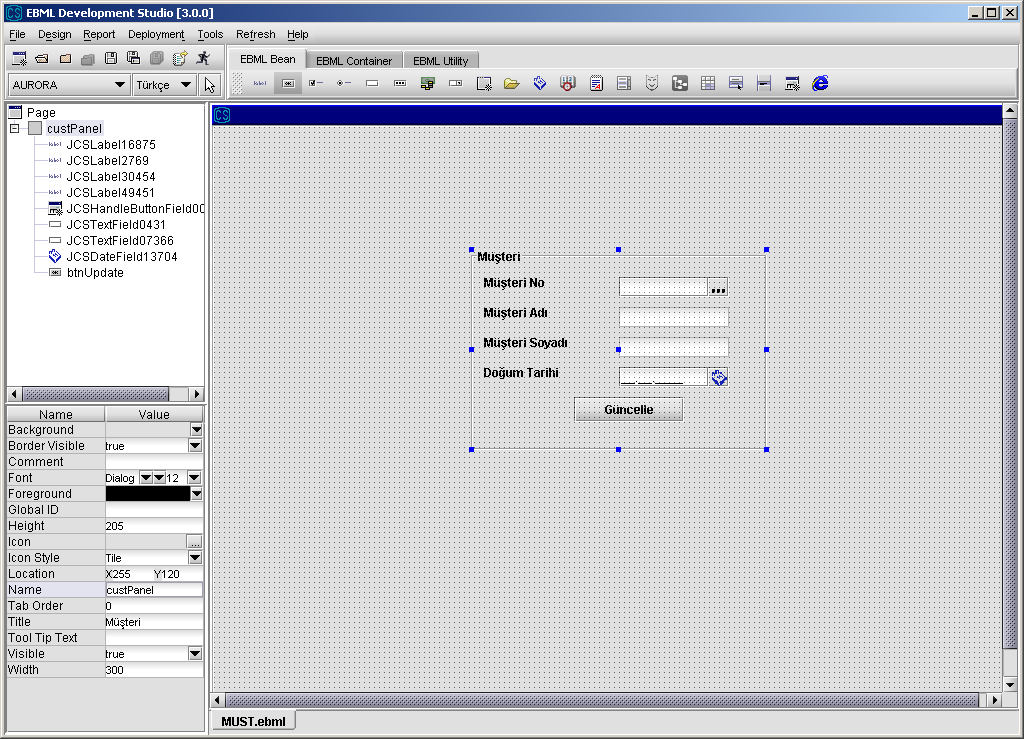


1. Deploy the screen you designed and test it by running “Test With CMS”.

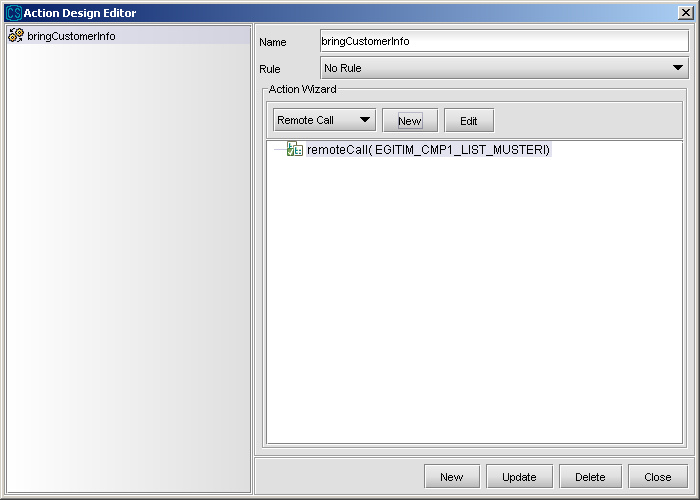
# 5.Sample Screen Design

In this section, we will design a simple screen which will try to show most aspects of the EDS. In this screen, we query a customer via a handle button field. Some information about the customer will be displayed on the screen. The user will be able to update the birth date of the customer via “Güncelle” button.

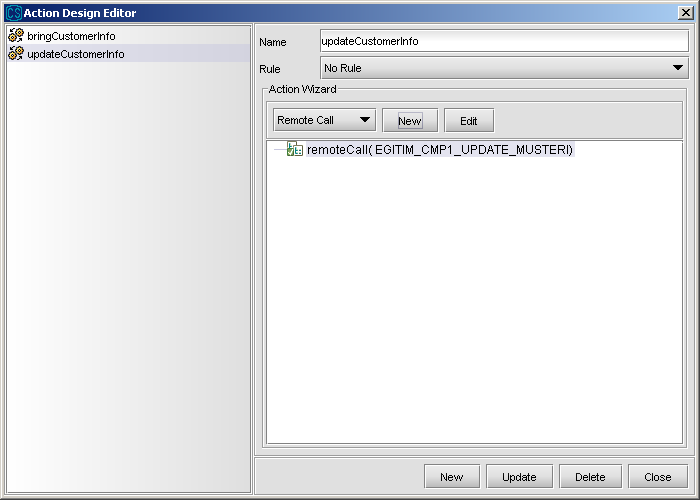
Step 1: The screen is designed using necessary beans in EDS and meaningful names are given to beans.



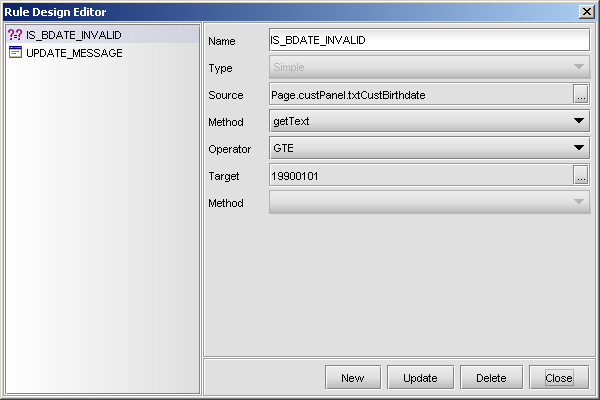
Step 2: We will define necessary actions to execute. We will bring the customer name and surname from popup, however since popup does not include birth date info, we should get it via a remote call. We call “EGITIM\_CMP1\_LIST\_MUSTERI” service, which is previously defined. It gets “CUST\_NO” bag key as input and gives “CUST\_BIRTHDATE” key as output. We mapped the “Customer No” field to “CUST\_NO” and “Customer Birthdate” field to “CUST\_BIRTHDATE”.

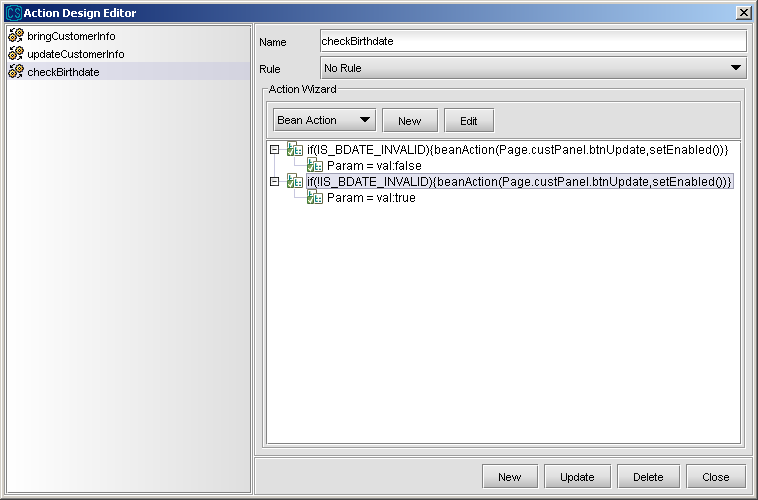


Step 3: We need an action to update the customer information when “Güncelle” button is clicked. This action also makes a remote call.

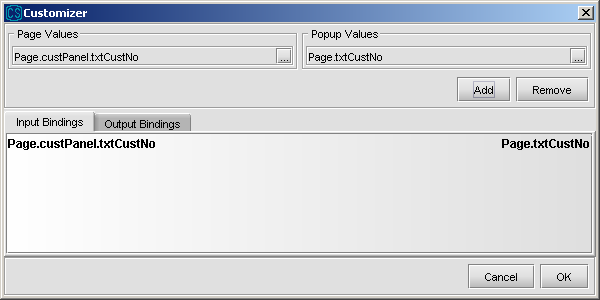


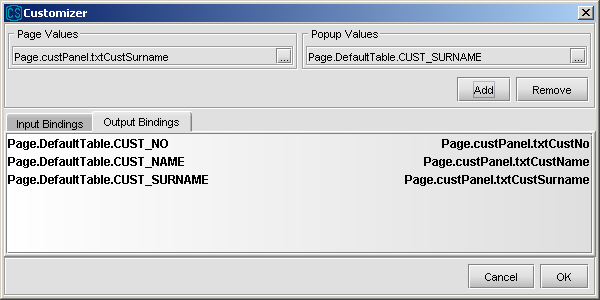
Step 4: We also want to make a control about the birthdate of the customer. We assume that the previously saved birthdates are valid. If user brings a customer info and tries to change the birthdate we will check whether the new birthdate is larger than or equal to “1/1/1990”. We will not accept customers who born later than this date and disable the “Güncelle” button. In order to do this, we first define a rule for this check and then define an action using this rule.



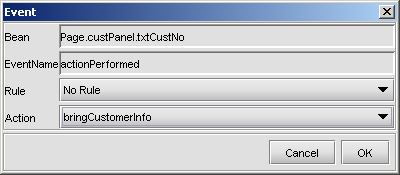


Step 5: Now we can make the mappings for handle button field. We want to transfer the value of “Customer No” field from page to popup and after an item is selected in the popup we will transfer Customer No, Name and Surname fields from the default table of popup to page.

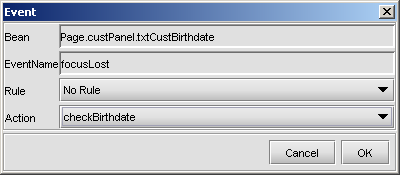




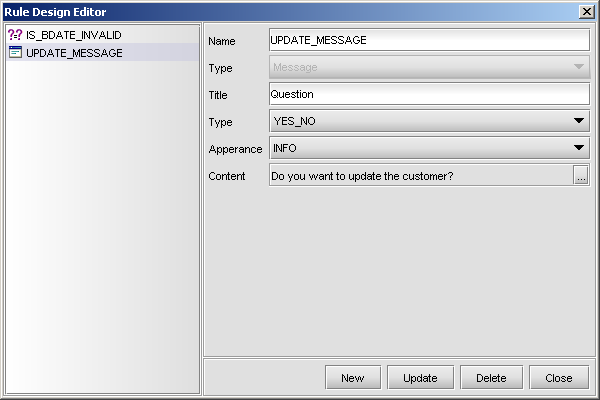
Step 6: When popup is activated, we should also call the bringCustomerInfo action in order to get birth date info.

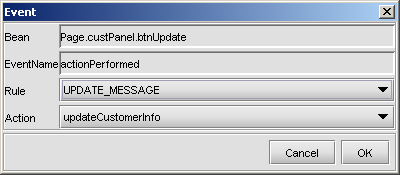


Step 7: If user enters the birth date field, we want to make the check when he leaves the field. For that purpose, we will associate the “checkBirthdate” action with the “focusLost” event of “Customer Birth Date” field.



Step 8: Now, we can associate the “actionPerformed” event of “Güncelle” button with the “updateCustomerInfo” action. However, before doing that, we want to show a confirmation message. For that purpose, we first define that message as a rule. In that case, user will be asked whether he is sure about updating. If he clicks “Yes”, update action will be called, otherwise nothing will be done.





# 6.Authentication &Authorization

AURORA Security Management System is based on “domain” concept, which is basically the environment where the system is operating. It defines the transactional behaviour of the system as well as the security mechanism integrated with this behaviour.

AURORA platform comes with a default domain called “AURORA”, which defines the basic transactional behaviour of the system and provides cyberSoft a maintenance base.

The domain-based approach has effects on runtime and design-time:

* **Runtime Effects :** Domain related variables can be used in the screens after the login. These variables are not being dynamically changed after the login. Their values are being used in the session construction and not changed until the next session creation.
* **Design-time Effects :** In the design time; the developer is provided with the domain specific variables. These variables can be used in all screens except the login screen where the session creation has not been performed yet.

A new defined domain should implement its own CSSession class. The implementation of domain’s CSSession class the following methods have to be implemented:

* **authenticate :** This method checks the users security credentials and returns success or failure as a result. It also creates the user session info on the server side.
* **authorize :** This method authenticates the user and generates the menu specified for that user (or the page after the login).

There is also another method, which is optional to implement:

* **userList :** This method returns the list of users in the new defined domain so that the basic “AURORA” domain security management system can be integrated with the new defined domain.

A sample implementatin of CSSession can be:

**import** java.util.Enumeration;

**import** tr.com.cs.aurora.auroracore.dictionary.bean.Domain;

**import** tr.com.cs.aurora.auroracore.dictionary.bean.User;

**import** tr.com.cs.aurora.auroracore.pom.BOMFactory;

**import** tr.com.cs.aurora.auroracore.pom.PomCriteria;

**import** tr.com.cs.aurora.auroracore.pom.PomFilter;

**import** tr.com.cs.aurora.auroracore.utility.CSBag;

**import** tr.com.cs.aurora.auroracore.utility.CSCaller;

**import** tr.com.cs.aurora.auroracore.utility.CSException;

**import** tr.com.cs.aurora.auroracore.utility.CSSession;

**public** **class** CSSessionAnadolu **extends** CSSession{

**public** CSSessionAnadolu() {

**super**();

}

**public** String getUserName() **throws** CSException{

**return** getSession().getSessionBag().get("CORE\_USERNAME").toString();

}

**public** CSBag authorize(CSBag iBag) **throws** CSException {

String result = "<interface>";

result += "<structure>";

result += "<bean class=\"cb.smg.fde.JCSPanelBase\" id=\"EDS.0\">";

result += "<style>";

result += "<p name=\"title\">AURORA</p>";

result += "<p name=\"bounds\">10,10,775,500</p>";

result += "</style>";

result += "<bean class=\"cb.smg.fde.bean.JCSMenuBar\" id=\"0.1\">";

result += "<style/>";

result += "<bean class=\"cb.smg.fde.bean.JCSMenu\" id=\"1.2\">"+

"<style>"+

"<p name=\"text\">Anadolu Sigorta</p>"+

"</style>";

result += "<bean class=\"cb.smg.fde.bean.JCSMenuItem\" id=\"2.1\">"+

"<style>"+

"<p name=\"text\">Mahsup Madde Girişi (TL)</p>"+

"<p name=\"page\">ANADOLU\_MAHSUP\_FIS\_GIRISI\_0</p>"+

"</style>";

result += "</bean>";

result += "</bean>";

result += "</bean>";

result += "<bean class=\"cb.smg.fde.bean.JCSLabel\" id=\"0.2\">";

result += "<style><p name=\"icon\">aurora.jpg</p>";

result += "<p name=\"bounds\">";

result += "93,33,589,435";

result += "</p>";

result += "</style></bean>";

result += "</bean>";

result += "</structure>";

result += "<events/>";

result += "</interface>";

result += "<data/>";

result += "<navigation/>";

result += "<ruleset/>";

CSBag oBag = **new** CSBag();

oBag.put("CORE\_MESSAGE",result);

oBag.put("CORE\_STATUS","MENU");

**return** oBag;

}

**public** CSBag authenticate(CSBag iBag) **throws** CSException {

String username = iBag.get("CORE\_USERNAME").toString();

String password = iBag.get("CORE\_PASSWORD").toString();

String message = "";

**boolean** loginState = **false**;

PomFilter filter = **new** PomFilter();

filter.put(**new** PomCriteria("USERNAME","=",username,1));

Enumeration enumeration = BOMFactory.select(**new** User(),filter);

User user = **null**;

**if** (enumeration.hasMoreElements()){

user = (User) enumeration.nextElement();

}

**if** (user != **null**) {

**if** (user.getPassword().equals(password)){

loginState = **true**;

}

}

CSBag oBag = **new** CSBag();

**if** (loginState) {

CSSession.getSession().getSessionBag().

put("CORE\_USEROID", user.getOID());

CSSession.getSession().getSessionBag().

put("CORE\_USERNAME", user.getUsername());

String globals = "<globals ";

globals +="username=\""+user.getUsername()+"\" ";

globals +="useroid=\""+user.getOID()+"\"/>";

oBag.put("CORE\_GLOBALS",globals);

oBag.put("CORE\_STATUS","MENU");

}**else** {

oBag.put("CORE\_MESSAGE","Login Failed....");

oBag.put("CORE\_STATUS","FAIL");

}

**return** oBag;

}

**public** **static** CSBag userList(CSBag iBag) **throws** CSException{

PomFilter filter = **new** PomFilter();

filter.put(**new** PomCriteria("NAME","=",

iBag.get("CORE\_DOMAINNAME").toString(),1));

Enumeration enumeration = BOMFactory.select(**new** Domain(),filter);

**if** (enumeration.hasMoreElements()){

Domain domain = (Domain) enumeration.nextElement();

iBag.put("CORE\_DOMAINOID",domain.getOID());

}

CSBag oBag = **new** CSBag();

CSBag sBag = CSCaller.call("AURORA\_DICT\_USER\_LIST\_FOR\_TABLE",iBag);

**int** rowCount = sBag.getSize("CORE\_TABLE");

**for** (**int** i = 0; i < rowCount; i++) {

oBag.put("CORE\_TABLE",i,"CORE\_MEMBERID",

sBag.get("CORE\_TABLE",i,"CORE\_USERNAME").toString());

oBag.put("CORE\_TABLE",i,"CORE\_FULLNAME",

sBag.get("CORE\_TABLE",i,"CORE\_FIRSTNAME").toString()+" "+

sBag.get("CORE\_TABLE",i,"CORE\_LASTNAME").toString());

}

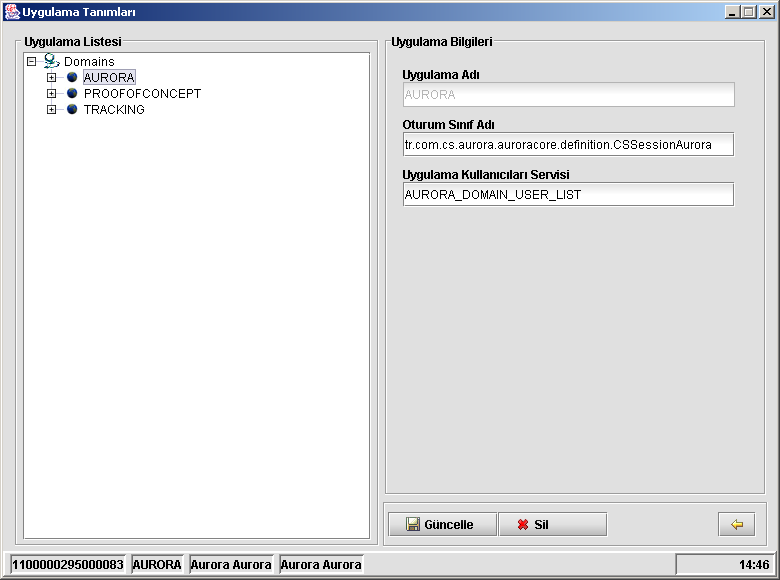
**return** oBag;

}

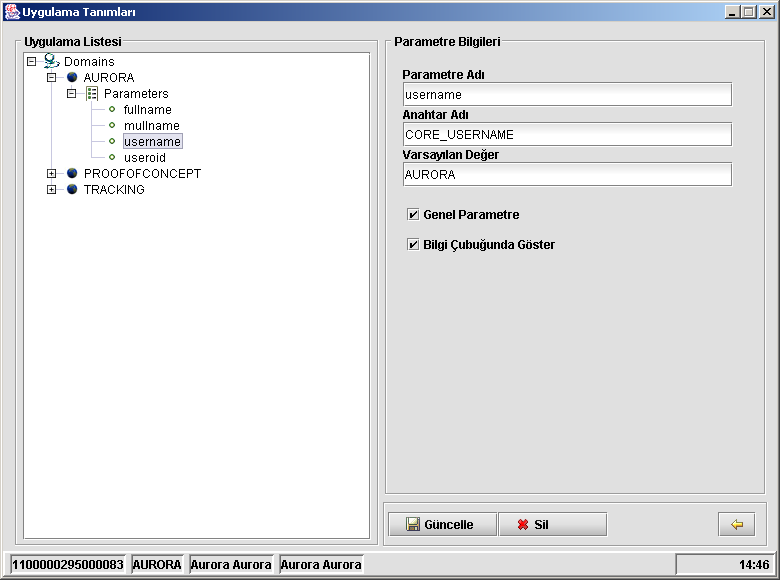
}

After implementing the CSSession class of the domain, domain defination must be implemented on the server side using domain definition screens.

In the first stage of domain definition, the name of the domain, its base CSSession class implementation, and the name of the user list provider service is defined.

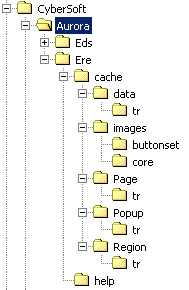


In the second stage of domain definition, the runtime (and design-time) variables for the domain are defined.



# 7.Caching In Ere

The data that comes from CMS to ERE can be categorized as dynamic which change very frequently and static which tends to change less frequently. In order to provide a faster response to the client, ERE caches the data that fall into second category. These data types are; reference datas, ebml and image files. These data are kept under the file structure of client according to their types and languages. This structure is shown below.



This structure is placed under the home directory of the user. Eds directory, which is not expanded above, exists if EDS is setup on the machine.

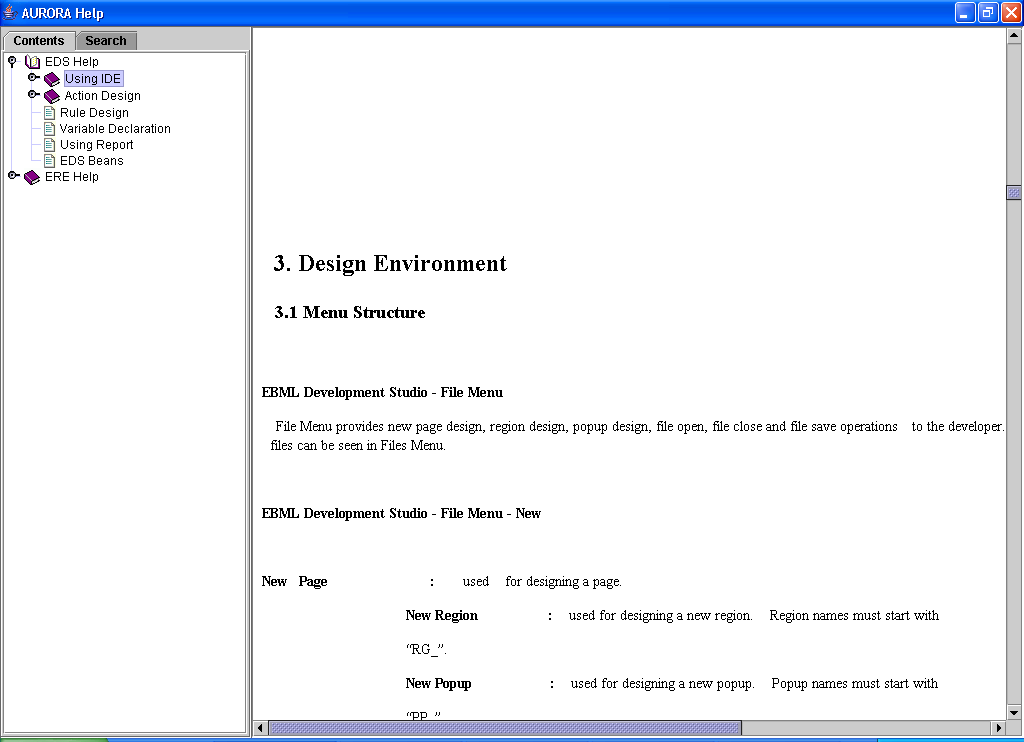
Reference data information has an XML format which has an attribute named “v” for version information of reference data at the root element of XML. When a client requests for a reference data, it puts a version information in the request (this should be the version information of the refence data it has or simply “” if it has no reference data). When CMS gets this request, CMS either returns a newer version of reference data if a newer version exists or returns “no change” if the version sent is the latest version. In the case of getting a newer version, ERE replaces the cached referenced data with the newer version.

EBML files, namely, pages, popups and regions have a similar approach. EBMl files that describes them are cached in the client. Version checking is made again using the value of “v” tag that is found in the root element of the EBML file.

Image files have a similar approach like EBML files. Only difference is there is no version information for image files, so last updated information is used instead of version information.

# 8.Using Help In Ere

While using ERE it is possible to reach help content about ERE by pressing “F1” button.



Content is build from an xml help definition file in CMS. Whenever F1 key is pressed, every time a command is sent to CMS to get xml file to build help tree. So, every new help version is accessible for every client. An example of help definition file is shown below.

<help title = "AURORA Help Title">

<base mainPage="AURORA\_FrontEnd\_v302.html" url="http://localhost"/>

<error notFound="error-not-found.html" invalidUrl="error-invalid-url.html"/>

<menu>

<item text="EDS Help" url="index.html">

<item text="Using IDE" url="guide/tutorial.html">

<item text="Script Yazma" url="guide/tutorial.html"/>

</item>

<item text="Action Design" url="dd">

<item text="1.2.1" url="guide/userguide.html"/>

<item text="Müşteri Kayit" url="guide/tutorial.html"/>

</item>

<item text="Rule Design" url=""/>

<item text="Variable Declaration" url=""/>

<item text="Using Report" url=""/>

<item text="EDS Beans" url=""/>

</item>

<item text="ERE Help" url="">

<item text="Getting Help" url="2.1.html">

<item text="Migration" url="2.1.1.html"/>

<item text="Testing" url="2.1.2.html"/>

</item>

</item>

</menu>

</help>

As seen, every help tree node is an xml item.

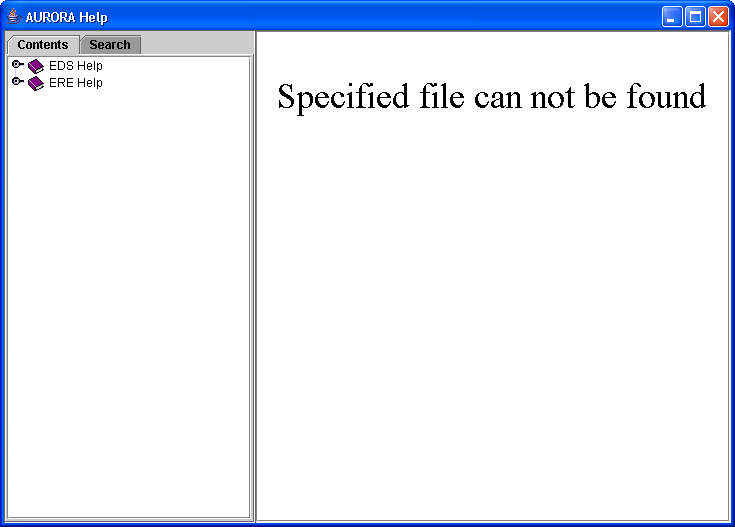
**<item text = “Display value” url=”Which page does this item call”>**

Every item can contain several items.

Every xml file has a root element, “help”. “title” property refers to help title. This element has 3 sub-elements.

* base : This element refers host address and protocol where help files will be requested. (“mainPage” property for future use)
* error : When no page is available or url is invalid, error pages can be defined in this element. “notFound” property defines html page when no page is available. “invalidUrl” is refers to the page when an invalid url exception raises.
* menu : This element is parent for all items.

To access a page relevant to cuurent EBML page, press F1 key. Suppose that current ebml file is TEST.EBML . If F1 key is pressed, ERE client requests TEST.HTML. If this file is available and any error is not returned, it is shown. Otherwise, a predefined error is shown.



Request is sent to server in this syntax.

{ protocol }/{ language } / { currentpage.html }

http protocol examples

<http://localhost/tr/turkish.html>

<http://10.10.10.10/en/english.html>

file protocol examples

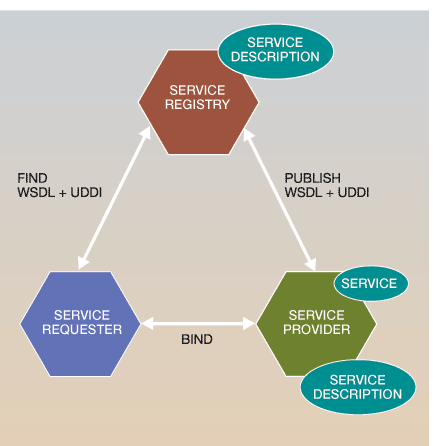
file:/C:/tr/Turkish.html

file:/C:/en/English.html

# 9.Web Services

A web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging.

A web service is an interface This architecture sets forth three roles and three operations. The three roles are the service provider, the service requester, and the service registry. The objects acted upon are the service and the service description, and the operations performed by the actors on these objects are publish, find, and bind.



A service provider creates a Web service and its service definition and then publishes the service with a service registry based on a standard called the Universal Description, Discovery, and Integration (UDDI) specification.

Once a Web service is published, a service requester may find the service via the UDDI interface. The UDDI registry provides the service requester with a WSDL service description and a URL (uniform resource locator) pointing to the service itself. The service requester may then use this information to directly bind to the service and invoke it.

## 9.1 Using Aurora Services As Web Services

### 9.1.1 Introduction to WSDL

WSDL stands for Web Services Description Language. WSDL is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) of the service exposes.

A WSDL document defines a web service using these abstract major elements:

|  |  |
| --- | --- |
| **Element** | **Defines** |
| <portType> | The operations performed by the web service |
| <message> | The messages used by the web service |
| <types> | The data types used by the web service |
| <binding> | The communication protocols used by the web service |

A WSDL document defines **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.

Hence, a WSDL document uses the following concrete elements in the definition of network services:

**Concrete Descriptions**

**Binding–** a concrete protocol and data format specification for a particular port type. Specifies binding(s) of each operation in the PortTypes section.

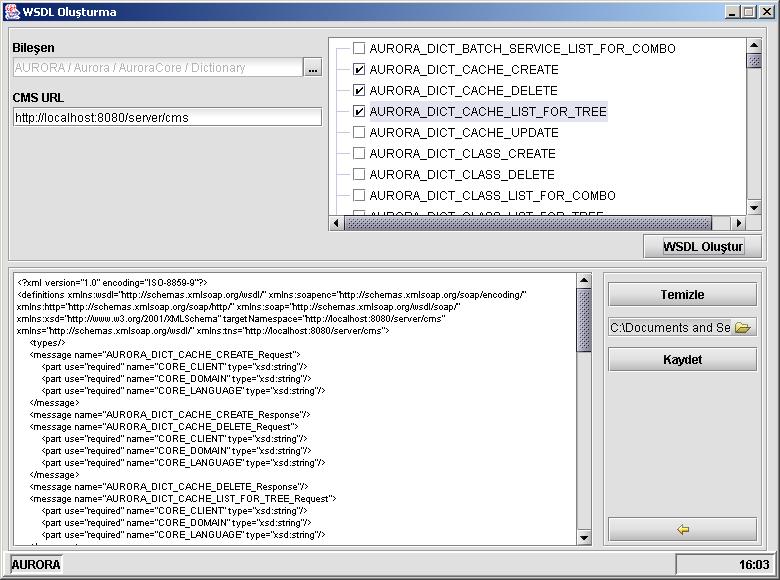
**Port–** a single endpoint defined as a combination of a binding and a network address.

**Service–** a collection of related endpoints. Specifies port address(es) of each binding.

### 9.1.2 WSDL Generation from Dictionary Service Definitions

For AURORA services, WSDL files will be generated based on components.

Following is a “WSDL Creation” screen used to generate WSDL files for selected AURORA services.



1. Project, component groups and component should be selected from the popup that will appear when the handle button at top-right of the screen is clicked.

1. From tree next, services to be defined as web services are selected by clicking the checkboxes that belong to them.
2. URL address for the content management server (CMS) is defined.
3. Selected services from the tree are sent to the service generating WSDL by pressing “Create WSDL” button.
4. In WSDL Generation service; for each service, service parameters are collected using Dictionary’s related parameter listing service.
5. Service parameters are grouped as INPUTS and OUTPUTS and added to the related messages parts of the WSDL.
6. If service parameter type is complex type like LIST or TABLE then the same Dictionary service for parameter collection is called again, this time to find LIST or TABLE contents.
7. If parameter type is LIST or TABLE, then array types are defined in WSDL’s definitions part . For LIST types one dimensional array is defined and for TABLE types two dimensional arrays is defined.

For example, for a string type array, the folllowing statement is added to the WSDL definitions part.

<complexType name="ArrayOfString">

<complexContent>

<restriction base="SOAP-ENC:Array">

<attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="String[]"/>

</restriction>

</complexContent>

</complexType>

and in message part, variable is defined using the Array complex type

<message name="HESAP\_HESAPBILGILERIGETIR\_Response">

…

<part name="HESAPYETKILISI" type="String"/>

<part name="MUSTERINO" type="tns:ArrayOfString"/>

</message>

1. WSDL file generated for the the selected services is shown in the screen. WSDL file can be saved on the screen by file selecting file and pressing “Save” button.
2. While generating WSDL file, some assumptions are made as detailed in section 2.3.

**Dictionary-WSDL fields mapping**

|  |  |  |
| --- | --- | --- |
| **WSDL Part** | **WSDL Attributes** | **Dictionary Fields Used** |
| <definitions> | xmlns:tns | CMS URL |
|  | targetNamespace | CMS URL |
| <types> | One and two dimensional array types and POM data types (CSCurrency, CSDate, CSTime) are defined |  |
| <message> | name | SERVICE NAME  parameters are grouped under  different message types like the following:   * inputs : SERVICE NAME\_Request * outputs : SERVICE NAME\_Response * exceptions : SERVICE NAME\_Fault |
| <*message* part> | name/type | PARAMETER NAME/ PARAMETER TYPE |
| <port> | name | COMPONENT NAME |
| <*port* operation> | name | SERVICE NAME |
| <*operation* documentation> | text | SERVICE DESC |
| <*operation* input> | message | WSDL input message.name |
| <*operation* output> | message | WSDL output message.name |
| <*operation* fault> | message | WSDL fault message.name |
| <binding> | name | COMPONENT NAME |
|  | type | WSDL portType.name |
| <*binding* soap:binding> | style | assumed as **“rpc”** |
|  | transport | assumed as **“...http”** |
| <*binding* operation> | name | SERVICE NAME |
| <*binding operation* soap:operation> | soapAction | SERVICE NAME |
| <*binding operation input*  *soap :body*> | use | assumed as **“encoded”** |
|  | namespace | CMS URL |
| <*binding operation output*  *soap :body*> | use | assumed as **“encoded”** |
|  | namespace | CMS URL |
| <service> | name | COMPONENT NAME |
| <*service* port> | name | COMPONENT NAME |
|  | binding | WSDL binding.name |
| <*service port* soap.address> | location | CMS URL |

Following is a sample WSDL code to indicate Dictionary and WSDL field’s mapping

<?xml version="1.0"?>

<definitions targetNamespace="**CMS URL** " xmlns:tns="**CMS URL** " xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">

<types/>

<!--types used for one and two dimensional ARRAY types and POM data types like CSCurrency, CSDate, CSTime-->

<!--for INPUT PARAMETERS-->

<message name="**SERVICE NAME** +Request">

<part name="**PARAMETER NAME**" type="**PARAMETER TYPE**"/>

...

</message>

<!--for OUTPUT PARAMETERS-->

<message name="**SERVICE NAME**+\_Response">

<part name="**PARAMETER NAME**" type="**PARAMETER TYPE**"/>

...

</message>

...

</message>

<portType name=" **COMPONENT NAME**+Port ">

<operation name="**SERVICE NAME**">

<documentation>**SERVICE DESC**</documentation>

<input message="**SERVICE NAME**+\_Request“/>

<output message= **SERVICE NAME**+\_Request“/>

</operation>

</portType>

<binding name="**COMPONENT NAME**+\_SOAP\_BINDING" type="tns: **COMPONENT NAME**+\_PORT ">

<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

<operation name="**SERVICE NAME**">

<soap:operation soapAction=" **CMS URL**"/>

<input>

<soap:body use="encoded" namespace=" **CMS URL** "/>

</input>

<output>

<soap:body use="encoded" namespace=" **CMS URL** "/>

</output>

</operation>

...

</binding>

<service name="**COMPONENT NAME**+ \_SERVICE\_LIST ">

<port name="**COMPONENT NAME**+\_PORT" binding="**COMPONENT NAME**+\_SOAP\_BINDING “>

<soap:address location="**CMS URL** "/>

</port>

</service>

</definitions>

### 9.1.3 Assumptions Made in WSDL for AURORA Services

There are some assumptions and naming standards made while generating WSDL for AURORA services.

***assumptions :***

* as a binding method, soapBinding is assumed to be used
* as a transport protocol, http transport protocol usage is assumed
* as a soap binding style, “rpc” is assumed to be used

naming standards:

* targetNamespace = CMS URL
* message name = SERVICE NAME + "\_Request”

message name = SERVICE NAME + "\_Response”

* <portType name = COMPONENT NAME + "PORT”>
* binding name = COMPONENT NAME + "SoapBinding”
* <soap:binding style = "rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
* <soap:operation soapAction = CMS URL + SERVICENAME />
* <service name = COMPONENT NAME + "ServiceList”
* <soap:adress location = CMS URL/>

### 9.1.4 Sample Usage

Here are examples just to indicate AURORA service definition in Dictionary and a WSDL generated from these definition.

**AURORA Dictionary Service Definition**

Service Name : AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE

Service Description : “Service to list a component's services within a tree”

Input : CORE\_COMPONENTOID

Type : String

Output : CORE\_TREE

Type : Table

This service is used in the WSDL example in following section.

### 9.1.5 WSDL Sample for an AURORA Web Service

Following is an WSDL file for service called AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE described in section 9.1.4

<?xml version="1.0" encoding="ISO-8859-9"?>

<definitions xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns=<http://schemas.xmlsoap.org/wsdl/>

xmlns:wsdl=<http://schemas.xmlsoap.org/wsdl/>

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:tns="http://localhost:8080/server/cms"

targetNamespace="http://localhost:8080/server/cms"

xmlns:http="http://schemas.xmlsoap.org/soap/http/"

xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">

<types>

<xsd:schema targetNamespace="http://localhost:8080/server/cms" xmlns="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="ArrayOfCORE\_TREE">

<xsd:complexContent>

<xsd:restriction base="SOAP-ENC:Array">

<xsd:attribute wsdl:arrayType="CORE\_TREE[]" ref="SOAP-ENC:arrayType"/>

</xsd:restriction>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CORE\_TREE">

<xsd:all>

<element name="CORE\_OID" type="xsd:string"/>

<element name="CORE\_NAME" type="xsd:string"/>

<element name="CORE\_DESCRIPTION" type="xsd:string"/>

</xsd:all>

</xsd:complexType>

</xsd:schema>

</types>

<message name="AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE\_Request">

<part use="required" name="CORE\_CLIENT" type="xsd:string"/>

<part use="required" name="CORE\_DOMAIN" type="xsd:string"/>

<part use="required" name="CORE\_LANGUAGE" type="xsd:string"/>

<part use="optional" name="CORE\_COMPONENTOID" type="xsd:string"/>

</message>

<message name="AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE\_Response">

<part use="optional" name="CORE\_TREE" type="tns:ArrayOfCORE\_TREE"/>

</message>

<portType name="Dictionary\_PORT">

<operation name="AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE">

<documentation>Service to list a component&apos;s services within a tree</documentation>

<input message="tns:AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE\_Request"/>

<output message="tns:AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE\_Response"/>

</operation>

</portType>

<binding name="Dictionary\_SOAP\_BINDING " type="tns:Dictionary\_PORT">

<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http/"/>

<operation name="AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE">

<soap:operation style="rpc" soapAction="http://localhost:8080/server/cms/AURORA\_DICT\_COMPONENT\_SERVICE\_LIST\_FOR\_TREE"/>

<input>

<soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/server/cms"/>

</input>

<output>

<soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/server/cms"/>

</output>

</operation>

</binding>

<service name="Dictionary\_SERVICE\_LIST">

<port name="Dictionary\_PORT" binding="tns:Dictionary\_SOAP\_BINDING">

<soap:address location="http://localhost:8080/server/cms"/>

</port>

</service>

</definitions>

## 9.2 Using Outside Web Services

### 9.2.1 Calling Web Servıce

Calling an outside web service is similar to calling a “regular” AURORA service. Like “CSCaller.call”, a special public static method is used to call web services. This method is “WebServiceCaller.callWebService” with structure :

CSBag callWebService(String serviceName,String operationName,String wsdl,CSBag inBag)

If we look closer to the arguments of the method :

* Name of the service specified in the related WSDL file (serviceName)
* Name of the operation specified in the related WSDL file (operationName)
* Content of the related WSDL file (wsdl)
* A CSBag object containing parameters for web service request (inBag)

And the return value of this method will be a CSBag object containing the prameters in the SOAP response message.

### 9.2.2 WSDL And Bag Key Name Mapping

In a WSDL file, all information to call service and to get response is specified. One of the important parts of this information is the structure of data to sent to the web service.

This data structure is specified in “message” and “types” (which is optional) parts of the WSDL file. In order to let the web service to work properly (or work in most cases) the data sent must be prepared as it is described in the WSDL file.

The web service calling mechanism in AURORA examines these parts when a web service is called and gets the structure of input parameters of the web service. After that it tries to create SOAP message.

To put input parameters in SOAP message, the mechanism must know which field in the input bag corresponds which field in the input parameters of the web service.

The best way to do this is while placing parameters in the bag, associate them with the Bag Key names which are same as the names of the input parameters of the web service.

There are two type of type declerations in WSDL files : simple and complex types. Simple types, as its name suggests, contain basic types like string etc. On the other hand, complex types contain type declerations consist of simple types.

If a parameter is defined in WSDL file, as a simple type like :

<part name="*parameter\_name*" type=" *simple\_parameter\_type*"/>

or

<element name="*parameter\_name*" type=" *simple\_parameter\_type*"/>

It should be behaved as a 0-dimensional bag structure like :

Bag.put(*parameter\_name*,*parameter\_value*)

If a parameter is defined in WSDL file, as a complex type like :

<complexType name="*main\_parameter\_name*">

<all>

<element name="*sub\_parameter\_name\_1*" type="*simple\_parameter\_type*"/>

<element name="*sub\_parameter\_name\_2*" type=" *simple\_parameter\_type*"/>

<element name="*sub\_parameter\_name\_3*" type=" *simple\_parameter\_type*"/>

</all>

</complexType>

It should be behaved as a 2-dimensional bag structure like :

Bag.put(*main\_parameter\_name*,i,*sub\_parameter\_name\_1*,*parameter\_value*)

Bag.put(*main\_parameter\_name*,i,*sub\_parameter\_name\_2*,*parameter\_value*)

Bag.put(*main\_parameter\_name*,i,*sub\_parameter\_name\_3*,*parameter\_value*)

### 9.2.3 Data Type Mapping Between SOAP And Bag

Just like the structure of the request message, the structure of the response message of a web service is clearly defined in the WSDL file of that service. This structure is specified in the “message” and “types” (which is optional) parts of the WSDL file.

Web service calling mechanism examines the returning SOAP message, and parses it into a CSBag object.

Like WSDL-Bag mapping, SOAP Response-Bag mapping, parses the SOAP response, and places parameters into CSBag objects in two ways :

If the parameter is simple type, which means it is represented in the SOAP response message as :

<*parameter\_name*> *parameter\_value*</*parameter\_name*>

It will be placed into the result CSBag object as 0-dimensional Bag structure like :

Bag.put(*parameter\_name*,*parameter\_value*)

If the parameter is complex type, which means it is represented in the SOAP response message as :

<*main\_parameter\_name*>

<*item*>

<*sub\_parameter\_1*>*parameter\_value11*</*sub\_parameter\_1*>

<*sub\_parameter\_2*>*parameter\_value12*</*sub\_parameter\_2*>

</*item*>

<*item*>

<*sub\_parameter\_1*>*parameter\_value21*</*sub\_parameter\_1*>

<*sub\_parameter\_2*>*parameter\_value22*</*sub\_parameter\_2*>

</*item*>

</*main\_parameter\_name*>

It will be placed into the result CSBag object as 0-dimensional Bag structure like :

Bag.put(*main\_parameter\_name*,0, *sub\_parameter\_1*,*parameter\_value11*)

Bag.put(*main\_parameter\_name*,0, *sub\_parameter\_2*,*parameter\_value12*)

Bag.put(*main\_parameter\_name*,1, *sub\_parameter\_1*,*parameter\_value21*)

Bag.put(*main\_parameter\_name*,1, *sub\_parameter\_2*,*parameter\_value22*)

### 9.2.4 Resources

#### 9.2.4.1 Limitations

The following limitations are known : No transaction support in calling the web service

### 9.2.5 Sample Usages

#### 9.2.5.1 Search “AURORA” in Google

The following steps needs to be carried out for the search

1. Creating the input bag

CSBag inBag = **new** CSBag();

inBag.put("key","7l/R8/15TUDRx1DuiE0PsjYxo982XT8k");

inBag.put("q","AURORA");

inBag.put("start","0");

inBag.put("maxResults","2");

inBag.put("filter","1");

inBag.put("restrict","");

inBag.put("safeSearch","1");

inBag.put("lr","");

inBag.put("ie","");

inBag.put("oe","");

1. Calling callWebService

CSFile file = **new** CSFile("C:/GoogleSearch.wsdl");

CSBag outBag= WebServiceCaller.callWebService("GoogleSearchService","doGoogleSearch",file.readAsString(),inBag);

1. SOAP Request

<SOAP-ENV:Envelope xmlns:wsdl=<http://schemas.xmlsoap.org/wsdl/>

xmlns:SOAP-ENC=<http://schemas.xmlsoap.org/soap/encoding/> xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:typens="urn:GoogleSearch">

<SOAP-ENV:Body>

<m:doGoogleSearch

SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:m="urn:GoogleSearch">

<key>7l/R8/15TUDRx1DuiE0PsjYxo982XT8k</key>

<q>AURORA</q>

<start>0</start>

<maxResults>2</maxResults>

<filter>1</filter>

<restrict/>

<safeSearch>1</safeSearch>

<lr/>

<ie/>

<oe/>

</m:doGoogleSearch>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

1. SOAP Response

<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Body>

<ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<return xsi:type="ns1:GoogleSearchResult">

<directoryCategories xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[1]" xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/">

<item xsi:type="ns1:DirectoryCategory">

<fullViewableName xsi:type="xsd:string">Top/Science/Astronomy/Solar\_System/Sun/Aurora</fullViewableName>

<specialEncoding xsi:type="xsd:string"/>

</item>

</directoryCategories>

<documentFiltering xsi:type="xsd:boolean">false</documentFiltering>

<endIndex xsi:type="xsd:int">2</endIndex>

<estimateIsExact xsi:type="xsd:boolean">false</estimateIsExact>

<estimatedTotalResultsCount xsi:type="xsd:int">2670000</estimatedTotalResultsCount>

<resultElements ns3:arrayType="ns1:ResultElement[2]" xsi:type="ns3:Array" xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/">

<item xsi:type="ns1:ResultElement">

<URL xsi:type="xsd:string">http://www.geo.mtu.edu/weather/aurora/</URL>

<cachedSize xsi:type="xsd:string">22k</cachedSize>

<directoryCategory xsi:type="ns1:DirectoryCategory">

<fullViewableName xsi:type="xsd:string">Top/Science/Astronomy/Solar\_System/Sun/Aurora</fullViewableName>

<specialEncoding xsi:type="xsd:string"/>

</directoryCategory>

<directoryTitle xsi:type="xsd:string">The &lt;b&gt;Aurora&lt;/b&gt; Page</directoryTitle>

<hostName xsi:type="xsd:string"/>

<relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>

<snippet xsi:type="xsd:string">This is the &lt;b&gt;Aurora&lt;/b&gt; Page from Michigan Tech featuring &lt;b&gt;Aurora&lt;/b&gt; images and links&lt;br&gt; to other &lt;b&gt;Aurora&lt;/b&gt; Resources. &lt;b&gt;...&lt;/b&gt; Welcome to The &lt;b&gt;Aurora&lt;/b&gt; Page. &lt;b&gt;...&lt;/b&gt; </snippet>

<summary xsi:type="xsd:string">Gallery and links.</summary>

<title xsi:type="xsd:string">The &lt;b&gt;Aurora&lt;/b&gt; Page</title>

</item>

<item xsi:type="ns1:ResultElement">

<URL xsi:type="xsd:string">http://www.aurora.edu/</URL>

<cachedSize xsi:type="xsd:string">19k</cachedSize>

<directoryCategory xsi:type="ns1:DirectoryCategory">

<fullViewableName xsi:type="xsd:string">Top/Reference/Education/Colleges\_and\_Universities/North\_America/United\_States/Illinois/Aurora\_University</fullViewableName>

<specialEncoding xsi:type="xsd:string"/>

</directoryCategory>

<directoryTitle xsi:type="xsd:string">&lt;b&gt;Aurora&lt;/b&gt; University</directoryTitle>

<hostName xsi:type="xsd:string"/>

<relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>

<snippet xsi:type="xsd:string">&lt;b&gt;...&lt;/b&gt; Prospective Students Current Students Parents and Visitors Faculty and Staff Alumni&lt;br&gt; and Friends George Williams Campus of &lt;b&gt;Aurora&lt;/b&gt; University - Lake Geneva &lt;b&gt;...&lt;/b&gt; </snippet>

<summary xsi:type="xsd:string">An independent four-year institution with campuses in &lt;b&gt;Aurora&lt;/b&gt;, Illinois and Williams Bay, Wisconsin....</summary>

<title xsi:type="xsd:string">Welcome to &lt;b&gt;Aurora&lt;/b&gt; University!</title>

</item>

</resultElements>

<searchComments xsi:type="xsd:string"/>

<searchQuery xsi:type="xsd:string">AURORA</searchQuery>

<searchTime xsi:type="xsd:double">0.26622</searchTime>

<searchTips xsi:type="xsd:string"/>

<startIndex xsi:type="xsd:int">1</startIndex>

</return>

</ns1:doGoogleSearchResponse>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

1. Return value of callWebService method will be a CSBag object :

return[0][estimateIsExact]..............false

return[0][searchQuery]..................AURORA

return[0][startIndex]...................1

return[0][searchComments]...............

return[0][searchTips]...................

return[0][endIndex].....................2

return[0][resultElements]...............~resultElements~2~18~1~cachedSize~19k~1~hostName~~1~snippet~<b>...</b> Prospective Students Current Students Parents and Visitors Faculty and Staff Alumni<br> and Friends George Williams Campus of <b>Aurora</b> University - Lake Geneva <b>...</b> ~1~directoryCategory~%7EdirectoryCategory%7E2%7E2%7E0%7EspecialEncoding%7E%7E0%7EfullViewableName%7ETop/Reference/Education/Colleges\_and\_Universities/North\_America/United\_States/Illinois/Aurora\_University%7E~1~relatedInformationPresent~true~1~directoryTitle~<b>Aurora</b> University~1~summary~An independent four-year institution with campuses in <b>Aurora</b>, Illinois and Williams Bay, Wisconsin....~1~URL~http://www.aurora.edu/~1~title~Welcome to <b>Aurora</b> University!~0~cachedSize~22k~0~hostName~~0~snippet~This is the <b>Aurora</b> Page from Michigan Tech featuring <b>Aurora</b> images and links<br> to other <b>Aurora</b> Resources. <b>...</b> Welcome to The <b>Aurora</b> Page. <b>...</b> ~0~directoryCategory~%7EdirectoryCategory%7E2%7E2%7E0%7EspecialEncoding%7E%7E0%7EfullViewableName%7ETop/Science/Astronomy/Solar\_System/Sun/Aurora%7E~0~relatedInformationPresent~true~0~directoryTitle~The <b>Aurora</b> Page~0~summary~Gallery and links.~0~URL~http://www.geo.mtu.edu/weather/aurora/~0~title~The <b>Aurora</b> Page~

return[0][searchTime]...................0.26622

return[0][documentFiltering]............false

return[0][directoryCategories]..........~directoryCategories~2~2~0~specialEncoding~~0~fullViewableName~Top/Science/Astronomy/Solar\_System/Sun/Aurora~

return[0][estimatedTotalResultsCount]...2670000

The value of “return[0][resultElements]” is a CSBag object like :

resultElements[0][title]................The <b>Aurora</b> Page

resultElements[0][URL]..................http://www.geo.mtu.edu/weather/aurora/

resultElements[0][summary]..............Gallery and links.

resultElements[0][directoryTitle].......The <b>Aurora</b> Page

resultElements[0][relatedInformationPresent]....true

resultElements[0][directoryCategory]....~directoryCategory~2~2~0~specialEncoding~~0~fullViewableName~Top/Science/Astronomy/Solar\_System/Sun/Aurora~

resultElements[0][snippet]..............This is the <b>Aurora</b> Page from Michigan Tech featuring <b>Aurora</b> images and links<br> to other <b>Aurora</b> Resources. <b>...</b> Welcome to The <b>Aurora</b> Page. <b>...</b>

resultElements[0][hostName].............

resultElements[0][cachedSize]...........22k

resultElements[1][title]................Welcome to <b>Aurora</b> University!

resultElements[1][URL]..................http://www.aurora.edu/

resultElements[1][summary]..............An independent four-year institution with campuses in <b>Aurora</b>, Illinois and Williams Bay, Wisconsin....

resultElements[1][directoryTitle].......<b>Aurora</b> University

resultElements[1][relatedInformationPresent]....true

resultElements[1][directoryCategory]....~directoryCategory~2~2~0~specialEncoding~~0~fullViewableName~Top/Reference/Education/Colleges\_and\_Universities/North\_America/United\_States/Illinois/Aurora\_University~

resultElements[1][snippet]..............<b>...</b> Prospective Students Current Students Parents and Visitors Faculty and Staff Alumni<br> and Friends George Williams Campus of <b>Aurora</b> University - Lake Geneva <b>...</b>

resultElements[1][hostName].............

resultElements[1][cachedSize]...........19k

# 10. Preferences

We will examined preferences in two parts. One is user preferences that is relevant to client side. Other is project preferences that is about project design concept. Project preferences lets you define your system design constant. For instance, when you design in EDS, you want every text field’s foreground is set to blue or every text pane with a specific bounds.

## 10.1 User Preferences

User preferences manages user specific preferences such as Look&Fell,session-timeout,etc. This menu can be accessed by pressing “Alt-F1” key pairs. This option can be enabled/disabled using “BrowserProp.xml” file. Content of this file is shown below.

<properties>

<language>tr</language>

<helpContent>AuroraHelp.xml</helpContent>

<cache>true</cache>

<debug>true</debug>

<domain>AURORA</domain>

<firstPage>AURORA\_LOGIN.ebml</firstPage>

<alert>false</alert>

<alert-period>1000</alert-period>

<alert-service>AURORA\_ALERT</alert-service>

<alert-ebml>AURORA\_ALERT.ebml</alert-ebml>

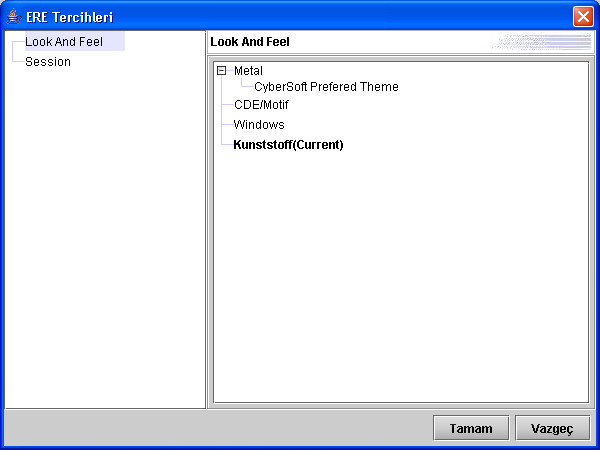
**<user-preferences>true</user-preferences>**

</properties>

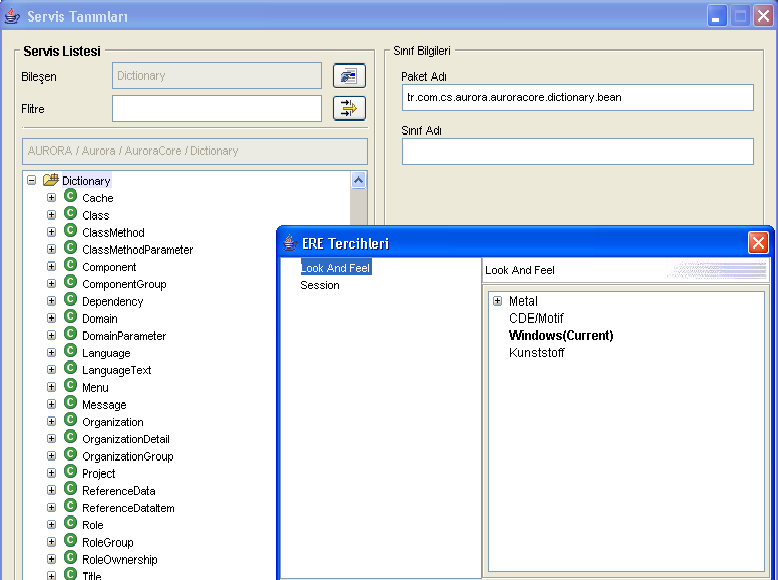
Follwing browser features can be accessed using this dialog.

#### 10.1.1 Look & Feel

Look&Feel determines how your Java UI components look like. This means that there are not a permanent appearances of components. ERE’s default look&fell manager is “Kunstoff” as you can see below.



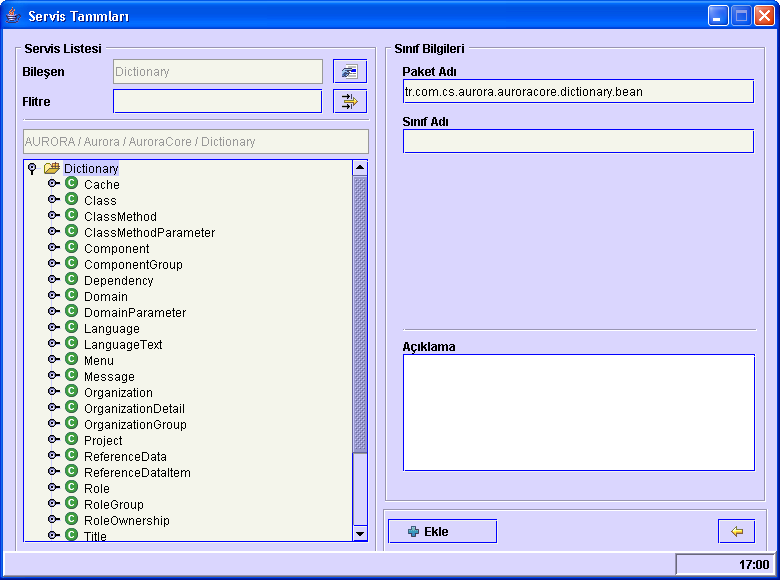
Some other look&feel samples are shown below.



**Windows Look&Feel**

Note: Windows loo&feel uses user interface of native Windows OS.

“Metal Look&Feel” has a special importance because it can use user defined themes. As you can see, Metal look&and feel tree node has subnodes. This nodes referes to metal themes defined in “EreMetalTheme.xml”. You can define lots of metal tehemes in this file. Don’t forget that themes are not look&feel’s. They can only effects GUI components color model not how they are rendered or shaped. When you expand metal node, you can see a predefined sample themes. As an example, we bring you “CyberSoft Prefered Theme” which is shown below.



CyberSoft Prefered Theme

EreMetalTheme.xml file content is show below.

<laf>

<theme name="CyberSoft Prefered Theme">

<p name="AcceleratorForeground" type="ColorUIResource"/>

<p name="AcceleratorSelectedForeground" type="ColorUIResource"/>

<p name="Black" type="ColorUIResource"/>

<p name="Control" type="ColorUIResource">218,214,254</p>

<p name="ControlDarkShadow" type="ColorUIResource">0,0,255</p>

<p name="ControlDisabled" type="ColorUIResource"/>

<p name="ControlHighlight" type="ColorUIResource"/>

<p name="ControlInfo" type="ColorUIResource"/>

<p name="ControlShadow" type="ColorUIResource"/>

<p name="ControlTextColor" type="ColorUIResource"/>

<p name="DesktopColor" type="ColorUIResource"/>

<p name="FocusColor" type="ColorUIResource"/>

<p name="HighlightedTextColor" type="ColorUIResource"/>

<p name="InactiveControlTextColor" type="ColorUIResource"/>

<p name="InactiveSystemTextColor" type="ColorUIResource"/>

<p name="MenuBackground" type="ColorUIResource">214,211,248</p>

<p name="MenuDisabledForeground" type="ColorUIResource"/>

<p name="MenuForeground" type="ColorUIResource"/>

<p name="MenuSelectedBackground" type="ColorUIResource">200,188,166</p>

<p name="MenuSelectedForeground" type="ColorUIResource"/>

<p name="PrimaryControl" type="ColorUIResource"/>

<p name="PrimaryControlDarkShadow" type="ColorUIResource"/>

<p name="PrimaryControlHighlight" type="ColorUIResource"/>

<p name="PrimaryControlInfo" type="ColorUIResource"/>

<p name="PrimaryControlShadow" type="ColorUIResource"/>

<p name="SeparatorBackground" type="ColorUIResource"/>

<p name="SeparatorForeground" type="ColorUIResource"/>

<p name="SystemTextColor" type="ColorUIResource"/>

<p name="TextHighlightColor" type="ColorUIResource"/>

<p name="UserTextColor" type="ColorUIResource"/>

<p name="White" type="ColorUIResource">244,245,235</p>

<p name="WindowBackground" type="ColorUIResource"/>

<p name="WindowTitleBackground" type="ColorUIResource"/>

<p name="WindowTitleForeground" type="ColorUIResource"/>

<p name="WindowTitleInactiveBackground" type="ColorUIResource"/>

<p name="WindowTitleInactiveForeground" type="ColorUIResource"/>

<p name="ControlTextFont" type="FontUIResource"/>

<p name="MenuTextFont" type="FontUIResource"/>

<p name="Primary1" type="ColorUIResource"/>

<p name="Primary2" type="ColorUIResource"/>

<p name="Primary3" type="ColorUIResource"/>

<p name="Secondary1" type="ColorUIResource"/>

<p name="Secondary2" type="ColorUIResource"/>

<p name="Secondary3" type="ColorUIResource"/>

<p name="SubTextFont" type="FontUIResource"/>

<p name="SystemTextFont" type="FontUIResource"/>

<p name="UserTextFont" type="FontUIResource"/>

<p name="WindowTitleFont" type="FontUIResource"/>

</theme>

</laf>

In this theme example, most of attributes are redundant. This theme can be defined as;

<theme name="CyberSoft Prefered Theme">

<p name="Control" type="ColorUIResource">218,214,254</p>

<p name="ControlDarkShadow" type="ColorUIResource">0,0,255</p>

<p name="MenuBackground" type="ColorUIResource">214,211,248</p>

<p name="MenuSelectedBackground" type="ColorUIResource">200,188,166</p>

<p name="White" type="ColorUIResource">244,245,235</p>

</theme>

An empty theme can be defined as;

<theme name="CyberSoft Prefered Theme"/>

This expression has no effects on Metal theme but definition is still valid.

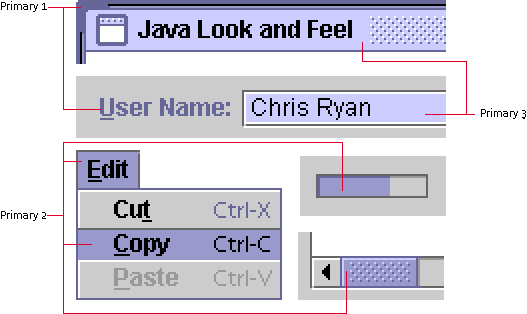
Some default definitions of look&fell are listed below.

#### 10.1.1.1 Primary Colors

The visual elements of Java look&feel applications use the primary colors as follows:

1. Primary 1 for active internal window borders, shadows of activated items, and system text, such as labels
2. Primary 2 for highlighting of activated items, such as menu titles and menu items; keyboard focus; active scroll boxes; and progress bar fill
3. Primary 3 for large colored areas, such as the title bar of active internal windows and the background of selected text

The usage is illustrated in the following figure.



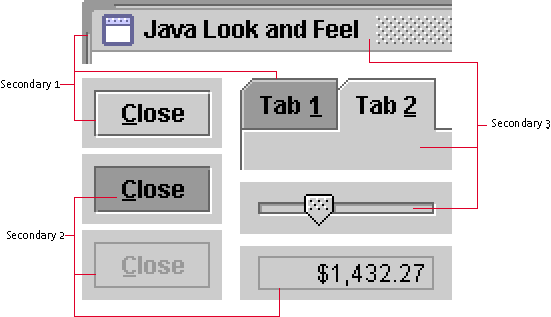
**Primary Colors in Default Color Theme**

#### 10.1.1.2 Secondary Colors

The visual elements of Java look& feel applications use the secondary colors as follows:

* Secondary 1 for the dark border that creates flush 3D effects for items such as command buttons
* Secondary 2 for inactive internal window borders, shadows, pressed buttons, and dimmed command button text and borders
* Secondary 3 for the background canvas, the background of noneditable text fields, and inactive title bars for internal windows

The usage is shown in the following figure.



**Secondary Colors in Default Color Theme**

#### 10.1.1.3 Black and White

Black and white have defined roles in the Java look&feel color model. In particular, black appears in:

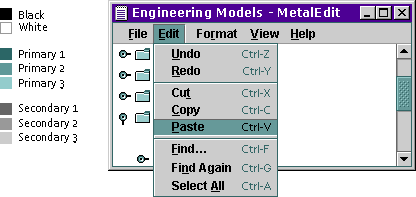
* User text, such as the entry in an editable text field
* Control text, such as menu titles and menu items
* Title text in an internal window
* Button text in command buttons
* Tab text in tabbed panes
* Text in noneditable text fields

White is used for:

* Highlighting the flush 3D appearance of such components as command buttons
* Background of editable text fields

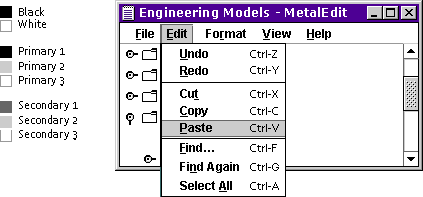
#### 10.1.1.4 Redefinition of Colors

The simplest modification you can make to the color theme is to redefine the primary colors. For instance, you can substitute greens for the purple-blues used in the default theme, as shown in the following figure.



**Green Color Theme**

You can use the same value for more than one of the eight colors--for instance, a high-contrast theme might use only black, white, and grays. The following figure shows a theme that uses the same grays for primary 2 and secondary 2. White functions as primary 3 and secondary 3 as well as in its normal role.



**High-Contrast Color Theme**

#### 10.1.1.5 Fonts

As part of the theme mechanism and parallel to the color model, the Java look and feel provides a default font style model for a consistent look. You can use themes to redefine font typefaces, sizes, and styles in your application. The default Java look and feel theme defines four font categories, called "type styles": the control font, the system font, the user font, and the small font. The actual fonts used vary across platforms.

The following table shows the mappings to Java look and feel components for the default theme.

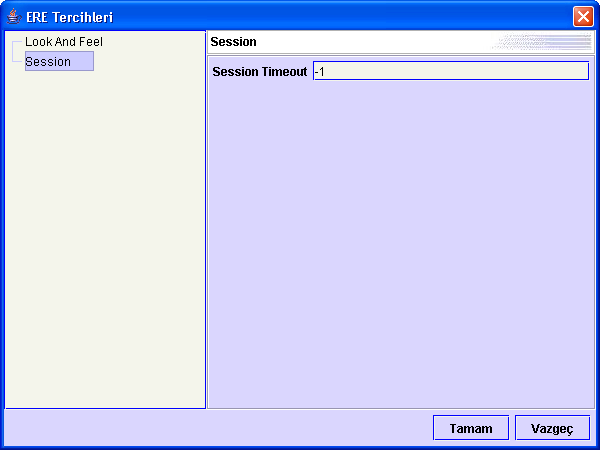
|  |  |  |
| --- | --- | --- |
| **Type Style** | **Default Theme** | **Use** |
| Control | 12-point bold | Buttons, checkboxes, menu titles, labels, and window titles |
| Small | 10-point plain | Keyboard shortcuts in menus and tool tips |
| System | 12-point plain | Tree components and tool tips |
| User | 12-point plain | Text fields and tables |

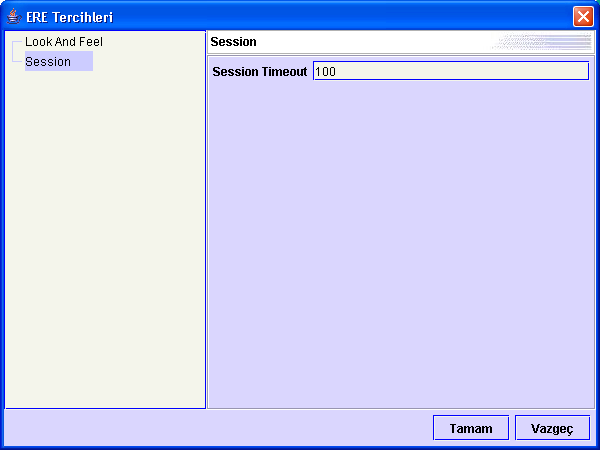
**Type Styles Defined by the Java Look&Feel**

Note : For more information ; <http://java.sun.com/products/jlf/ed2/book/>

#### 10.1.2 Session

Session timeout value determines how long a session can last without any user action. You can edit this value using preferences dialog. “-1” indicates unlimited session time. Otherwise this value refers to session time in seconds.





## 10.2 Project Preferences

Project preferences defines an xml file (EBMLProjectPreferences.xml) which every AURORA design bean is defined in. This is a server side xml file which is requested in ERE session.

Every bean is defined just as being EBML files. “EBMLProjectPreferences.xml” file content is shown below.

<preferences>

<p name="font">Dialog,12,1,0,0</p>

<beans>

<bean class="tr.com.cs.aurora.ebml.bean.swing.JCSList">

<style>

<p name="background">255,255,255</p>

<p name="bounds">170,50,150,30</p>

<p name="enabled">true</p>

<p name="font">$font</p>

<p name="foreground">0,0,0</p>

<p name="tabOrder">0</p>

<p name="unique">false</p>

<p name="visible">true</p>

</style>

</bean>

...

...

</beans>

</preferences>

In this file, you can define global property definition for this file. As seen in red font, it is a font definition that is defined once and used any bean’s font definition.

For instance, you can define a global bounds(x,y,width,height) that every text component uses. Example is given below.

<preferences>

<p name="font">Dialog,12,1,0,0</p>

<p name="bounds">100,100,120,30</p>

It is important to realise that any component which’s bounds are set to $bounds will be 120 in width and 30 in height. It doesn’t mean that x and y localtion will be 100,100. Because you drag it anywhere on page. Only width and height properties will be set.

This file is important for another reason too. First a textfield bean definition is given below.

<bean class="tr.com.cs.aurora.ebml.bean.swing.JCSTextField">

<style>

<p name="autoSkip">false</p>

<p name="background">255,255,255</p>

<p name="bounds">15,140,110,20</p>

<p name="contentType">Plain</p>

<p name="editable">true</p>

<p name="font">Dialog,12,0,0,0</p>

<p name="foreground">0,0,0</p>

<p name="horizontalAlignment">10</p>

<p name="limit">-1</p>

<p name="minCharCount">0</p>

<p name="mustEnter">false</p>

<p name="mustFill">false</p>

<p name="readOnly">false</p>

<p name="tabOrder">0</p>

<p name="visible">true</p>

<p name="zeroPad">false</p>

</style>

</bean>

If you put a text field bean on EBML page, insect property values with this bean definition are not written in EBML file. So, file size is kept small. Follwing example shows that only limit property is changed for text field.



So, the output EBML is only includes limit and bounds properties of text field. It is shown below.

<style>

<p name="bounds">100,75,110,20</p>

<p name="limit">4</p>

</style>

## 10.3 Global Preferences

Global preferences which defines an xml file provides global bean property definitions by means of global ID. Every EDS bean consist of a property ,“Global ID”. When global ID property is set to any value, first job is to look for whether it is a preset value. If global ID is found, second job is to examine whether that global ID consist of a class definition which’s value is equal to bean class name. At the end, found class properties are all set.

Global preferences subsystem uses an xml file, which has a unique name “EBMLGlobals.xml”. An example “EBMLGlobals.xml” file content is given below.

<global-preferences>

<p name="font">Dialog,12,1,0,0</p>

<globals>

<global name="Customer">

<bean class="tr.com.cs.aurora.ebml.bean.swing.JCSPage">

<style>

<p name="background">224,224,224</p>

</style>

</bean>

<bean class="tr.com.cs.aurora.ebml.bean.swing.JCSLabel">

<style>

<p name="bounds100,100,100,20</p>

<p name="enabled">true</p>

<p name="font">$font</p>

<p name="foreground">0,0,0</p>

<p name="horizontalAlignment">10</p>

<p name="visible">true</p>

<p name="text">Test Label</p>

</style>

</bean>

</global>

</globals>

</global-preferences>

If you read preoject preferences, you will be familiar with this. These are a global property definition in this file. You have to use this value with a “$” character before it.

### Why Need Global Preferences ?

Global Preferences has the highest priority before project preferences and EBML design property. Imagine that, you design an EBML file and you set label’s foreground to “255,0,0”, red. Also, this label has a global ID “Customer”.

<global name="Customer">

<bean class="tr.com.cs.aurora.ebml.bean.swing.JCSLabel">

<style>

<p name="foreground">0,0,255</p>

</style>

</bean>

<bean>

...

...

</bean>

...

...

</global>

That global preference definition forces label component look like BLUE which’s value is “0,0,255”.

So, when you deploy that file to CMS, you manage all clients with a quick and an effective manner.

# 

# 11. ADDING NEW BEAN

In this section, you will learn to add new beans to EDS toolbar. Designing a bean consist of 3 aspects.

* Writing bean class , implementing someinterfaces.
* Writing beanInfo class, which defines beans property,event and method descriptors
* If necessary, implementing your listener classes and event objects.

## 11.1 AURORA BEAN INTERFACES

Before writing a test class, you must implement bean interfaces. You should not implement all.

* ICSBean : For non-focusable components. JCSLabel,etc.

**public** **interface** ICSBean {

**public** **void** cleanup() ;

**public** **void** setGlobalID(String globalID);

**public** String getGlobalID() ;

**public** **void** setDirty(**boolean** dirty) ;

**public** **boolean** isDirty() ;

**public** **boolean** isBeanValid() ;

}

* ICSFocusableBean: For focusable components. JCSTable,JCSTree,etc.

**public** **interface** ICSFocusableBean **extends** ICSBean{

**int** getTabOrder();

}

Let’s have a look at what that method means.

* + Cleanup: When a page load event occurs or some thing else , bean cleans up its content
  + isBeanValid : When user attempts to call a service, aurora checks wherther this component is valid for remote call. For instance, when JCSTextField’s limit property is set to “n” and it goes to service with empty content, isBeanValid return false. So that, service call fails.
  + isDirty / setDirty : When bean content is changed from initial content this method returns false.
  + setTabOrder / getTabOrder : It sets or gets an internal integer variable which defines tab order on page.

## 11.2 WRITING BEAN CLASS

Now lets write a bean class, called “JCSFoo.java”.

**package** tr.com.cs.aurora.ebml.bean.swing;

**import** javax.swing.JComponent;

**import** tr.com.cs.aurora.ebml.bean.ICSFocusableBean;

**public** **class** JCSFoo **extends** JComponent **implements** ICSFocusableBean {

**private** String comment ;

**private** String globalID ;

**private** **boolean** dirty ;

**private** **int** tabOrder ;

**public** JCSFoo() {

**super**();

}

**public** **void** cleanup() {

}

**public** **void** setGlobalID(String globalID) {

**this**.globalID = globalID;

}

**public** String getGlobalID() {

**return** globalID;

}

**public** **void** setDirty(**boolean** dirty) {

**this**.dirty = dirty;

}

**public** **boolean** isDirty() {

**boolean** someValue = **false**;

//Check whether this component is dirty.

**return** someValue;

}

**public** **boolean** isBeanValid() {

**boolean** someValue = **false**;

//Check whether this component is valid.

**return** someValue;

}

**public** **void** setTabOrder(**int** tabOrder){

**this**.tabOrder = tabOrder;

}

**public** **int** getTabOrder() {

**return** tabOrder;

}

**public** String getComment() {

**return** comment;

}

**public** **void** setComment(String comment) {

**this**.comment = comment;

}

**public** **void** testMethod(String str,**int** i){

System.out.println(str + i);

fireFooEventOccured();

}

**public** **void** addFooListener(CSFooListener listener){

listenerList.add(CSFooListener.**class**,listener);

}

**public** **void** removeFooListener(CSFooListener listener){

listenerList.remove(CSFooListener.**class**,listener);

}

**public** **void** fireFooEventOccured() {

EventListener[] listeners =

listenerList.getListeners(CSFooListener.**class**);

**for** (**int** i = 0; i < listeners.length; i++) {

CSFooListener listener = (CSFooListener)listeners[i];

listener.fooEventOccured(**new** CSFooEvent(**this**));

}

}

}

Listener interface and event object is declared shown below.

**package** tr.com.cs.aurora.ebml.bean.swing.event;

**import** java.util.EventListener;

**public** **interface** CSFooListener **extends** EventListener {

**public** **void** fooEventOccured(CSFooEvent event);

}

**package** tr.com.cs.aurora.ebml.bean.swing.event;

**import** java.util.EventObject;

**public** **class** CSFooEvent **extends** EventObject{

**public** CSFooEvent(Object source) {

**super**(source);

}

}

## 11.3 WRITING BEANINFO CLASS

After wiritng bean class, you should write a bean info class which’s name starts with class name and ends with “BeanInfo”. For instance, JCSFooBeanInfo.

**package** tr.com.cs.aurora.ebml.bean.swing;

**import** java.awt.Image;

**import** java.beans.EventSetDescriptor;

**import** java.beans.IntrospectionException;

**import** java.beans.Introspector;

**import** java.beans.MethodDescriptor;

**import** java.beans.PropertyDescriptor;

**import** tr.com.cs.aurora.ebml.bean.EBMLBeanInfo;

**public** **class** JCSFooBeanInfo **extends** EBMLBeanInfo {

**private** **final** **static** Class beanClass = JCSFooBeanInfo.**class**;

**public** JCSFooBeanInfo() {

**super**();

}

**public** PropertyDescriptor[] getPropertyDescriptors() {

addPropertyDescriptor(

beanClass,"background","Background",**false**,**false**,**fals**)

addPropertyDescriptor(

beanClass,"comment","Comment",**true**,**false**,**true**);

addPropertyDescriptor(

beanClass,"enabled","Enabled",**false**,**false**,**false**);

addPropertyDescriptor(

beanClass,"focusable","Focusable",**false**,**false**,**false**);

addPropertyDescriptor(

beanClass,"font","Font,**false**,**false**,**false** );

addPropertyDescriptor(

beanClass,"foreground","Foreground",**false**,**false**,**false**);

addPropertyDescriptor(

beanClass,"height","Height",**false**,**false**,**true**);

addPropertyDescriptor(

beanClass,"name","Name",**true**,**false**,**true**);

addPropertyDescriptor(

beanClass,"tabOrder","Tab Order",**true**,**false**,**false**);

addPropertyDescriptor(

beanClass,"visible","Visible",**false**,**false** ,**false**);

addPropertyDescriptor(

beanClass,"width","Width",**false**,**false**,**true**);

addPropertyDescriptor(

beanClass,"location","Location",**true**,**false**,**true** );

addPropertyDescriptor(

beanClass,"globalID","Global ID",**false**,**false**,**false**);

addPropertyDescriptor(

beanClass,"bounds","Bounds",**true**,**true**,**false**);

**return** **super**.getPropertyDescriptors();

}

**public** Image getIcon(**int** iconKind) {

**try** {

**return** Introspector.getBeanInfo(JCSFooBeanInfo.**class**).getIcon(iconKind);

} **catch** (IntrospectionException e) {

e.printStackTrace();

}

**return** **null**;

}

**public** EventSetDescriptor[] getEventSetDescriptors() {

String[] fooListenerMethods = {" fooEventOccured "};

addEventSetDescriptor(

beanClass, CSFooListener.**class**,fooListenerMethods);

**return** **super**.getEventSetDescriptors();

}

**public** MethodDescriptor[] getMethodDescriptors() {

addMethodDescriptor(

beanClass,"testMethod" ,**new** Class[]{String.**class**,**int**.**class**} ,**new** String[]{"str","value"});

**return** **super**.getMethodDescriptors();

}

}

# 12. JASPERREPORTS

Another reporting concept in AURORA is **JasperReports**. JasperReports is pure java implemented reporting engine. You can reach more information at <http://jasperreports.sourceforge.net/> and <http://jasperreports.sourceforge.net/tutorial/>

JasperReport API is distributed with AURORA.

**iReport** is JasperReport design tool which any report can be designed visually. How to design reports in iReport can be found at <http://ireport.sourceforge.net/docs.html>.

In this section, AURORA-JasperReports entegration will be mentioned. Other design information can be obtain via internet.

## 12.1 JASPER REPORT XML (jrxml)

JasperReport engine parses and executes an xml file (\*.jrxml). This file contains report view definitons, report variable, field related to database columns, report parameters which can be pass from AURORA platfom. An empty report xml is shown below.

<?xml version="1.0" encoding="UTF-8" ?>

<!-- Created with iReport - A designer for JasperReports -->

<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//EN" "http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport

name="TEST\_REPORT" columnCount="1" printOrder="Vertical" orientation="Portrait" pageWidth="595" pageHeight="842"

columnWidth="535" columnSpacing="0" leftMargin="30" rightMargin="30"

topMargin="20" bottomMargin="20" whenNoDataType="NoPages"

isTitleNewPage="false" isSummaryNewPage="false">

<property name="ireport.scriptlethandling" value="2" />

<background>

<band height="0" isSplitAllowed="true" >

</band>

</background>

<title>

<band height="50" isSplitAllowed="true" >

</band>

</title>

<pageHeader>

<band height="50" isSplitAllowed="true" >

</band>

</pageHeader>

<columnHeader>

<band height="30" isSplitAllowed="true" >

</band>

</columnHeader>

<detail>

<band height="100" isSplitAllowed="true" >

</band>

</detail>

<columnFooter>

<band height="30" isSplitAllowed="true" >

</band>

</columnFooter>

<pageFooter>

<band height="50" isSplitAllowed="true" >

</band>

</pageFooter>

<summary>

<band height="50" isSplitAllowed="true" >

</band>

</summary>

</jasperReport>

## 12.2 DATASOURCE XML (dsxml)

Jasper report definition file defines the report in the visual way. There has to be another part to define the data content of the report, and this is done by datasource definition (datasource xml) files (\*.dsxml).

When a jasper report xml file (.jrxml) created and deployed to CMS, a related datasource xml file with the same name has to be created and deployed , too.

There are two type of datasource definition files :

### Sql Type Dsxml

This type of datasource definition executes a sql query for report datasource.

<report v="1.00" type="**sql**">

<sql dataSource="AuroraDS">

<query>SELECT \* FROM AURORA.DICT\_ML\_USER</query>

<parameters/>

</sql>

<report-parameters/>

</report>

"dataSource" property of "sql" tag defines which datasource will be used in executing the query specified.

"paramaters" as a child of "sql" tag is used for parameter passing to sql definition.

<sql dataSource="AuroraDS">

<query>SELECT USERNAME,USER\_PASSWORD FROM AURORA.DICT\_ML\_USER WHERE

DOMAIN\_OID LIKE ?

</query>

<parameters>

<param suffix="%" prefix="%" type="string">DOMAIN</param>

</parameters>

</sql>

Parameter definitions takes three properties; suffix (must-value after parameter), prefix (must-value before parameter) and type (optional-type of the parameter, default value "string")

Every "?" in sql definition replaces with a sql parameter in order (like preparedStatement in Java).

### Service Type Dsxml

This type of datasource definition calls the service mentioned in dsxml file and returns for report datasource.

<report v="1.00" type="**service**">

<service>AURORA\_DICT\_DOMAIN\_LIST\_FOR\_COMBO</service>

<report-parameters>

<param>TITLE</param>

</report-parameters>

</report>

As it can be seen from the example above, service type datasource definitions do not "parameters" tag; parameters to the service are taken from the function definition. The service ,mentioned in dsxml file, must set data content in 2-dimensional way under the BagKey "CORE\_DATA" in the CSBag object it returns.

### Report Parameters

Report parameters is used for passing parameters from ERE to JasperReport Engine.

<report v="1.00" type="**sql**">

<sql dataSource="AuroraDS">

<query>SELECT \* FROM AURORA.DICT\_ML\_USER</query>

<parameters/>

</sql>

<report-parameters>

<param>TITLE</param>

</report-parameters>

</report>

Report parameters must have the same name defined in reports. A sample report parameter passing example is shown below. In this example, data and report parameters will ve passed to JasperReport. Parameter passing is done using function desing in EDS.

<report v="1.00" type="sql">

<sql dataSource="AuroraDS">

<query>

SELECT \* FROM AURORA.DICT\_ML\_USER WHERE USERNAME LIKE ? </query>

<parameters>

<param suffix="%" prefix="%">USER\_NAME</param>

</parameters>

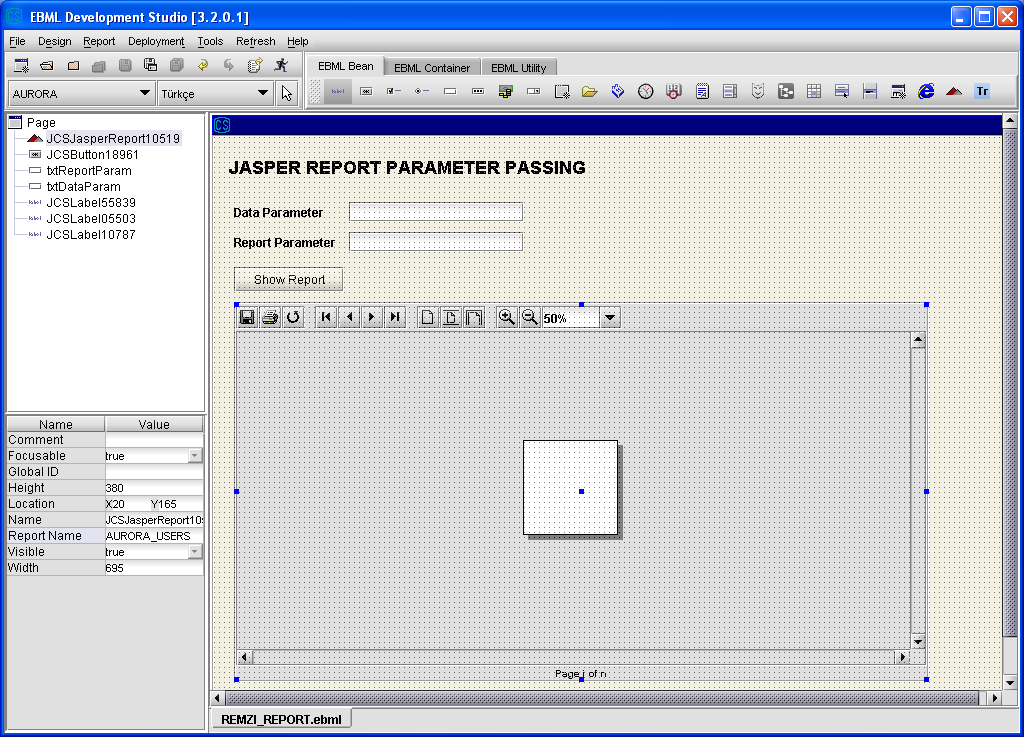
</sql>

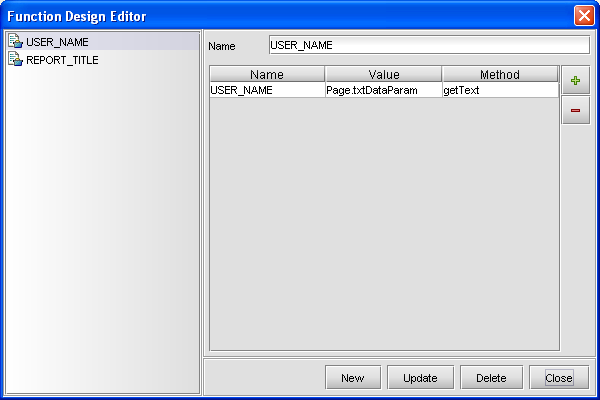
<report-parameters>

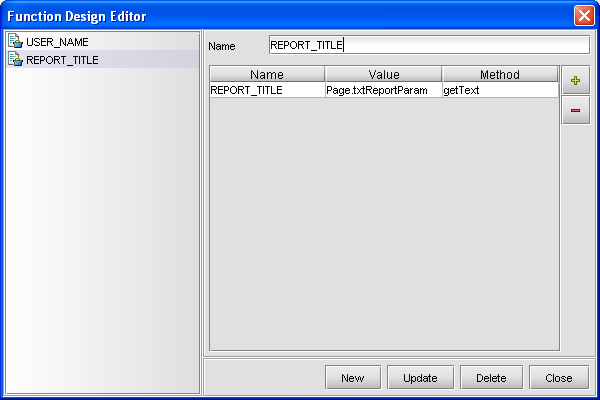
<param>REPORT\_TITLE</param>

</report-parameters>

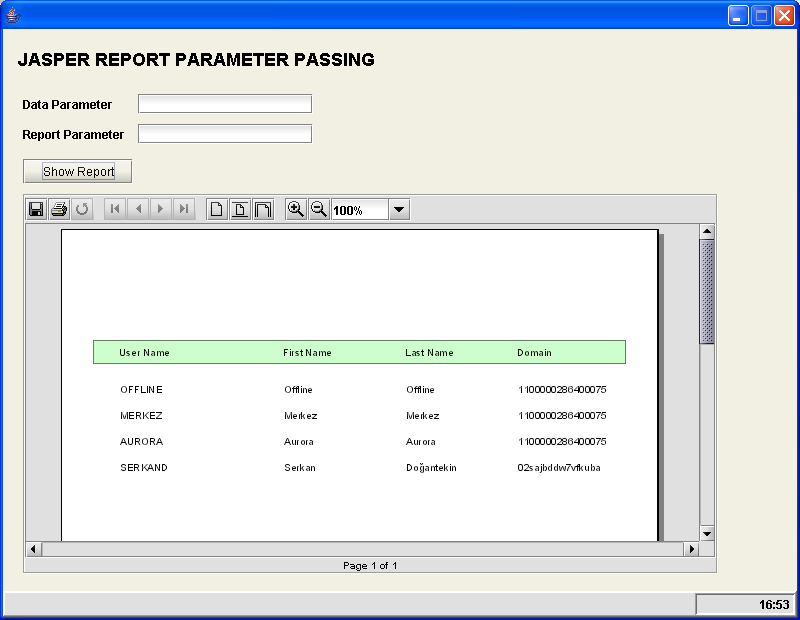
</report>

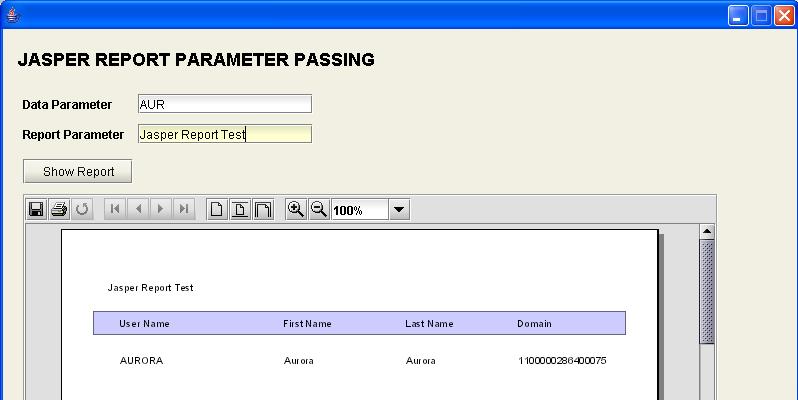






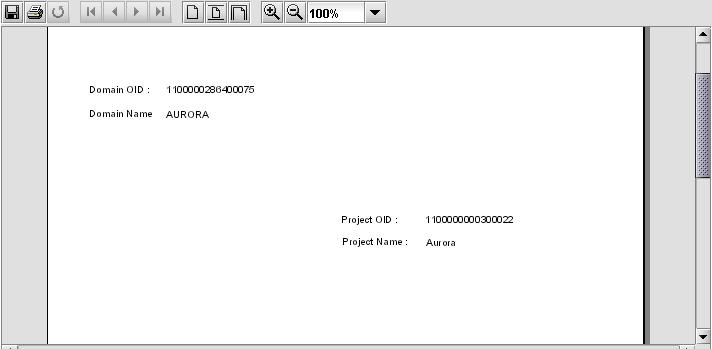






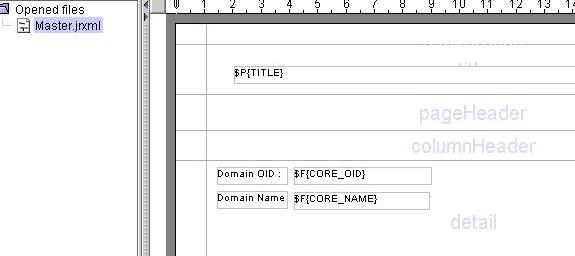
## 12.2 Sub Report

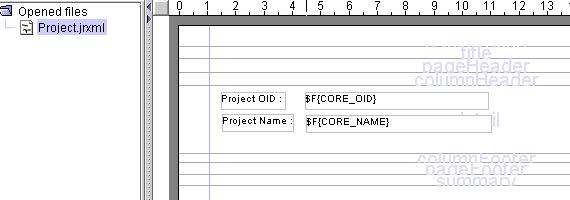
One of the strong points of Jasper Reports is its support for the sub reports. By this functionality, it is possible to represent more than one reports in one "main" report.



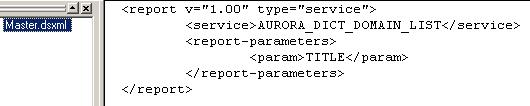
Steps in designing reports that have subreports :

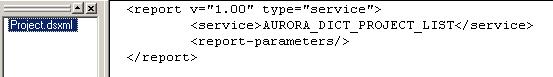
* Design all reports as they are all stand-alone reports. (create .jrxml files)



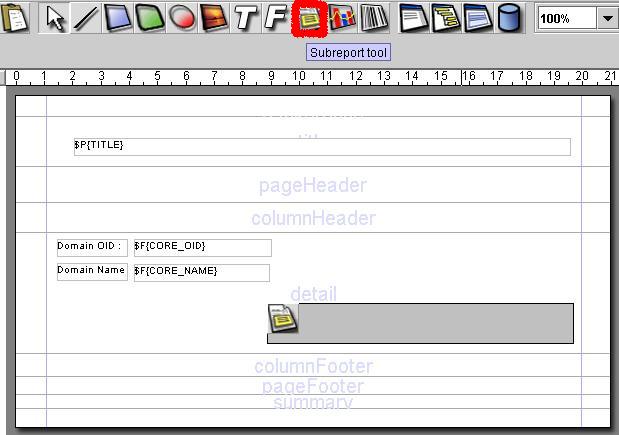


* Design all reports datasource definitions as they are all stand-alone reports. (create .dsxml files)

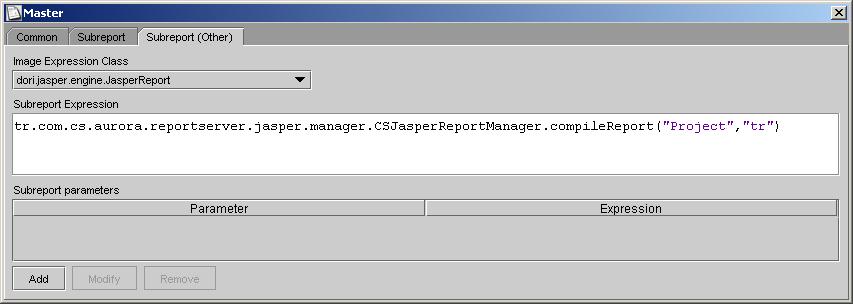




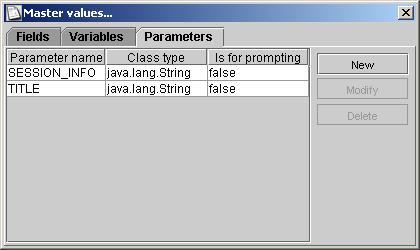
* Insert subreport tool in master report’s definition.



* Define the subreport and its language by selecting "properties" menu item on subreport.

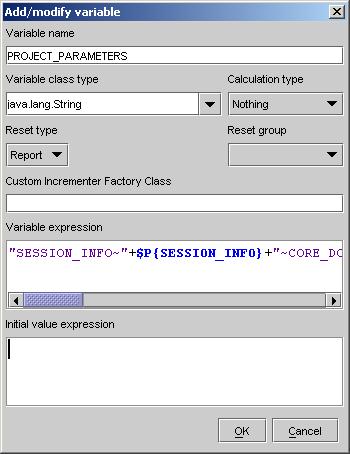


* Define parameter passing between master and subreport. All parameters are passed in variables. A parameter named "SESSION\_INFO" must be passed from master to subreport. It must be defined in master report with default value "" in the master report, its content is handled by the AURORA.

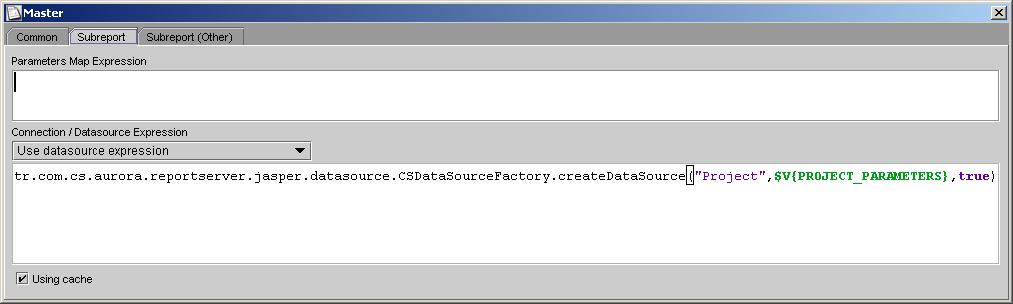


To pass parameters:

* 1. Define a variable in master report (in this example "PROJECT\_PARAMETERS")
  2. Set this variable’s "variable expression" property a string as "parameter\_name~parameter\_value"



* Define subreport’s datasource with the variable designed for parameter passing by selecting "properties" menu item on subreport.



* Deploy necessary parts (.jrxml and .dsxml files)

## 12.3 Using Text Report Viewer

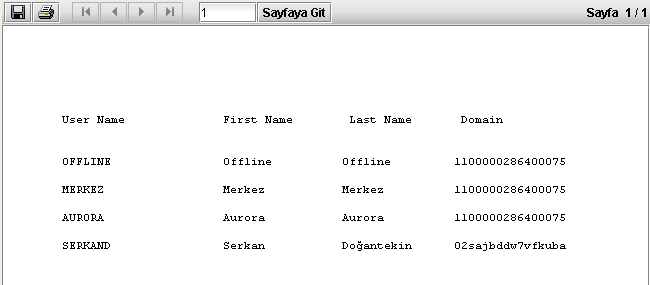
Text report is a jasper report extension. Because jasper report can not directly support text report viewing , this component produces text output for screen and text printer(JCSTextPrinter).

You can design text reports using iReport but aware that you can see it a bit different. Indeed, this component design to print out text reports using printer whichs is not available. So, text reports can be printed by any text printer not caring if there should be printer driver.

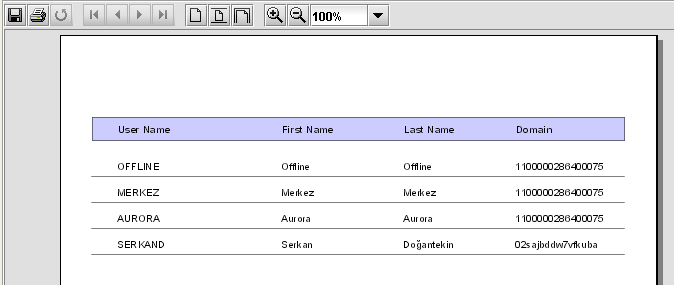
Text report exported ,on server side, produces a character matrix for a text pane. So every line is placed an row in text pane. Be aware that report data can not painted on report viewer just as been in jasper report viewer. It is set to text pane as text data. So you would be probably in trouble while designing text reports.

Viewer doesn’t care font or color information of report fields. So that text printer can be print byte stream supported by escape codes. But it is available to print bold font if a field is set to bold using iReport. It means that only bold information of report field is available for print out.

**Restriction** : When designing a text report (a report that intent to be a text report) , every fields ***width*** must be different from each other. Otherwise, unexpectedly report will seen disorganized.

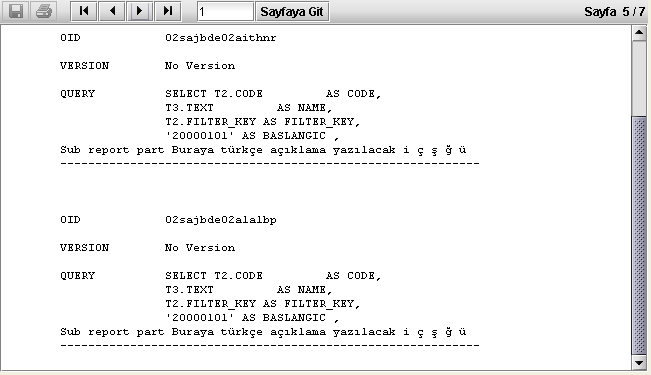


Tetx report view of report USER.jrml



Jasper report view of report USER.jrml

Text report viewer support multiline strings.



Multiline text report example