

IBM Cúram Social Program Management

Cúram Web Client Reference Manual

Version 6.0.4

Note

Before using this information and the product it supports, read the information in Notices at the back of this guide.

This edition applies to version 6.0.4 of IBM Cúram Social Program Management and all subsequent releases and modifications unless otherwise indicated in new editions.

Licensed Materials - Property of IBM

Copyright IBM Corporation 2012. All rights reserved.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

© Copyright 2008-2012 IBM Corporation

Table of Contents

Chapter 1 Introduction	1
1.1 Introduction	1
1.2 Prerequisites	1
1.3 Companion Guides	1
1.4 Structure	2
1.5 Summary	2
Chapter 2 Concepts	3
2.1 Objective	3
2.2 Prerequisites	3
2.3 Introduction	3
2.4 Application User Interface Overview	4
2.5 User Interface Meta-data	5
2.5.1 Page Content Meta-data	5
2.6 Applications	8
2.7 Page Context	11
2.8 Page Look-and-Feel	11
2.9 Application Controller Java Server Page	12
2.10 Direct Browsing	12
2.11 Summary	12
Chapter 3 Development	14
3.1 Objective	14
3.2 Prerequisites	14
3.3 Introduction	14
3.4 Outline of the Development Process	14
3.5 Installation	15
3.6 Project Folder Structure	16
3.7 Application Components	19
3.7.1 Component Folders	19
3.7.2 Component Order	20
3.8 Component Artifacts	21
3.9 Application Locales	22
3.10 Building an Application	23
3.10.1 Build Targets	23
3.10.2 Related Build Targets	25
3.10.3 Full and Incremental Builds	25

3.10.4	Dependency Checking	26
3.10.5	Build Logs	26
3.10.6	Error Reporting	26
3.10.7	Server Interface Reference	27
3.10.8	Page Previews	28
3.10.9	UIM Generator Tool	28
3.10.10	External Client Applications	29
3.11	Deployment	30
3.11.1	Overview	30
3.11.2	Configuring the Application	30
3.11.3	Customizing the Web Application Descriptor	34
3.12	Customization	36
3.12.1	Overview	36
3.12.2	Adding New Artifacts	36
3.12.3	Overriding or Merging Artifacts	37
3.12.4	Externalized Strings	37
3.12.5	Images	38
3.12.6	Image Mapping	39
3.12.7	CuramLinks.properties	40
3.12.8	XML Runtime Configuration Files	40
3.12.9	Login Pages	40
3.12.10	JavaScript Files	41
3.12.11	Cascading Stylesheets	42
3.12.12	Application Configuration Files	44
3.12.13	General Configuration	45
3.12.14	Custom Resources	52
Chapter 4	Localization	55
4.1	Objective	55
4.2	Prerequisites	55
4.3	Introduction	55
4.4	Numbers	55
4.5	File Encoding	55
4.5.1	XML Files	56
4.5.2	Java properties files	56
4.5.3	Non-XML Files	57
4.6	Locales	57
4.6.1	Non JavaScript property files	58
4.6.2	JavaScript property files	58
4.7	UIM Externalized Strings	59
4.8	JavaScript Externalized Strings	59
4.8.1	Accessing properties in JavaScript	60
4.9	Image.properties	60
4.10	Infrastructure Widget Properties Files	61
4.10.1	Frequency Pattern Selector Localization	62
4.11	CDEJResources.properties	64
4.12	ApplicationConfiguration.properties	64
4.13	Application-wide Menu	64
4.14	Tabbed Configuration Artifacts	65

4.15 Runtime Messages	65
Chapter 5 UIM Reference	67
5.1 Objective	67
5.2 Prerequisites	67
5.3 Introduction	67
5.4 Creating UIM Documents	67
5.5 UIM Document Types	67
5.6 UIM Pages	68
5.7 UIM Views	68
5.8 Externalized Strings	69
5.9 UIM Reference for Pages and Views	69
5.9.1 Introduction	69
5.9.2 Connection Types	69
5.9.3 ACTION_CONTROL	71
5.9.4 ACTION_SET	76
5.9.5 CLUSTER	78
5.9.6 CONDITION	81
5.9.7 CONNECT	82
5.9.8 CONTAINER	82
5.9.9 DETAILS_ROW	84
5.9.10 DESCRIPTION	85
5.9.11 FIELD	86
5.9.12 FOOTER_ROW	91
5.9.13 IMAGE	93
5.9.14 INCLUDE	93
5.9.15 INITIAL	94
5.9.16 INFORMATIONAL	94
5.9.17 INLINE_PAGE	95
5.9.18 IS_FALSE	97
5.9.19 IS_TRUE	98
5.9.20 JSP_SCRIPTLET	98
5.9.21 LABEL	101
5.9.22 LINK	102
5.9.23 LIST	108
5.9.24 MENU	111
5.9.25 PAGE	118
5.9.26 PAGE_PARAMETER	121
5.9.27 PAGE_TITLE	122
5.9.28 SCRIPT	123
5.9.29 SERVER_INTERFACE	124
5.9.30 SOURCE	126
5.9.31 TAB_NAME	127
5.9.32 TARGET	128
5.9.33 TITLE	128
5.9.34 VIEW	129
5.10 UIM Reference for Widgets	130
5.10.1 Introduction	130
5.10.2 WIDGET	130

5.10.3	WIDGET_PARAMETER	132
5.10.4	The EVIDENCE_COMPARE Widget	133
5.10.5	The FILE_EDIT Widget	133
5.10.6	The FILE_UPLOAD Widget	136
5.10.7	The FILE_DOWNLOAD Widget	139
5.10.8	The MULTISELECT Widget	140
5.10.9	The SINGLESELECT Widget	144
5.10.10	The RULES_SIMULATION_EDITOR Widget	144
5.10.11	The IEG_PLAYER Widget	146
5.11	Dynamic UIM Cross Reference	146
5.12	Dynamic UIM System Initialization	147
Chapter 6	Application Configuration	149
6.1	Objective	149
6.2	Prerequisites	149
6.3	Introduction	149
6.4	Configuration Files	150
6.5	Applications	151
6.5.1	Introduction	151
6.5.2	Definition	152
6.5.3	Optional Header	158
6.5.4	Example	158
6.5.5	Associate an Application with User	160
6.6	Sections	161
6.6.1	Introduction	161
6.6.2	Definition	162
6.6.3	Example	164
6.7	Section Shortcut Panel	164
6.7.1	Introduction	164
6.7.2	Definition	165
6.7.3	Example	167
6.8	Tabs	167
6.8.1	Introduction	168
6.8.2	Definition	169
6.8.3	Context Panel UIM	175
6.8.4	Example	176
6.9	Tab Actions Menu	176
6.9.1	Introduction	176
6.9.2	Definition	177
6.9.3	Dynamic Support	181
6.9.4	File Download Menu Item	182
6.9.5	Example	183
6.10	Tab Navigation	183
6.10.1	Introduction	183
6.10.2	Definition	184
6.10.3	Dynamic Support	188
6.10.4	Example	189
6.11	Opening Tabs and Sections	190
6.11.1	Introduction	190

6.11.2 Links	190
6.11.3 Page to Tab Associations	191
6.11.4 Tab to Section Associations	192
6.11.5 Page Parameters	192
Chapter 7 Session Management	195
7.1 Objective	195
7.2 Prerequisites	195
7.3 Introduction	195
7.4 Session Basics	195
7.5 Tab Restoration	196
7.6 Configuration	197
7.7 Limitations	198
7.8 Browser Specific Session Management	198
Chapter 8 Domain Specific Controls	200
8.1 Objective	200
8.2 Prerequisites	200
8.3 Introduction	200
8.4 Dates	200
8.5 Date-Times	201
8.5.1 Representing time-only values	201
8.5.2 Customizing the Time Format	202
8.6 Frequency Pattern Selector	202
8.7 Selection Lists	203
8.7.1 Populated from a Code-Table	203
8.7.2 Populated from Server Interface Properties	204
8.7.3 Drop-down, Scrollable and Checkboxed List types	205
8.7.4 Adding an Empty Entry to a List for Non-Mandatory Fields	205
8.7.5 Enabling Multiple Selection	205
8.7.6 Transfer List Widget	206
8.8 User Preferences Editor	206
8.9 Rules Trees	207
8.9.1 Introduction	207
8.9.2 Default Rules View	208
8.9.3 Summary Rules View	209
8.9.4 Failed Rules View	209
8.9.5 Dynamic Rules View	209
8.9.6 Dynamic Full Tree Rules View	213
8.9.7 Rules Editor	213
8.10 Meeting View	215
8.10.1 Overview	216
8.10.2 Single Selection Mode	216
8.10.3 Multiple Selection Mode	216
8.10.4 XML Formats	216
8.11 Charts	218
8.11.1 Overview	218
8.11.2 Chart appearance	218
8.11.3 Chart configuration	221
8.11.4 Chart Data Formats	225

8.12 Heatmap Widget	226
8.12.1 Overview	226
8.12.2 Configuration	227
8.13 Workflow	228
8.13.1 Overview	228
8.13.2 Workflow Details	228
8.13.3 Workflow XML Formats	229
8.14 Evidence View	233
8.14.1 Evidence Display Mode	233
8.14.2 Evidence Comparison Mode	234
8.14.3 Configuration	234
8.14.4 Data Format	235
8.15 Calendar	236
8.16 Payment Statement View	240
8.17 Batch Function View	242
8.18 Addresses	242
8.19 Schedule View	244
8.20 Radio Button Group	245
8.21 Pop-up Pages	245
8.21.1 Configure the Pop-up Page	245
8.21.2 Create the Pop-up Page	248
8.21.3 Using the Pop-up Page	251
8.21.4 Using Multiple Pop-up Search Pages for a Single Field	252
8.21.5 Configure the Multiple Pop-up Page	252
8.21.6 Using the Multiple Pop-up Page	253
8.22 Agenda Player	254
8.22.1 Agenda Player screen structure	254
8.22.2 Navigation modes	255
8.22.3 Navigator-less View	256
8.22.4 Agenda Player Configuration	256
8.22.5 Agenda Player Customization	257
8.22.6 Player data	258
8.23 LOCALIZED_MESSAGE Domain	262
8.24 Decision Assist: Decision Matrix Widget	263
8.24.1 Overview	263
Chapter 9 Custom Data Conversion and Sorting	264
9.1 Objective	264
9.2 Prerequisites	264
9.3 Introduction	264
9.4 Data Conversion and Sorting Operations	265
9.5 Data Conversion Life Cycle	267
9.6 The Domain Hierarchy and Domain Plug-ins	268
9.7 Overview of Domain Plug-ins	270
9.7.1 Common Features of Plug-ins	270
9.7.2 Converter Plug-ins	270
9.7.3 Comparator Plug-ins	272
9.7.4 Default Value Plug-ins	272
9.8 Domain Plug-in Configuration	273

9.9	Out-of-the-Box Domain Plug-ins	275
9.9.1	Extending Existing Plug-ins	275
9.9.2	Converter Plug-ins	277
9.9.3	Comparator Plug-ins	283
9.9.4	Default Value Plug-ins	285
9.10	Error Reporting	286
9.10.1	Exception Classes	286
9.10.2	Custom Exception Classes	287
9.10.3	Reusing Cúram Error Messages	290
9.11	Java Object Representations	290
9.12	Customization Guidelines	291
9.12.1	Where to Start	291
9.12.2	Custom Formatting	292
9.12.3	Custom Parsing	294
9.12.4	Custom Validation	295
9.12.5	Custom Sorting	297
9.12.6	Custom Error Reporting	301
9.12.7	Custom Default Values	302
9.13	Advanced Topics	304
9.13.1	Type Checking and Null Checking	304
9.13.2	Plug-in Instance Management	304
9.13.3	Naming Conventions	305
9.13.4	Generic Parse Operations	306
9.13.5	Code-Tables	306
Appendix A	Unsupported Dynamic UIM features	308
A.1	Introduction	308
A.2	PAGE	308
A.3	PAGE TITLE	309
A.4	CLUSTER	309
A.5	LIST	309
A.6	FIELD	310
A.7	CONTAINER	310
A.8	ACTION_SET	310
A.9	WIDGET	311
A.10	ACTION_CONTROL	311
A.11	LINK	312
A.12	INLINE_PAGE	312
A.13	MENU	313
A.14	SERVER_INTERFACE	313
A.15	INFORMATIONAL	313
Appendix B	Maintaining Dynamic UIM Pages	315
B.1	Working in a Development Environment	315
B.2	Working in a Running System	317
B.2.1	Search for Dynamic UIM Pages by Category	317
B.2.2	Uploading a Dynamic UIM page to the Resource Store	317
B.2.3	Editing a Dynamic UIM page in the resource store	318
B.2.4	Deleting a Dynamic UIM File from the Resource Store	318
B.2.5	Validating a dynamic UIM file in the resource store	319

	B.2.6 Publish dynamic UIM files	319
Notices	320

Chapter 1

Introduction

1.1 Introduction

This guide is the definitive reference guide for all aspects of the development of Cúram web client applications using the Cúram Client Development Environment for *Java*® (Cúram CDEJ).

The Cúram web client application produces a HTML user interface which is generated by a middle-tier web application. This conforms to the Java EE architecture, in which the Cúram web client application is a HTML user interface driven by JavaServer Pages (JSP) and Servlet technology based on the Apache Struts framework. This HTML user interface makes use of standard browser and Web 2.0 technologies, including JavaScript and Cascading Style Sheets (CSS).

The Cúram CDEJ provides a means of easily developing a HTML client application by reducing the complexity of development associated with web based applications, and insulating the developer from the underlying technologies.

1.2 Prerequisites

A basic understanding of Java EE development environments, XML and Web technologies such as Hypertext Transfer Protocol (HTTP), JavaServer Pages (JSP), Cascading Style Sheets (CSS) and JavaScript is helpful, but not required, before reading this document.

1.3 Companion Guides

Working with the Cúram User Interface acts as a companion guide to this reference manual. It illustrates the application of features outlined in this guide using an example led approach.

In addition a separate reference guide, the *Cúram Web Client Error Message Guide*, lists all messages that can be reported by the Cúram CDEJ development tools at development time and by the web application at runtime, including what they mean, and how they can be resolved.

1.4 Structure

This document is divided into the following chapters:

Chapter 2, *Concepts* introduces Cúram's meta-data driven development paradigm for client applications.

Chapter 3, *Development* describes how, after installing the Cúram Application (*IBM Cúram Social Program Management*), the web client application project is structured, where each type of file should be created, and how to override and extend the default application.

Chapter 4, *Localization* outlines the process of localizing an application into several languages.

Chapter 5, *UIM Reference* is a complete reference for the User Interface Meta-data (UIM) of the Cúram Application.

Chapter 6, *Application Configuration* is a complete reference for the User Interface configuration files of the Cúram Application.

Chapter 7, *Session Management* details how browser sessions are handled by the Cúram application.

Chapter 8, *Domain Specific Controls* details controls that are used to handle specific domain types such as dates, schedules, and calendars.

Chapter 9, *Custom Data Conversion and Sorting* describes a feature that supports the association of custom validation and sorting routines with domain definitions.

1.5 Summary

- This guide is the definitive reference for all Cúram web client development. It should be read with the companion guide, *Working with the Cúram User Interface*.
- The Cúram Client Development Environment (CDEJ) allows the development of lightweight, standards-based (Java EE), portable client applications that can be accessed from a web browser.
- The Cúram CDEJ simplifies the development associated with web based applications by insulating the developers from the underlying technologies.

Chapter 2

Concepts

2.1 Objective

In this chapter you will be introduced to the concepts and terminology used to describe the Cúram Client Development Environment (CDEJ).

2.2 Prerequisites

A basic understanding of Java EE development environments, XML and Web technologies such as Hypertext Transfer Protocol (HTTP), JavaServer Pages (JSP), Cascading Style Sheets (CSS) and JavaScript is helpful, but not required, before reading this chapter.

2.3 Introduction

The goal within the Cúram application is to reduce the complexity of developing web applications by providing mechanisms to generate client screens which define content, layout and navigation. When working with the Cúram CDEJ, a user interface developer can concentrate on the data required on a screen rather than the graphical layout. The CDEJ will generate a standardized user interface from a simple meta data description.

The Cúram user interface comprises of a number of user interface elements that can be combined together. The main element of the interface is a User Interface Meta-data (UIM) page. A UIM page defines the data to be displayed in a page. UIM pages are combined together to provide a view of Cúram known as an application.

In this chapter Section 2.5, *User Interface Meta-data* provides an overview of the User Interface Meta-data used to define a UIM page and Section 2.6, *Applications* provides an overview of the elements that can be combined in an application.

By the end of this chapter you will understand the main concepts that power the Cúram CDEJ to generate a HTML user interface. The concepts defined in this chapter are expanded on throughout the guide.

2.4 Application User Interface Overview

The figure below illustrates an overview of the User Interface meta data in a sample Cúram application page. This sample application page will be re-used elsewhere in the guide, in order to describe how each of the User Interface elements can be configured in an application.

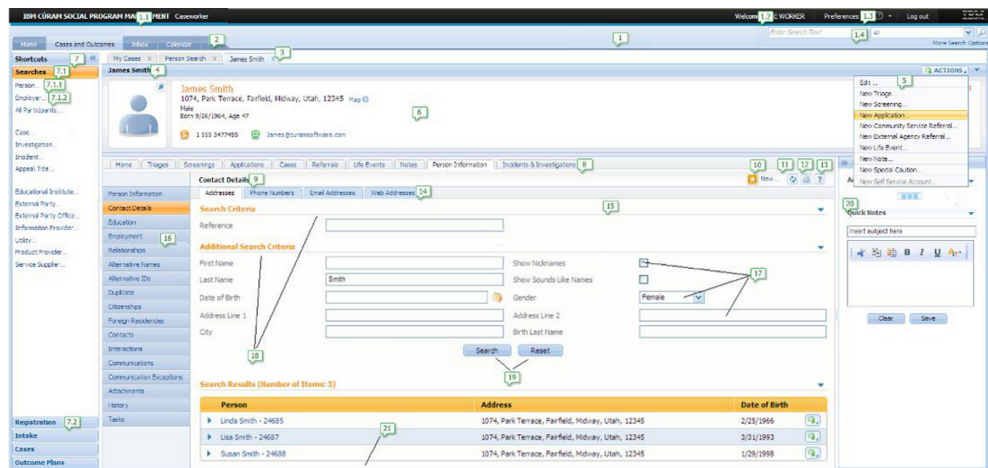


Figure 2.1 Application User Interface Overview

This table describes the mapping between the numbers and User Interface elements referenced in the figure above.

Number	User Interface Element Name
1	Application Banner
1.1	Application Name
1.2	Welcome Message
1.3	Application Menu
1.4	Application Search
2	Application Sections
3	Application tab
4	Tab Title Bar
5	Tab Actions Menu
6	Tab Context Panel
7	Section Shortcut Panel
7.1	Section Shortcut Category
7.1.1	Section Shortcut Menu Item

Number	User Interface Element Name
8	Content Area Navigation Bar
9	Page Title
10	Page Action Control
11	Refresh Button
12	Print Button
13	Help Button
14	In page Navigation Tabs
15	Page Content Area
16	Page Group Navigation Bar
17	Fields
18	Clusters
19	Action Controls
20	Smart Panel
21	List

Table 2.1 User Interface Elements

2.5 User Interface Meta-data

User Interface Meta-data (UIM) is an XML language that describes the contents and layout of one of the main elements in the Cúram user interface, a UIM page.

By limiting the variety of interface layout options available to developers, and by defaulting user interface characteristics based on the known formats of server interfaces, the UIM is kept simple and the user interface layout has an enforced consistency across the whole application.

The developer creates the UIM page definitions in files with a `.uim` extension, with each file corresponding to a single page.

Individual pages are made up from different elements such as page titles, labels, buttons and links as well as the most important element, the data content. UIM focuses on defining elements rather than how they are graphically laid out. The CDEJ provides the tools to generate client screens from UIM definitions.

2.5.1 Page Content Meta-data

The main content area of an application allows server data to be displayed and entered. The basic unit of data is a *field*. Each field is either an output or input parameter of a server interface.

- **Fields.** Fields are visually organized into *clusters* and *lists* on a UIM

page. There may be zero or more of each on a page. Clusters and lists can have a *title* which describes the type of data displayed. There may also be a title for the whole UIM page. Refer to User Interface element 9 in Figure 2.1, *Application User Interface Overview* for an example of a page title.

- **Clusters.** A cluster is a rectangular areas that displays fields in a tabular format. A cluster can have one or more columns of fields, and fields can be displayed with or without an associated *label*. Fields can be read-only, or they may be editable. If editable, they appear as a control such as a text area, drop-down menu, or check-box.

Refer to User Interface Element 18 in Figure 2.1, *Application User Interface Overview* which shows an example of two configured clusters in the page content area - each with a configured title.

- **Lists.** A list is used to display rows of repeating (or *indexed*) fields. As in clusters, fields can have associated labels which are displayed as column headings in the list.

Refer to User Interface Element 21 in Figure 2.1, *Application User Interface Overview* which shows an example of a list in the page content area. The list's title is configured.

- **Action Controls.** Action Controls, displayed as buttons, are used to submit form data, to link to related pages, or to open a modal dialog. Action controls can be organized into *Action Sets* which are associated with clusters, lists, or the UIM page. Individual Action Controls can also be associated with a single field in a cluster or a column in a list. When an action control is used to link to another page it can also send parameters to the target page which are normally used as keys to retrieve server data that populates the target page.

Refer to User Interface Element 19 in Figure 2.1, *Application User Interface Overview* which shows an example of two action controls. These action controls are configured to only appear at the bottom of a cluster but by default Action Controls appear at the top and bottom of the widget they are associated with.

- **Server Interfaces.** A server interface is a method that has been implemented using the Cúram Server Development Environment (SDEJ). See *Cúram Server Developers Guide* and the *Cúram Server Modelling Guide* for more information on developing server interface methods.

The server interface is a non-visual element of a UIM page and each UIM page can be associated with one or more server interface methods. Each method is associated with either the *initialization phase* or the *process phase*. When the UIM page is first opened, the initialization phase methods are executed. Typically an initialization phase method uses *Page Parameters* as input parameters, and the resulting server data is mapped to output fields on the screen.

The Process Phase is initiated when an Action Control of type *Submit* is

selected by the user. Data from input fields on the screen are mapped to input parameters of process phase server methods and the methods are invoked. After execution of process phase methods, the flow of control is determined by the Submit Action, which can specify a link to a new target page, or by the default action which returns to the same page.

Various XML elements correspond to the user interface elements described above—PAGE, FIELD, CLUSTER, LIST, ACTION_CONTROL, ACTION_SET and so on. Other elements such as PAGE_PARAMETER and SERVER_INTERFACE do not have visual representations, but are important to the functionality of the page. The CONNECT element is an important construct that allows fields to be associated with parameters to Server Interfaces. As well as mapping fields, connections can also map page parameters and static text. The latter is not stored directly in the UIM, but is externalized in a property file which facilitates easier language localization of user interfaces.

Example 2.1, *Page UIM Example* contains an extract of UIM used to create the content area. This extract displays how the major elements that make up a screen of content area, such as clusters and lists, are represented in UIM. Chapter 5, *UIM Reference* is a full UIM reference. Refer to User Interface Element 15 in Figure 2.1, *Application User Interface Overview* to see an example of a configured page content area.

```
<PAGE PAGE_ID="Person_search">

  <PAGE_TITLE>
    <CONNECT>
      <SOURCE NAME="TEXT"
        PROPERTY="PageTitle.StaticText1"/>
    </CONNECT>
  </PAGE_TITLE>

  <SERVER_INTERFACE NAME="ACTION"
    CLASS="Person_fo"
    OPERATION="search"
    PHASE="ACTION" />

  <CLUSTER NUM_COLS="2"
    TITLE="Cluster.Title.SearchCriteria">

    <FIELD LABEL="Field.Label.ReferenceNumber">
      <CONNECT>
        <TARGET NAME="ACTION" PROPERTY="referenceNumber"/>
      </CONNECT>
    </FIELD>

    <FIELD CONTROL="SKIP"/>

  </CLUSTER>

  <CLUSTER NUM_COLS="2"
    TITLE="Cluster.Title.AdditionalSearchCriteria">

    <FIELD LABEL="Field.Label.FirstName">
      <CONNECT>
        <TARGET NAME="ACTION" PROPERTY="forename"/>
      </CONNECT>
    </FIELD>

    ... more <FIELD> elements...
```

```

<ACTION_SET ALIGNMENT="CENTER" TOP="false">
  <ACTION_CONTROL LABEL="ActionControl.Label.Search"
    IMAGE="SearchButton"
    TYPE="SUBMIT">
  <LINK PAGE_ID="THIS" />
</ACTION_CONTROL>

  <ACTION_CONTROL LABEL="ActionControl.Label.Reset"
    IMAGE="ResetButton">
  <LINK PAGE_ID="Person_search" />
</ACTION_CONTROL>

</ACTION_SET>
</CLUSTER>

<LIST TITLE="List.Title.SearchResults">
  <FIELD LABEL="Field.Title.Name" WIDTH="44">
  <CONNECT>
  <SOURCE NAME="ACTION"
    PROPERTY="personName" />
  </CONNECT>
</FIELD>
  ... more <FIELD> elements...
</LIST>
</PAGE>

```

Example 2.1 Page UIM Example

2.6 Applications

When a user logs into the Cúram application they are presented with a view that is specific to their role. This view is known as an application. An application in the Cúram user interface is a collection of user interface elements, predominantly based on UIM pages, combined to create specific content for a particular user or role.

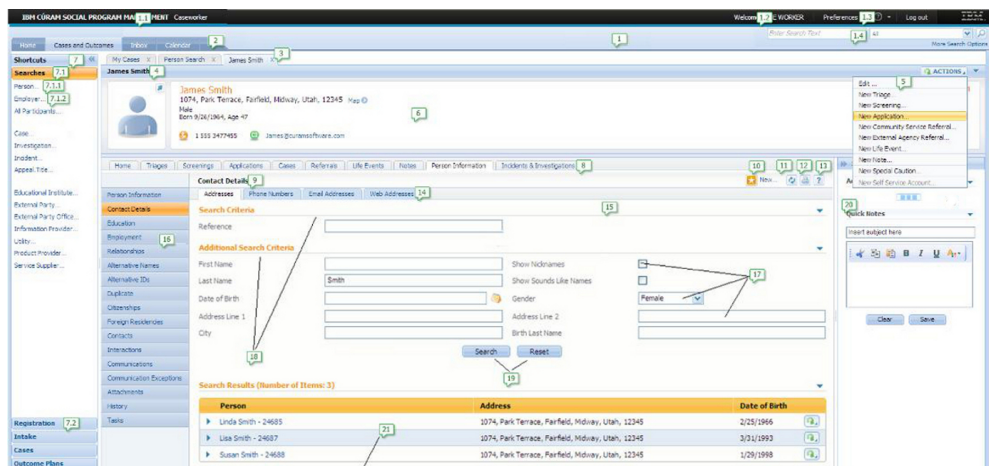


Figure 2.2 Application User Interface Overview

- **Application Banner.** An application is defined to present a specific view of the data for a user or user role. The application banner provides

the user with the context of the application they are currently accessing. Refer to User Interface Element 1 of Figure 2.2, *Application User Interface Overview* for more details of a configured application banner in an application. The banner also include a number of application links, i.e. Help, Logout and Preferences and an application search facility.

- **Application Sections.** An application contains a number of sections , which allow quick and easy access to some of the more common tasks and activities performed by a user. Refer to User Interface Element 2 of Figure 2.2, *Application User Interface Overview* for more details of configured sections in an application.
- **Section Shortcut Panel.** Each section can optionally have a section shortcut panel, which is collapsed by default. When expanded the shortcut panel provides quick links to open content, in the form of UIM pages, and perform actions within the section. The content in the section shortcut panel is organized into categories of menu items. Refer to User Interface Element 7 of Figure 2.2, *Application User Interface Overview* for more details of a configured section shortcut panel in an application.
- **Tabs.** Content in a section is displayed in a tab, and each section can open multiple tabs, where each tab represents a business object or logical grouping of information. A tab can also be described as a logical grouping of UIM pages. Refer to User Interface Element 3 of Figure 2.2, *Application User Interface Overview* for more details of a configured tab in an application.
- **Tab Context Panel.** A tab contains a context panel , which contains context information associated with the data displayed in the tab. This context information is always available when working with the data on the tab. See Refer to User Interface Element 6 of Figure 2.2, *Application User Interface Overview* for more details of a configured context panel in an application.
- **Tab Navigation.** A tab comprises of one or more pages of information, represented by UIM pages. These pages can be navigated using a navigation bar, which contains navigation tabs linking to single pages or sets of pages. Where a navigation tab links to a set of pages, a page group navigation bar is displayed. Refer to User Interface Element 8 of Figure 2.2, *Application User Interface Overview* for more details of a configured navigation bar in an application.
- **Content Area.** The content area displays the currently selected UIM page. Refer to User Interface Element 15 of Figure 2.2, *Application User Interface Overview* for more details of a configured page content area in an application.

In addition to defining the layout of the screen, an application controls the flow between pages available in the application. Within an application, links to other pages are available from a section shortcut panel, the tab navigation bar and page group navigation bar, in addition to links on the page displayed in the content area.

Activating any of these links will result in accessing a new page in the content area, or opening a new page in a modal dialog. For new pages in the content area, the application definition is used to determine what tab the page belongs to and what section the relevant tab belongs to. The page is then opened in the context of the relevant section and tab.

Applications are defined in an XML format using a number of different files. For example, an application is defined using an XML file with the extension `.app`. Each section referenced in the application is defined using an XML file with the extension `.sec` and any tabs referenced by the section are defined using an XML file with the extension `.tab`.

Example 2.2, *Sample Application (.app) File* details an example of an application configuration file (`.app`). The example creates an application containing two sections, in addition to an application banner with a quick search facility.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ac:application
  id="SimpleApp"
  title="SimpleApp.title"
  subtitle="SimpleApp.subtitle"
  user-message="SimpleApp.UserMessage">

  <ac:application-menu>
    <ac:preferences title="preferences.title"/>
    <ac:help title="help.title"/>
    <ac:logout title="logout.title"/>
  </ac:application-menu>

  <ac:application-search>
    <ac:search-pages>
      <ac:search-page type="SAS01"
        description="Search.Person.LastName.Description"
        page-id="Person_searchResolver"
        initial-text="Search.Person.LastName.InitialText"
        default="true" />
      <ac:search-page type="SAS02"
        description="Search.Person.Gender.Description"
        page-id="Person_listByGender"
        initial-text="Search.Person.Gender.InitialText" />
    </ac:search-pages>
    <ac:further-options-link
      description="Search.Further.Options.Link.Description"
      page-id="Person_search" />
  </ac:application-search>

  <ac:section-ref id="SimpleHomeSection"/>
  <ac:section-ref id="SimpleWorkspaceSection"/>
</ac:application>
```

Example 2.2 Sample Application (.app) File

This separation of configuration into multiple files allows for reuse of different elements across multiple applications. For example, a common Inbox section can be defined and referenced by multiple applications. For more information on application configuration consult Chapter 6, *Application Configuration*.

2.7 Page Context

UIM pages are displayed in different contexts within an application. The context the UIM page is displayed in may result in different behavior for some of the elements. The main contexts are outlined below.

- **Content Area.** The content area is where the main content for an application is displayed. When a UIM page is displayed in the content area it will automatically contain a refresh, help and print button¹ within its title bar. Refer to User Interface Element 15 of Figure 2.2, *Application User Interface Overview* to see an example of a configured content area.
- **Context Panel.** A context panel displays a specific kind of UIM page that displays common information for the tab that is always viewable. Refer to User Interface Element 6 of Figure 2.2, *Application User Interface Overview* to see a configured example of context panel.
- **List Dropdown Panel.** A list dropdown panel displays a UIM page when a list row is expanded in a list. Expanded rows are a supported feature of lists. Refer to User Interface Element 21 of Figure 2.2, *Application User Interface Overview* to see unexpanded list items (toggle buttons) in a list. Refer to Section 5.9.23, *LIST* for more information.
- **Modal Dialog.** A modal dialog displays a UIM page in a dialog window, displayed above the main content. While the dialog is open, the parent content cannot be accessed. See Section 5.9.22.3.1, *Using Modal Dialogs* for more information.
- **Smart Panel.** A smart panel, is an optional panel that can be added to the right of the content area in a tab and displays a UIM page. For more information see Section 6.8.2.5, *smart-panel*. Refer to see User Interface Element 20 of Figure 2.2, *Application User Interface Overview* to see an example of a configured smart panel in an application.

2.8 Page “Look-and-Feel”

Just as important to the simplicity of the Cúram client development approach is what you *do not* specify in application and page meta-data. There is very little positioning information for user interface elements:

- the application banner, sections and tabs are in fixed positions;
- clusters and lists flow from top to bottom on a page;
- fields are automatically positioned within them.

Some control is allowed through attributes of the various elements, but sensible defaults are provided for all these attributes to minimize the situations where they have to be used. Refer to User Interface Element 19 of Figure 2.2, *Application User Interface Overview* to see how action controls are

aligned to the center of a cluster. This was achieved with the `ALIGNMENT` attribute of the `ACTION_SET` element in Example 2.1, *Page UIM Example*.

2.9 Application Controller Java Server Page

A single Java Server Page, `AppController.do`, is responsible for rendering the Cúram client on the browser. This application controller JSP is why the URL in the browser is always `AppController.do` and does not change as the user navigates between separate pages within the Cúram application. As a result of this, the back button of the browser is not supported.

It is still possible to request the URL of a specific page in the browser. In this scenario, on receipt of the request, the browser will be automatically redirected to `AppController.do` which loads the requested page. See Section 2.10, *Direct Browsing* for details.

2.10 Direct Browsing

A page can be directly accessed by typing its full URL into the browser's navigation bar, e.g. `http://host:port/Curam/en_US/SomePage.do`. In this scenario the session and its associated tabs will first be restored, then a request will be sent for the specified page. The page will then be loaded in its associated section and tab. However, if this page is not associated with a tab, it will be loaded in the currently selected tab. In the case of a new session, this will be the “Home” tab.

Tabs changed in this manner can be returned to their default state by closing and reopening the tab where possible. For the “Home” tab; logging out and back into the application will restore the “Home” tab to the user's default home page. See Section 7.5, *Tab Restoration* for more information on tab restoration and session management.

2.11 Summary

- Cúram web application development is simplified by describing pages and applications in terms of their content and flow rather than the graphical “look-and-feel” and layout of that content.
- User Interface Meta-data (UIM) consists of definitions in XML format that describe the contents, and to a certain extent the layout, of one of the main elements in the Cúram user interface, a UIM page.
- An application is a collection of user interface elements, predominantly based on UIM pages, combined to create specific content for a particular user or role.
- Graphical layout options available to a developer are restricted to enforce a consistent user interface across the whole application.

Notes

¹The Cúram application does not support the web browser File->Print functionality. A print button is provided for printing the contents of the Content Area only.

Chapter 3

Development

3.1 Objective

This chapter will describe the structure of the Cúram web client application project, including related files in the Cúram server project, and how to develop, build and deploy the application.

3.2 Prerequisites

You should be familiar with the basic concepts of Cúram CDEJ development (see Chapter 2, *Concepts*) and should have some knowledge of the basic format of XML documents. Finally, you should know how to set and edit system environment variables.

3.3 Introduction

The Cúram CDEJ translates files specified in UIM (User Interface Metadata) format into the JavaServer Pages (JSP) that will be deployed on your web application server. These UIM files are supported by various properties files, configuration files, and others. Collectively, these files are called the application's *artifacts*.

Your Cúram web client application project can be divided into various functional components for ease of development. With this system, application changes and updates can be introduced by dropping in a new component that will automatically override the artifacts of another component, where appropriate. The location and purpose of these artifacts and components will be described in detail in this chapter.

3.4 Outline of the Development Process

Much of the client development process is driven by executing specific build scripts. The following is an outline of the typical steps in the process:

1. Install the Cúram Application and the Cúram CDEJ. Directions to the installation guide are provided in Section 3.5, *Installation*.
2. The installer creates both an server application and client application project on your file system containing all the source files. These files will include the application configuration files, the XML-based User Interface Metadata (UIM) for all your pages, any images and other resources that the application requires.
3. Create and edit your source files (UIM and application configuration files) or customize existing files.
4. Deploy your application to an application server. During development, this might be a server embedded in your integrated development environment.
5. Once deployed, you can test your application using a web browser, for example using the following URL:

`http://localhost:9080/'server_name'/AppController.do`

3.5 Installation

To install the Cúram CDEJ, follow the instructions contained in the *Cúram Installation Guide*. The installer will install the Cúram CDEJ and the Cúram Application project ready for further development and customization. The Cúram Application is divided into two major parts: the server application that defines the business entities and business logic of the application, and the web client application that defines how this information is presented to the user.

In this manual, the folders into which parts of the application and the infrastructure are installed will be referred to using placeholders, as the actual locations will vary depending on where they are installed and whether or not you are developing the Cúram Application, additional applications or samples.

Folder Placeholders

<app-dir>

The top-level application folder containing both the server application and the client application.

<client-dir>

The folder containing the web client application. Typically this is a folder called `webclient` within the `<app-dir>` folder.

<server-dir>

The folder containing the server application. Typically this is a folder

called EJBServer within the `<app-dir>` folder.

<cdej-dir>

The folder containing the Cúram CDEJ, the tools and infrastructure required to build and run web client applications. Typically this is a folder called CuramCDEJ.

<sdej-dir>

The folder containing the Cúram SDEJ, the tools and infrastructure required to build and run server applications. Typically this is a folder called CuramSDEJ. More information on this folder can be found in the *Cúram Server Developers Guide*

For example, if you have installed the Cúram Application into the folder `C:/Curam`, then the `<app-dir>` placeholder refers to this folder, the `<client-dir>` placeholder refers to the `C:/Curam/webclient` folder, the `<server-dir>` refers to the `C:/Curam/EJBServer` folder, and the `<cdej-dir>` refers to the `C:/Curam/CuramCDEJ` folder.

3.6 Project Folder Structure

A Cúram web client application project is organized into a folder structure that is recognized by the Cúram CDEJ when the application is built. Example 3.1, *Web Client Folder Structure*, shows an outline of this folder structure for the project and the list that follows describes each folder within this structure in more detail. The base folder of this structure is the `<client-dir>` folder.

```
<client-dir>
+ build
  + bean-doc
+ buildlogs
+ components
  + core
  + <custom>
    + Images
    + javasource
    + WebContent
+ JavaSource
+ project
+ WebContent
  + <locale>
  + Previews
  + WEB-INF
```

Example 3.1 Web Client Folder Structure

Web Client Folders

build

Temporary generated artifacts. The only contents of interest are the generated reference documentation for the façade server interfaces.

build/bean-doc

Generated reference documentation for the façade server interfaces in HTML format. These are regenerated each time the application model

changes. See Section 3.10.7, *Server Interface Reference* for more details.

buildlogs

Log files generated from each build. See Section 3.10.5, *Build Logs* for more details.

components

The top-level folder for the application components. Each sub-folder of this folder contains a separate application component. See Section 3.7, *Application Components* for more information on application components.

components/core

The pre-defined core Cúram application component artifacts that provide the core functionality. These artifacts should not be modified directly. To change them, you should create new artifacts in another component which will then override the core artifacts.

components/<custom>

One or more extra application components containing artifacts that add additional application functionality or customize existing functionality.

components/<custom>/Images

Arbitrary custom resources that you want to deploy with your application. Files and folders within this folder will be copied to the top-level WebContent folder during the build process.

components/<custom>/javasource

Javasource code and properties files used to add extra functionality to an application or to define externalized strings used across many application pages. There are a number of different customizations that can be applied to files within this directory. These include updates to control one or more of the data conversion or sorting operations. Please refer to Chapter 9, *Custom Data Conversion and Sorting* for more details on these customizations. This javasource directory is optional, however if this directory is added, the webclient/.classpath file must be updated to reference this new source directory. This ensures that the changes in this directory are recompiled when a client build is run within the specified development environment. The following is an entry in the webclient/.classpath file, (where <custom> represents the name of a custom directory):

```
<classpathentry kind="src" path="components/<custom>/javasource"/>
```

components/<custom>/WebContent

Arbitrary custom resources that you want to deploy with your application. Files and folders within this folder will be copied to the top-level WebContent folder during the build process.

JavaSource

Contains the CDEJResources.properties file that defines prop-

erties used across many application pages. This file is described in further detail throughout this document. Also contains the `Initial_ApplicationConfiguration.properties` file, that is described in Section 3.11.2, *Configuring the Application*.

project

Configuration files used when customizing the application deployment descriptors. See Section 3.11.3, *Customizing the Web Application Descriptor* for more details.

WebContent

The generated web application files. This contains the generated JSP files and other application artifacts that can be used to start and test an application in the development environment. When an application is to be deployed outside of the development environment, many of the files in this folder are packaged in the application EAR file. See Section 3.11, *Deployment* for more details.

WebContent/<locale>

The generated JSP files for each locale supported by the application are placed in folders named after the locales. For example, for American English pages there will be a folder named `en_US`. These JSP files are generated as necessary when the application is built, so they will be replaced automatically if deleted or out of date with respect to the corresponding UIM file. The JSP files are placed in sub-folders of the locale folder using the first two letters of the page ID as the sub-folder name. This reduces the likelihood that an option provided by some application server software to pre-compile the JSP files will fail when trying to pre-compile too many JSP files at the same time.

WebContent/Previews

Generated HTML files providing a rough preview of what each corresponding JSP will look like when the application is running. These previews can be viewed directly in a web browser without running the application. See Section 3.10.8, *Page Previews* for more information.

WebContent/WEB-INF

The standard folder which must exist in every Java EE web application. No files in this folder will be served by the web container, the files are only used internally by the web client application. It contains a `classes` folder that contains all the compiled Java class files and properties files required by the application. In a Cúram web application project, this includes the classes and properties files from the component specific `javasource` folders and the properties file from the `<client-dir>/JavaSource` directory. It also contains a `lib` folder that contains all required library classes packaged in JAR files. The CDEJ supplies all the JAR files required for this folder and they are copied during the build process. You should not modify any files in this folder.

In addition to the web client folders, there are a number of folders in the `<server-dir>` project that are relevant to web client application devel-

opment. The `<server-dir>` project maintains a similar structure to the web client, specifically in relation to the `component` folder.

Server Folders

components/<component-name>/clientapps

Application configuration artifacts. These are the XML configuration files for defining applications, sections, tabs, etc. For more information see Chapter 6, *Application Configuration*.

components/<component-name>/tab

Application configuration artifacts pre-defined in the Cúram application. XML configuration files shipped with the `core` and other out-of-the-box components will exist in this folder. These should not be modified. To change these you should create new artifacts in the `clientapps` folder in another component, which will then override these artifacts.

3.7 Application Components

3.7.1 Component Folders

Cúram web client applications are organized into collections of artifacts called *components*. Each component has its own folder below the `<client-dir>/components` folder. The `core` component is always present. This contains all of the artifacts needed for the core functionality of the Cúram reference application. The name of the component folder is used as the name of the component.

A component does not necessarily define a discrete part of an application; rather it defines an additional *customization layer* of an application. By adding new components, it is possible to selectively replace pages in the core application, add new pages, change the appearance of the application and alter various settings. It should never be necessary to edit files within the core application, thereby ensuring that when the core application is upgraded, the core changes do not overwrite your custom changes.

Within a component, you can use an arbitrary folder structure to allow you to organize your artifacts as you see fit. Artifacts in a component must have unique file names and the folder structure does not affect this. For example, you cannot place two UIM files with the same name within the same component, even though they would be in different folders. Likewise, a UIM file in one component is considered equivalent to a UIM file in another component, even if the folders within the components containing these UIM files have different names. Technically, a component represents a single namespace for artifacts and the folder structures within the components are mostly ignored.

The only exception to the requirement to use unique file names for artifacts is within the optional `WebContent` folder within a component. Within this

folder, you can place arbitrary files in an arbitrary folder structure that you want to deploy with your application. The files will be copied to the main `<client-dir>/WebContent` folder during the build process and the folder structure will be preserved, so files in different folders may share the same name.

3.7.2 Component Order

There can be any number of application components, but they are processed in a strict *component order*. This order determines the priority that will be given to artifacts that share the same name but appear in different components. This is fundamental to the manner in which Cúram web client applications are customized.

The component order is defined by the `CLIENT_COMPONENT_ORDER` environment variable. This is a comma-separated list of component names. Use only commas; do not use spaces. You must place the component with the highest-priority first in the list and continue in descending order of priority. The `core` component always has the lowest priority and is implicitly assumed to be at the end of the list; you do not need to add it explicitly.

For example, setting the component order to “MyComponentOne,MyComponentTwo” will give the highest priority to artifacts in the `MyComponentOne` folder within `<client-dir>/components`, a lower priority to artifacts in the `MyComponentTwo` folder, and the lowest priority to artifacts in the `core` folder. Any component folder not listed in the component order will not be included in the build and a warning will be displayed to indicate that these components have been ignored. If you do not set the component order at all, the default component order will include all components in alphabetical order.



Note

The `SERVER_COMPONENT_ORDER` order, used for the `<server-dir>` project, will *always* include all component folders existing in the `components` folder. If they are omitted from the `SERVER_COMPONENT_ORDER` environment variable, they will automatically be added to the end of the component order in alphabetical order. For more information consult the *Cúram Server Developers Guide*.

Localized Components

Localized components contains translated artifacts for the base components and are of the format “<component name>_<locale>”. It is *not* necessary for these to be added to the `CLIENT_COMPONENT_ORDER` environment variable as the tooling that processes this environment variable will prepend any available components that match entries in the `LOCALE_LIST` environment variable. Localized components are matched both on complete locale entry and on the two-character, lower-case language code. Localized components

are prepended before the base component in the complete component order.

3.8 Component Artifacts

Components contain a number of artifacts that are used to build an application. All the artifacts in a single component have the same priority in the component order. The artifacts in one component may be used to customize the artifacts in a lower-priority component, or they may be entirely new artifacts that extend the application. The main type of artifacts are as follows:

UIM Pages

UIM pages are the principal artifacts of a web client application. Each UIM page describes a web page that users will see when accessing the web client application with their web browsers. The files for these artifacts use the `.uim` extension.

UIM Views

UIM views define portions of a page that may be re-used by many UIM pages. The files for these artifacts use the `.vim` extension.

Properties Files

Properties files store the natural language text for a page separately from the pages, views and page groups. When applications are localized into different languages, there will be a separate properties file for each language (or *locale*, see Section 3.9, *Application Locales*). This allows a single UIM page, view or page group to be defined for all of the supported languages.



Note

UIM properties files do not support any form of visual layout or formatting capabilities such as using carriage returns or inserting HTML elements.

Application Configuration Files

Application configuration files define the layout of the user interface and how UIM pages are grouped into sections and tabs. The files for these artifacts are defined using the extensions `.app`, `.sec`, `.tab`, `.nav`, `.mnu`, and `.ssp`. Note, these files are located in the `<server-dir>` project. See Chapter 6, *Application Configuration* for details.

Image Files

Images file referenced from your UIM pages or views can be added to your component's `Images` sub-folder. See Section 3.12.5, *Images* for details.

Configuration Files

Configuration files are used to alter the behavior or appearance of the application or of elements of the application. There are a variety of different configuration files that can be used for different purposes.

Custom Resources

Custom resources are arbitrary files that you want to deploy with your application. For example, you may want to customize the appearance of a page to reference your own image file for a logo; this image file is a custom resource.

3.9 Application Locales

A locale describes a user's language, country and determines what the user will see in the pages they access via their web browser. While the data will largely remain the same (other than in the details of the formatting of numbers and dates) the labels for the data will appear in the appropriate language. Locales are specified using a simple identifier that contains a two-character, lower-case language code optionally followed by an underscore character and a two-character, upper-case country code. For example, “en” indicates the English language, and “en_US” indicates the regional variation of the English language appropriate for the United States of America. This regional variation may help to identify differences in the dialect or usage of the language, American English in this example, but it may also affect the way dates and numbers are formatted.

The language and country codes have been standardized and support for any specific locale is determined by the Java Runtime Environment (JRE) that you are using for your application and whether you have localized your application appropriately for that locale. Consult the documentation provided by the vendor of your JRE for details on the supported locales and see Chapter 4, *Localization* for full information on the procedure for localizing a Cúram web client application.

Before building a Cúram application that may have been localized for a number of locales, you need to specify what locales you want to include. To do this, you set the `LOCALE_LIST` environment variable to a comma-separated list of the locale codes. Use only commas, do not use spaces. For example, “en_US,es” specifies the American English locale and the Spanish locale (with no regional variation). The first locale in the list is treated as the *default locale*. Certain operations, such as the generation of page previews (see Section 3.10.8, *Page Previews*), are only performed for the default locale.



Improving Build Performance

The Cúram CDEJ performs most of the translation work for the application's locales during the build process; from a single UIM file it will produce one JSP file for each locale in the locale list. If your application supports many locales, you may find it convenient when developing the application to omit some of the locale codes from the locale list, as this will improve the build performance. You can replace the locales when you want to view or test all of the localized pages.

3.10 Building an Application

3.10.1 Build Targets

The client application is built using *Apache*® Ant build scripts. These build scripts define ordered sequences of processing steps called *targets*. To invoke a target, you open a command prompt window and change to the `<client-dir>` folder and then pass the name of the target to the command you use to start Apache Ant. Typically this command is called **build** or **appbuild**. The name depends on the script provided for your application, but it will be referred to as **build** in this manual. For example, to build the web client application, the command is **build client**. You can run more than one target at a time by passing the target names separated by space characters. For example, **build clean client** will first clean all the generated output that may be present before building the full web client application again.

The following build targets are available for Cúram client projects:

client

Builds the client application. See Section 3.10.3, *Full and Incremental Builds* for further details.

clean

Deletes all of output generated by the other build targets. See Section 3.10.3, *Full and Incremental Builds* for further details.

beandoc

Generates reference documentation for the façade server interfaces. See Section 3.10.7, *Server Interface Reference* for further details.

client-with-previews

Builds the client application and also generates previews of the pages in HTML format in the `<client-dir>/WebContent/Previews` folder. See Section 3.10.8, *Page Previews* for further details.

uimgen

Generates skeleton UIM pages from the façade server interface definitions. See Section 3.10.9, *UIM Generator Tool* for further details.

A number of environment variables affect the build process for a web client application. Some have been introduced already and others are explained elsewhere, but all are shown below. When you install the Cúram Application, the build command will set most of these for you, as they mostly refer to files and folders that will be in fixed locations relative to where you installed the application. However, for a new application, or if you are modifying the build command, you may need to confirm that these are set correctly.

Name	Re- quired	Description
CURAMCDEJ	Yes	The location of the installed Cúram CDEJ infrastructure. This is the same as the value of the <code><cdej-dir></code> placeholder used in this manual. See Section 3.5, <i>Installation</i> for details.
CLIENT_DIR	Yes	The location of your web client application. This is the same as the value of the <code><client-dir></code> placeholder used in this manual. See Section 3.5, <i>Installation</i> for details.
CLIENT_PROJECT_NAME	Yes	Defines the name of the application being built. This name is used as a base name for many generated artifacts, for example, for Java package names. The name is defined in the UML model. For the installed Cúram Application, the value should be “Curam”.
LOCALE_LIST	Yes	Defines the locales that will be supported by the application. See Section 3.9, <i>Application Locales</i> for details.
CLIENT_COMPONENT_ORDER	No	Defines the prioritized order of the application's components. See Section 3.7.2, <i>Component Order</i> for details. This is not required, but it is highly recommended that you set it explicitly. By default, all components will be processed in alphabetical order.
ENCODING	No	Defines the character encoding that will be used to interpret files that do not explicitly define an encoding. By default, the system's default character encoding will be used. See Section 4.5, <i>File Encoding</i> for details.
MULTIPLE_VALIDATION_ERRORS	No	Controls the number of errors that are reported during the

Name	Re- quired	Description
RS		build process before the build terminates. See Section 3.10.6, <i>Error Reporting</i> for details.

Table 3.1 Environment Variables

3.10.2 Related Build Targets

The server application is built using Apache Ant build scripts, in the same way as the client application is built. The application configuration files are located in the `<server-dir>` project and as a result, the targets for processing these are part of the server project. The following targets are used to process the client application configuration files:

inserttabconfiguration

Combines and imports the client application configuration files onto the database. See Section 6.4, *Configuration Files* for more details.

database

The last step of the **database** target is to call the **inserttabconfiguration** target. For more information the database target see the *Cúram Server Developers Guide*.

3.10.3 Full and Incremental Builds

The `client` build target will generate a complete web client application. If no previous build output is present, running this target will build the entire application. This is called a *full build*. Subsequently, on running this target, the build scripts will compare your source files to the previously generated output files to detect what you have changed and will update the minimum number of output files possible. This is called an *incremental build*. An incremental build is performed automatically as long as the output of a previous build is present and is much faster than a full build. To perform a full build again, you must first run the `clean` target to remove all of the outputs from the previous build.



Building after Upgrading

If you upgrade your Cúram application or Cúram CDEJ, you must perform a full build by first running the `clean` target. Failure to do this could result in unpredictable behavior during the build process or when then application is running.



Platform Specific Setting

When executing the `client` build target from a text-only interface (e.g., using a terminal emulator to access a *UNIX*® machine), -

`Djava.awt.headless=true` must be added to the `ANT_OPTS` environment setting.

3.10.4 Dependency Checking

For most changes that you make, you need only run the incremental build, as the changes will be detected automatically and only the dependent output files will be updated. However, some changes are not detected and you may need to run a full build for your changes to take effect. In particular, if you change a setting in the `curam-config.xml` configuration file that affects the build process (typically by affecting the appearance of the pages in a way that is applied at build-time), then you will need to perform a full build manually, as the changes will not be detected automatically.

Dependency checking will identify changes to server interfaces used by UIM pages. Server interfaces are defined in the application's UML model and more information can be found in Section 3.10.7, *Server Interface Reference*. Only changes to interface properties, not their underlying domain types, are recognized in an incremental build. For example, changing a code-table name will not be detected by dependency checking and a clean build will be required.

3.10.5 Build Logs

Every time you run the `client` target to build the application, all of the messages produced by the build scripts are written to a file in the `<client-dir>/buildlogs` folder. The files created are named for the date and time on which the build was started. If errors occur during a build, you may find it easier to review them by reading the log file instead of scrolling through messages at the command prompt.

3.10.6 Error Reporting

One of the main steps performed by the `client` target is the generation of the JSP files from the UIM files. This process will check the validity of your UIM files as they are processed. The validity of the UIM files is determined in a number of steps:

1. They must contain well-formed XML and must not attempt to include VIM files that do not exist.
2. They must conform to the XML schema for UIM and to some additional context-sensitive rules that cannot be defined in the XML schema.
3. They must refer only to externalized strings that exist in their associated properties files.
4. They must meet a number of other requirements related to the connections made to the properties of server interfaces. For example, the property names must be unambiguous, or an address field must be the only field in a cluster.

Normally, the processing will stop when the first error occurs and the indicated problem must be fixed before the build can be executed again. However, for the errors detected in the second step—the schema and schema-related validation errors—there is an option to continue processing as far as possible after an error occurs to allow you to locate and fix more than one error at a time. Errors reported during the other steps will always stop the build immediately.

To allow multiple validation errors to be reported during a build, set the `MULTIPLE_VALIDATION_ERRORS` environment variable to `true`. If not set, the default value is `false` and the build will terminate after the first validation error occurs.

The number of errors reported is limited by the number of UIM files being validated at one time. The validation is typically performed on files in groups of one hundred, so this option will cause all of the validation errors in the current group to be reported before the build is terminated. No further groups will be processed after a group containing files with validation errors has been encountered.

3.10.7 Server Interface Reference

When developing UIM pages, you will need to know details about the façade server interfaces and their properties so that you can select the information that you want to display on each page. This information is all defined in the application's UML model, but, for your convenience, you can generate simple reference documentation in HTML format to make the information more easily accessible.

The `beandoc` target generates this reference documentation for all of the available façade server interfaces (“classes”), creating many HTML files in the `<client-dir>/build/bean-doc` folder. To view the documentation, open the `index.html` file created in that folder in a web browser. This document provides links to alphabetical lists of all classes, all operations on those classes, all domain definitions used by properties of those operations, and all code-tables referenced by any of those domain definitions. Each of these lists provides further links for cross-references or providing more details. Viewing a class will display a list of its operations and selecting an operation will show a list of its properties.

In UIM, you do not have to use the full property name; you can use only part of the ending of the name as long as it is unambiguous. In the reference documentation for each operation, both the full property name and the shortest, unique ending of the property name are given. This will help you to choose a name that is short and readable, but that will not cause any build errors later.

Beside many of the class, operation, and property names, you will see a *Copy* button. Clicking this button will copy the name to the clipboard, allowing you to paste it into your UIM file. For property names, the shortest unique name is copied. Copying to the clipboard using the *Copy* button only

works in *Microsoft®Internet Explorer*. In other browsers, you will have to select the text and use the normal copying commands.

3.10.8 Page Previews

Page previews are produced by running the `client-with-previews` build target. This will generate static HTML pages for the default locale that can be opened in a browser to give you an impression of what the page will look like when the application is running. The HTML pages are located in the `<client-dir>/WebContent/Previews` folder. You do not need to start a server to view the pages. The pages display a default value for each field but do not support any user-interaction (buttons, links, pop-ups, etc. do not function). The preview page represents only the main content area of the page (the part specified in UIM) and not the sidebar or page header or footer.

The default values for the fields are defined by associating a default value with the domain definition of the field. These default values are used only for the preview pages and are defined in the `domain-defaults.xml` file in `<client-dir>/components/core`. Overriding this file in other components is not currently supported so it must be modified in place.

The file uses a simple XML format, a sample of which is shown below. The root element is `DOMAIN_DEFAULTS`. This element contains one `DOMAIN` element for each domain definition for which a default value is to be defined. The `DOMAIN` element requires a `NAME` attribute specifying the domain name, and a `DEFAULT` attribute specifying the default value for that domain.

```
<DOMAIN_DEFAULTS>
  <DOMAIN NAME="MY_DOMAIN" DEFAULT="My value"/>
  <DOMAIN NAME="YOUR_DOMAIN" DEFAULT="Your value"/>
</DOMAIN_DEFAULTS>
```

Example 3.2 Default Preview Values for Domain Definitions

When generating preview pages, if there is no default value defined for a domain, a warning message will be displayed. These warnings will not prevent the preview page from being generated and a fall-back value will be used in the generated page (for example, “[field-value]”). Note that fields that have a complex domain value are not parsed or processed in the normal manner. Most of these are simply replaced by an image of the typical output and no default value is required. Complex fields like this are described in Chapter 8, *Domain Specific Controls*.

3.10.9 UIM Generator Tool

The UIM Generator tool provides a user interface for automatically generating a UIM page for a particular server interface.

To start the *UIM Generator* tool:

1. Open a command prompt and change to the `<client-dir>` folder.
2. Run **build uimgen**.
3. The first time you run the UIM Generator you will be asked to locate a `ServerAccessBeans.xml` file. This file is generated by the `client` target and can be found in the `<client-dir>/build` folder.

Once the *UIM Generator* has started, you should see a screen containing the following:

- A *File* menu containing options to view your current configuration settings and to exit the application.
- A tree on the left hand side which lists all the server interfaces in the application.
- Two options, *Display Phase* and *Action Phase*, which determine when the selected server interface is called in the generated page.
- A *Make Page* button which generates the UIM for the current settings.

To generate a page perform the following:

1. Select the interface you wish to test from the tree (e.g. *Register-Person.read*).
2. Select the phase in which the interface should be called, for example, *Action*. Action phase pages call the interface when the page is submitted. Data can be entered for each input field and a button is generated to submit the page.
3. Click the *Make Page* button and you will be asked to specify a location for the generated UIM. You can change the default name if you wish. The location should be in the appropriate component folder of your application.

A UIM file and a properties file are generated. The labels for each field are given defaults based on the name of the server interface property associated with the field.

3.10.10 External Client Applications

Due to the webclient directory containing a mix of components that are targeted for different EAR packaging, it can be difficult to use the single development environment and component order to develop and test these.

To allow for this a build target `external-client` will allow for creation of an environment and building of the components specified for an EAR entry in the `deployment_packaging.xml`.

The target requires a parameter `-Dapp` which should refer to the name of an EAR entry within the `deployment_packaging.xml`.


```
build external-client -Dapp=SamplePublicAccess
```

Example 3.3 external-client invocation

The build target will copy the components specified for this EAR entry to a `webclient\build\apps\<app name>` directory and here will both build the project and create the relevant Eclipse project configuration files to allow for the project directory to be imported into Eclipse and development-type testing to be performed on these external client applications.

3.11 Deployment

3.11.1 Overview

A detailed description of the deployment procedure is provided in the *Cúram Deployment Guide* appropriate for your application server and operating system. However, there are a number of configuration settings available in your web client application project prior to deployment. These settings are described below.

3.11.2 Configuring the Application

The `ApplicationConfiguration.properties` file defines the most important application configuration settings. The file should be located in the `curam/omega3` sub-folder of the `<client-dir>/JavaSource` folder. When you create a new application, this folder will contain a sample file named `Initial_ApplicationConfiguration.properties`. You should copy this file and rename it to `ApplicationConfiguration.properties` and change the settings to match your requirements. For the installed Cúram Application, this will be already be done for you, but you may still want to make some changes.

The properties that may be set in this file are as follows:

dateformat

Example: `dateformat=M d yyyy`

The application-wide date format used when displaying dates or when parsing dates entered by a user. This specific format (per user) is not supported within the Cúram application.

The value of `dateformat` can be set to any one of a number of pre-defined formats. Formats in day-month-year order: “d M yyyy” (the default), “d MMM yyyy”, “d MMMM yyyy”, “dd MM yyyy”, “dd MMM yyyy”, “dd MMMM yyyy”. Formats in month-day-year order: “M d yyyy”, “MMM d yyyy”, “MMMM d yyyy”, “MM dd yyyy”, “MMM dd yyyy”, “MMMM dd yyyy”. Formats in year-month-day order: “yyyy M d”, “yyyy MMM d”, “yyyy MMMM d”, “yyyy MM dd”, “yyyy MMM dd”, “yyyy MMMM dd”.

In these predefined formats, “d” represents the day number, “dd” represents the two-digit day number padded with a leading zero if necessary, “M” represents the month number, “MM” represents the two-digit month number padded with a leading zero if necessary, “MMM” represents the abbreviated month name, “MMMM” represents the full month name, and “yyyy” represents the four-digit year. An upper-case letter “M” is used for the month, as the lower-case letter “m” is used in Java applications to represent the minute value when formatting times. The formats are specified using a space character as a separator. The actual separator character that you wish to use is specified separately.

dateseparator

Example: `dateseparator=/`

The value of `dateseparator` can be set to one of “.”, “;”, “/”, or “-”. The date separator character that will be applied to the specified date format. The value can be set to any one of a number of predefined separator characters: “/” (the default), “.”, “;”, or “-”.

timeformat

Example: `timeformat=HH mm`

The value of `timeformat` can be set to one of “h m s a”, “h m a”, “H m”, “hh mm a”, “HH mm”, “hhmm a” or “HHmm”. Where not specified, “HH mm” is used as the default.

timeseparator

Example: `timeseparator=:`

The value of `timeseparator` can be set to one of “:” or “.”. Where not specified, “:” is used as the default.

serverConnectionType

Example: `serverConnectionType=single`

Do not change this value.

addressFormatType

Example: `addressFormatType=US`

Default address format for addresses in the application.

addressDefaultCountryCode

Example: `addressDefaultCountryCode=US`

Default, application-wide country code for addresses. This must match an entry on the server application's Country code table.

uploadMaximumSize

Example: `uploadMaximumSize=-1`

Maximum file upload size in bytes. Files that exceed this size will be rejected. This should be set to match the allocated storage in the database for fields containing uploaded files. This cannot be tailored to suit different database fields. The value `-1` indicates no maximum limit.

uploadThresholdSize

Example: `uploadThresholdSize=1024`

The maximum size in bytes of an uploaded file before a temporary file will be created on the server to reduce the memory overhead of storing the data as it is being processed. By default, uploaded files are written to temporary disk storage if they exceed 1024 bytes.

uploadRepositoryPath

Example: `uploadRepositoryPath=c:/temp`

Temporary files created during file upload will be written to this location if they exceed the upload threshold size. By default files will be written to the Java system temporary folder (as defined by the Java system property `java.io.tmpdir`).

use.synchronizer.token

Example: `use.synchronizer.token=true`

Whether to use a synchronizer token to prevent accidental re-submission of forms due to use of the browser's *Back* button. Can be set to `true` (default) or `false`.

synchronizer.token.timeout

Example: `synchronizer.token.timeout=1800`

A synchronizer token will expire if its associated form is never submitted. Values are specified in seconds. The default value for this property is 1,800 seconds.

errorpage.stacktrace.output

Example: `errorpage.stacktrace.output=false`

The value for this property is `true` or `false`, with `true` as the default.

Stacktrace output is used in the development environment for debugging purposes. When the value for this property is `true`, the Java exception errors are output into the HTML error pages.

The property must be set to `false` in a production environment, e.g. `errorpage.stacktrace.output=false`, otherwise it will introduce security vulnerabilities into the application. The HTML error pages, which contain the Java exception stack trace, are not subject to the Cúram's application malicious code and filtering checks and will potentially leave the application open to injection attacks, e.g. Cross-site scripting and link injection.

dbtojms.credentials.getter

Example: `dbtojms.credentials.getter=curam.sample.CredentialsGetter`

Specifies the name of the class used to obtain credentials to be used for triggering a DBtoJMS transfer. If not specified, a default set of creden-

tials will be used for this operation. For more information about DBto-JMS and using this property please see section entitled 'Security Considerations' of the *Cúram Batch Processing Guide*.

modal.dialogs.minimum.height

Example: `modal.dialogs.minimum.height=200`

Specifies the minimum required height for a modal dialog in pixels and will be used when the calculated height of the modal dialog is less than the minimum required height or the specified height is less than the minimum required height. The default value of 100 pixels applies if this is not set.

tabSessionUpdateCountThreshold

Example: `tabSessionUpdateCountThreshold=10`

Specifies the number of tab session data updates that must be received before the data is persisted from the web tier to the database. Once the threshold is reached, the recent updates are written and counting starts again from zero until the threshold is reached. A value of one causes writes on every update. A value of zero (or a negative or invalid value) disables writing based on update counts.

The default is every 10 updates.

For more information consult Chapter 7, *Session Management*.

tabSessionUpdatePeriodThreshold

Example: `tabSessionUpdatePeriodThreshold=120`

Specifies the number of seconds that must have elapsed since the last time session data was persisted from the web tier to the database before a new update will trigger another write. A value of zero (or a negative or invalid value) disables writing based on update periods.

The default value is 120 seconds, or 2 minutes.

For more information consult Chapter 7, *Session Management*.

resourceCacheMaximumSize

Example: `resourceCacheMaximumSize=16000000`

Specifies the size of the application resource store cache. By default, the cache is limited to 16MB (approx.) in size. When that limit is reached, the least recently used resources will be ejected from the cache to make room for newly requested resources that are not already in the cache. The size of the cache is specified in bytes.

Note: If a single resource exceeds the size limit for the cache, it will not be cached.

dynamicUIMInitModelOnStart

Example: `dynamicUIMInitModelOnStart=false`

Indicates if the Dynamic UIM system should initialize the required information on the application model during startup or when it is first re-

quired for a Dynamic UIM page. The default value is `true` and it should be set to `false` to cause the model to be initialized when it is first required by a Dynamic UIM page.

See Section 5.12, *Dynamic UIM System Initialization* for more detailed information.

sanitize.link.parameter

Example: `sanitize.link.parameter=true`

Enables protection from link injection attacks. The default value is `false`.

When the value of this property is `true`, any parameters in the request URL containing links to content with the Cúram application are validated using a regular expression. The validation ensures that a third party hasn't replaced the link value with a malicious link to an external site.

Tracing

As described in Chapter 4, *Localization*, the file `webclient\JavaSource\curam\omega3\i18n\CDEJResources.properties` defines properties for localizing certain features of the application. It also contains the setting to enable tracing of server function calls on the web-tier. Add the following property to enable this tracing:

```
TraceOn=true
```

When enabled, the inputs to and outputs from all server function calls will be written to “Standard Out”¹.

3.11.3 Customizing the Web Application Descriptor

The web application descriptor—defined in a file named `web.xml`—is a standard Java EE web application file. A Cúram web application contains various settings that a developer may wish to change, for example, server connection settings and the session time-out. The default settings can be seen in the following files based on the environment you are running the application from:

Development Environment

```
<cdej-dir>/lib/curam/web/WEB-INF/web.xml
```

IBM® WebSphere® Application Server

```
<cdej-dir>/ear/WAS/war/WEB-INF/web.xml
```

WebLogic® Application Server

```
<cdej-dir>/ear/WLS/war/WEB-INF/web.xml
```

Customizing the `web.xml` file is done differently depending on whether you are changing the version of the file to be included in the Cúram EAR file or the version to be used at development time (e.g. in Apache Tomcat).

Customizing the `web.xml` for development time can be done by creating a

custom version of the `web.xml` file in the `WebContent/WEB-INF` directory of a particular component, e.g. `custom`. Where multiple versions of `web.xml` exist in different components, the version in the highest precedence component, based on `CLIENT_COMPONENT_ORDER`, will be used.

The `web.xml` used within a Cúram EAR file can be customized using the `deployment_packaging.xml` file located in the `Curam Server project/config` directory. It is possible to specify a custom `web.xml` using the `custom-web-xml` property. For more information on customizing `web.xml` at runtime please consult the Cúram Deployment Guide for the relevant Application Server.

When customizing `web.xml`, the existing security, filter and servlet settings should not be modified.

The server and port settings in `ApplicationConfiguration.properties` are now obsolete and no longer need to be specified. They are now automatically configured as `context-param` elements in `web.xml` when the Cúram EAR file is created. The server and port values are set according to the values specified in the `AppServer.properties` files (see the Cúram Server Deployment Guides for more information), with the exception of the `web.xml` used at development time. The development `web.xml`, located in `<cdej-dir>/lib/curam/web/WEB-INF/web.xml`, has the server and port set to `localhost` and `900` respectively.

To change or add a locale, locate the `init-param` elements of the `ActionServlet` and duplicate them, changing the value of the `param-name` element as appropriate so it is in the form `config/<locale-code>`. See the example below.

```
<init-param>
  <param-name>config/en</param-name>
  <param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
```

Example 3.4 Configuring an Application Locale

By default the `web.xml` for both *WebSphere* and *WebLogic* application servers is configured to enforce secure http (https), i.e. a secure SSL connection between the web client and the server. This can be modified by changing the `transport-guarantee` from `CONFIDENTIAL` to `NONE`. Note, this does not disable access to the Cúram web client over https, but enables additional access via http. Please refer to the Curam Security Handbook for further details.

Customizing the 404 or Page Not Found error response.

The 404 or Not Found error message is a HTTP standard response code indicating that the client was able to communicate with the server, but the server could not find what was requested. The default `web.xml` files for WebSphere, and WebLogic specify a default error page for the Cúram application when an HTTP 404 error is thrown by the application server. The

following is the error message displayed on that default page:

- The page you have requested is not available. One possible cause for this is that you are not licensed for the necessary Cúram module - if that is the case, you can use the User Interface administration screens to remove these links.

This message may be customized by adding a `HTTP404Error.properties` file into the `webclient\JavaSource\curam\omega3\i18n` folder of the application and overriding the `error.message` property specified in that file.

3.12 Customization

3.12.1 Overview

A Cúram web client application can be customized without modifying the original components or their artifacts. This makes it easier to upgrade a base application while preserving your custom changes to that application. In this section you will see how the customization process works and how you can modify or extend a base application.

Customizations are applied according to the component order. The changes that you make to customize an application should be made in a separate component from the application's original components. The Cúram Application will be installed with a number of components (the core component and a number of other add-on components). To make customizations, create a new component folder—a new sub-folder in the folder called `components`—and add that component's name—the folder name—to the component order (see Section 3.7.2, *Component Order*). You will always want to add your component name to the beginning of the component order to give it the highest priority when artifacts are being selected at build-time. You can add more than one custom component, but you must decide what their relative position in the component order should be.

To begin with, your custom component will be an empty folder. You make your customizations by adding artifacts (e.g., UIM pages, configuration, files, etc.) to this component folder. You can create arbitrary sub-folders to help you organize these artifacts. You can customize an application by adding new artifacts, *overriding* existing artifacts, or *merging* new content with existing artifacts.

3.12.2 Adding New Artifacts

You can add new artifacts to extend a base application. To add a new artifact, you simply create the new file in your component folder. The file name of the artifact should not be the same as the file name of an artifact in another component. If it is, the artifact will override another artifact or be merged with one. All types of artifacts can be added to an application in this man-

ner, note artifacts added to the WebContent sub-folder will always override other delivered artifacts, as described in Section Section 3.12.14, *Custom Resources*.

3.12.3 Overriding or Merging Artifacts

Some types of artifacts can be overridden (effectively replaced) by adding an artifact with the same file name as an artifact in another component to your custom component. When building the application, the artifact in the highest priority component will be selected and the others ignored. Not all types of artifacts are overridden so completely. Other types of artifacts are merged with the same named artifacts in the lower priority components. The content of all of the artifacts is combined and, where the content is related, the content from the highest priority component is selected. The customized artifacts only need to share the same file name, they do not have to share the same relative folder location, though you may find it advantageous to organize them in a similar manner.

For example, for UIM files that share the same name, the file in the highest priority component will be selected and the others ignored; but for properties files that share the same name, all of the properties are merged together and, where the files contain properties with the same key name, the value of the property from the file in the highest priority component will be used. When building an application, the artifacts in the components are not modified. The selection and merging of artifacts is performed in temporary locations, leaving the original artifacts intact.

The different ways in which artifacts are merged or overridden is covered in the sections below.

3.12.4 Externalized Strings

All string values in UIM documents and JavaScript must be externalized. This aids maintenance and allows the application to be localized. JavaScript, UIM pages and UIM views can reference externalized strings.

The syntax of a properties file is simple. Each line contains a *name=value* pair, where the name is an arbitrary name for the string (it should not contain the “=” character), and the value is the localized string value. Blank lines and lines beginning with a “#” character are ignored. Example 3.5, *A Sample Properties File* contains an example. The syntax is defined by the `java.util.Properties` class provided with your Java Runtime Environment; you can consult the API documentation for that class for more details.

It is worth noting that the property value will be reproduced in the final application page exactly as you have typed it in the properties file. The value can contain any character from any language and it does not matter if that character is reserved in XML, HTML or anywhere else—it will be safely processed and displayed as you intended in the application.

If you find that you need to enter a character in a property value that you cannot generate from the keyboard, the only one way to do it is to use the Unicode value of that character in a *Unicode escape sequence*—a backslash and a “u” followed by the four-digit hexadecimal character code. For example, if you want to enter a non-breaking space, the corresponding Unicode escaped sequence is “\u00a0”. An example of this is included in the sample properties file below.

```
# Main Titles
MyPage.Title=My First Page
Cluster.User.Title=User Details

# Field labels
Field.FirstName.Label=First Name
Field.Surname.Label=Surname

# Other
Separator=\u00a0
```

Example 3.5 A Sample Properties File

As you can see, using “.” characters is a useful way to add some structure to the properties in the file, though it is not a requirement.

When customizing an application, you can customize properties independently of pages and views by adding the appropriately named properties file to your custom component and defining the externalized string properties. You do not need to add the corresponding page or view file to your component and you do not need to redefine any of the properties that you do not want to change.

3.12.5 Images

All references to icons or other graphics within a UIM document are externalized in a manner similar to normal strings. The `Image.properties` file (you can include one in each component, if you wish) uses the same format as the string properties files to associate image references with image file names. The image files should be stored in the component's `Images` sub-folder and can be organized into a folder structure below this folder if desired. Most web browsers will support images in the portable network graphics (PNG) format, the graphics interchange format (GIF), and the joint photographic experts group (JPEG) format.

The `Image.properties` file simply associates a key with a path to the corresponding image file specified relative to the component folder. A sample of this file is shown below. To use these images, the key is used as the value of the `IMAGE` attribute on the `ACTION_CONTROL` element in the UIM page.

```
Button.Ok=Images/ok.gif
Button.Cancel=Images/cancel.gif
MyPage.Title.Icon=Images/bluedot.gif
```

Example 3.6 A Sample `Image.properties` File

The entries in the `Image.properties` file in the core component can be overridden individually or in total by creating an `Image.properties` file in your custom component and overriding the properties as required. You can override the image files themselves by creating files in your custom component with the same names as the files in the core component.

If you need to localize your images for different languages, you can add several `Image.properties` files using a different locale code as the file name suffix. See Section 4.6, *Locales* for details on locale code suffixes. Each properties file should define the same keys, but the image files can be different for each locale. If only some of the images need to be localized, the common images can be defined in the default `Image.properties` file (the one without the locale code suffix) and only properties for the localized images in the other properties files.

3.12.6 Image Mapping

Images can also be used within the Cúram application to represent different values of displayed fields instead of presenting the value as text. For example, a typical boolean value of `true` or `false` could be represented by two images of, say, a green check mark and a red X.

The mapping between values and images is stored in the `ImageMapConfig.xml` file. There is no need to specify this in any way in UIM. If you use a property with a domain listed in the `ImageMapConfig.xml` file, it will automatically be displayed as an image.

```
<map>
  <domain name="MY_BOOLEAN">
    <locale name="en">
      <mapping value="true"
        image="Images/ValuesToImages/true.gif"
        alt="True"/>
      <mapping value="false"
        image="Images/ValuesToImages/false.gif"
        alt="False"/>
    </locale>
    <locale name="fr">
      <mapping value="true"
        image="Images/ValuesToImages/true.gif"
        alt="Vrai"/>
      <mapping value="false"
        image="Images/ValuesToImages/false.gif"
        alt="Pas Vrai"/>
    </locale>
  </domain>
</map>
```

Example 3.7 A Sample `ImageMapConfig.xml` file

In the example, a field with domain type `MY_BOOLEAN` has been assigned an image mapping. Note that you should specify an image mapping for each available locale even if the images used are identical. This is because the alternative text (“alt text”) attached to the image will be different for different locales. This text is important for accessibility reasons (users who have visual difficulties might use an audio browser, for example, which will read out the “alt text”).

ImageMapConfig.xml files in different components are merged with all unique image mappings preserved. If the same value in the same locale is mapped in two ImageMapConfig.xml files in two different components, the mapping from the higher priority component prevails.

3.12.7 CuramLinks.properties

The UIM LINK element allows links to other client pages to be specified indirectly. The PAGE_ID_REF attribute is a key into the CuramLinks.properties file that returns the actual ID of the linked page.

Many links can point to the same page reference. The advantage of using a page reference is that all the links can be updated by changing a single entry in this file.

Each component can have its own CuramLinks.properties file. During generation, these individual files will be merged. As usual, if a particular key is present in more than one CuramLinks.properties file, the component priority order is used to decide which value is retained.

3.12.8 XML Runtime Configuration Files

There are a few miscellaneous XML files that are used by the running client application. To change any of these files, copy the original file into the custom component sub-directory and modify the copied file. The default files can be found in <cdej-dir>/lib.. The client generators will use the xml file from the highest priority as specified by the CLIENT_COMPONENT_ORDER environment variable. The following is a list of these files:

- CalendarConfig.xml
- DynamicMenuConfig.xml
- ICDynamicMenuConfig.xml
- MeetingViewConfig.xml
- RatesTableConfig.xml
- RulesDecisionConfig.xml
- RulesEditorConfig.xml

Further details on the customization of these configuration files are given in Chapter 8, *Domain Specific Controls*.

3.12.9 Login Pages

A default login page is supplied, called logon.jsp and located in the lib/curam/web/jsp directory of the Cúram Client Development Environment. This can be overridden by placing a copy, with the required

changes, in a `webclient/components/<custom component>/WebContent` folder. However, there are some guidelines that should be followed.

Firstly, the following JavaScript should be included in the head section of the page:

```
<jsp:include page="no-dialog.jsp"/>
<script type="text/javascript"
  src="${pageScope.path1}/CDEJ/jscript/curam/util/Logon.js">
  //script content</script>
<script type="text/javascript">
  curam.util.Logon.ensureFullPageLogon();
  function window_onload() {
    document.loginform.j_username.focus();
    return true;
  }
</script>
```

This prevents the login page from being loaded in a dialog window.

Secondly, if it is desired to use the `j_security_check` login mechanism, the form submitted from the page should have an `action` attribute of `j_security_check`, a user name input with the name attribute `j_username` and a password input with the name attribute `j_password`.

The *Cúram Server Developers Guide* contains details of some common customizations to the `logon.jsp` file to support an external user client application and automatic login.

The styling of `logon.jsp` can be customized in the usual way. Simply add relevant CSS to any `.css` file in the custom component.

3.12.10 JavaScript Files

The UIM `SCRIPT` element allows events on the page to trigger JavaScript functions. You can simply provide a path to the JavaScript file that is relative to your component folder. For example, if you have a JavaScript file in a sub-folder of your component folder: `MyComponent/scripts/myScript.js`, you can just refer to this in the `SCRIPT` tag as follows:

```
<SCRIPT SCRIPT_FILE="scripts/myScript.js" ...>
```

The paths you have specified will be fully preserved during application generation.

JavaScript allows HTML and CSS to be queried and manipulated. The underlying HTML and CSS source code used to style the Cúram application is not documented. No guarantees are made about its stability across Cúram releases. Therefore, custom JavaScript may have to be updated in line with changes to HTML structure.

A number of JavaScript APIs for use in the custom JavaScript code are provided within the Cúram application. They are documented in the following location in your CDEJ installation: `Curam-CDEJ\doc\Javascript\index.html`. Use of any other Cúram

JavaScript APIs, discovered through web developer tools for example, is not supported. The same is true of the JavaScript APIs and functions of third party frameworks used within the Cúram application. While there is nothing prevent a developer using these, using them means the code will be impacted by changes to the Cúram application in future releases.

Using the techniques described above to add new JavaScript files to the custom component, new third party APIs could be added to Cúram pages. This is at the customers discretion, as no guarantees can be made on third-party APIs that have not been used and verified within the Cúram application.

3.12.11 Cascading Stylesheets

Stylesheets (*.css) define the appearance (colors, fonts, etc.) of the client pages when viewed in a web browser. Default stylesheets are provided for the Cúram client application. It should never be necessary to edit these files, you can view them in the `WebContent/WEB-INF/css` folder. Instead, you can override particular styles or add new styles by creating new CSS files in one of your application components. Any CSS file located in the `component/<some-component>` folder (or sub-folder) will be automatically concatenated into the `custom.css` file. The `custom.css` file is included on all pages in the Cúram client application.

The underlying HTML and associated CSS used to style the Cúram user interface can easily be viewed in a variety of ways, such as using developer tools like the Internet Explorer Developer Toolbar. An example of customization would be to view the CSS used to apply a color to a field's label. The same CSS style can then be added to your custom CSS file and a different color specified. For example, assuming the HTML and CSS has been analyzed and the CSS rule `.field .label` applies the label color, the following CSS could be used to override the default:

```
.field .label {
  color: red;
}
```

This will take precedence over the Cúram style because custom CSS is included on the page after Cúram's default CSS. Another customization technique would be to create a new rule that is an extension of a Cúram rule. Continuing the above example, a developer analyzes the HTML and sees that within the Cúram application a span element is generated as a child of the `.label` element. It is possible to create a new rule that is specific to this span, even if Cúram has not done so. The complete customization will now look like this:

```
.field .label {
  color:red;
}
.field .label span{
  color:blue;
}
```

The underlying HTML and CSS source code used to style the Cúram user

interface is not documented (hence the use of developer tools to view it). No guarantee is made about its stability across Cúram releases. Therefore, customizations as described above or any customization based on analysis of the Cúram application's underlying HTML and CSS may be lost as new releases are taken on. The customizations may have to be re-applied by analyzing the HTML and CSS again.



Note

Some UIM elements support the `STYLE` tag which allows specific styling to be added to any instance of that element. This styling will always override that included in `.CSS` files. For more information, see Chapter 5, *UIM Reference*.

Application Specific CSS

CSS can be specific to the application being viewed. The `id` of the application (`.app` file) currently being viewed is added as a class on the `BODY` element of each HTML page, allowing application specific styling to be added to that page.

For example, a System Administrator views the `SYSADMAPP` application. The following is an example of CSS specific to that application:

```
.SYSADMAPP .field .label {
    color:red;
}
```

Media Specific CSS

CSS can be specific to the type of media being used to view the web page. So, for example, it is possible to have some styles that only apply when a page is printed and others that only apply on-screen. It is possible to include CSS specific to a media using the following pattern:

```
<STYLE type="text/css">
@media print {
    BODY {font-size: 10pt; background: white;}
}
@media screen {
    BODY {font-size: medium;}
}
</STYLE>
```

Browser Specific CSS

CSS can be specific to the browser used to view the web page. Internet Explorer specific CSS files can be created in any folder in a component. A naming convention is used to distinguish between versions of Internet Explorer. Specifically the following suffixes are to be used:

- `ie.css` This file will be included in all versions of Internet Explorer.

- `_ie6.css` This file will be included in Internet Explorer 6.
- `_ie7.css` This file will be included in Internet Explorer 7.
- `_ie8.css` This file will be included in Internet Explorer 8.

Please note that developers should continue to strive for using the same CSS on all browsers. Internet Explorer specific styling should only be used as a last resort.

3.12.12 Application Configuration Files

The application configuration files for defining application, section and tabs can be added to the `<server-dir>\components\<component-name>\clientapps` directory, where `<component-name>` is a custom component. Sub-folders are supported within the `clientapps` folder. Any artifacts added to this directory will override files of the same name in the `<server-dir>\components\<component-name>\tab` directory. The `tab` directory contains files that are shipped with existing components within the Cúram application and these files should not be modified.



Note

The OOTB Cúram application uses fragments of configuration artifacts that are merged into single files at build time, this is not supported for custom application configuration artifacts. (i.e.) you should not have a `tab` folder in `EJBServer\components\custom`.

When customizing application configuration files that ship with the Cúram application, the XML configuration file and `.properties` file should always be customized as a unit. For example, a change to the `SimpleApp.properties` file, associated with the `SimpleApp.app` file, should result in adding both `SimpleApp.app` and `SimpleApp.properties` to the `clientapps` folder. These files should be based on the merged version of the files. The **inserttabconfiguration** target can be used to get a development copy of the merged file. See the *Cúram Server Developer Guide* for more information.

There are a few general rules and best practices when working with the application configuration files:

- The `id` attribute on the root element of each configuration file must match the name of the file. E.g. `SimpleApp.app` must have an `id` of `SimpleApp`.
- The `id` attributes should not contain the period (`.`) or underscore (`_`) characters.
- Localizable text should be added to a `.properties` file which matches the name of the configuration file. E.g. `SimpleApp.app` will have a corresponding `SimpleApp.properties`.

- Properties files can be re-used across configuration files. E.g. `Person.nav` and `Person.tab` can share the same `Person.properties` file.
- Ensure when developing the XML files to add the proper namespace information. This will allow for validation. For example:

```
<ac:application
...
</ac:application>
```

3.12.13 General Configuration

Overview

The `curam-config.xml` file contains a number of general-purpose configuration options that affect the appearance or behavior of the web client application. Each of the following sections describe in detail the main elements of this configuration file.

POPUP_PAGES

See Section 8.21, *Pop-up Pages*.

MULTIPLE_POPUP_DOMAINS

See Section 8.21, *Pop-up Pages*.

ERROR_PAGE

If an error occurs at run-time, the user will be redirected to a page defined here. Depending on the error cause, two types of error page could be provided for reporting system or application failure (or a default page for reporting both kind of errors could be configured instead).

```
<ERROR_PAGE TYPE="SYSTEM" PAGE_ID="CuramSystemError" />
<ERROR_PAGE TYPE="APPLICATION" PAGE_ID="CuramError" />
```

Example 3.8 Error_Page Section Example

```
<ERROR_PAGE PAGE_ID="CuramError" />
```

Example 3.9 Error_Page Section Example with one default page

Please note: when overriding the `ERROR_PAGE` setting it is not possible for a custom configuration to define an `ERROR_PAGE` element without a `TYPE` attribute if a low priority component defines an `ERROR_PAGE` element with a `TYPE` attribute. In that case, the custom component needs to use a `TYPE` attribute and must override both supported types of error page to get

the desired effect

MULTIPLE_SELECT

Domains which should display as multiple select list boxes in forms are specified here. The `MULTIPLE` attribute, if true, allows multiple selection in the list.

```
<MULTIPLE_SELECT>
  <DOMAIN NAME="PRIMARY_ID" MULTIPLE="true" />
  <DOMAIN NAME="OTHER_ID" MULTIPLE="true" />
</MULTIPLE_SELECT>
```

Example 3.10 Multiple Select Section Example

FILE_DOWNLOAD_CONFIG

See Section 5.9.3.1, *File Downloads*.

ENABLE_COLLAPSIBLE_CLUSTERS

Set to `false` to disable collapsible clusters. By default this value is set to `true`.

```
<ENABLE_COLLAPSIBLE_CLUSTERS>false</ENABLE_COLLAPSIBLE_CLUSTERS>
```

Example 3.11 Disable Collapsible Clusters Example

APPEND_COLON

Set to `true` to automatically append colons to `FIELD` and `CONTAINER` labels within `CLUSTER` elements.

```
<APPEND_COLON>true</APPEND_COLON>
```

Example 3.12 Append Colon Section Example

ADDRESS_CONFIG

See Chapter 8, *Domain Specific Controls*.

ADMIN

The `ADMIN` element can contain any number of `CODETABLE_UPDATE`, `TAB_CONFIG_UPDATE` and `RESOURCE_UPDATE` elements. The `PAGE_ID` attribute of these elements specifies the page that will clear the relevant caches whenever its submit action is called.

```
<ADMIN>
  <CODETABLE_UPDATE PAGE_ID="CodeTableAdmin" />
</ADMIN>
  <TAB_CONFIG_UPDATE PAGE_ID="ApplicationConfigAdmin" />
```



```
<RESOURCE_UPDATE PAGE_ID="publishResourceChanges" />
```

Example 3.13 Admin Section Example

Please note: The caches are only cleared for the current instance of the web application. Other instances will have to be restarted to receive the code table updates. This feature applies at development time only.

STATIC_CONTENT_SERVER

This option specifies a base URL for all static content such as images, CSS files and JavaScript files.

```
<STATIC_CONTENT_SERVER>
  <URL>http://www.myserver.com/staticresources/</URL>
</STATIC_CONTENT_SERVER>
```

Example 3.14 Static Content Base URL Example

The forward slash at the end of the URL is optional. A full build is required to pick up this setting. This option allows the relocation of all static content to a separate server. If this option is used, the following folders and files need to be duplicated on the static content server:

- WebContent/*.*
- WebContent/CDEJ/**/*.*
- WebContent/genImages/**/*.*
- WebContent/Images/**/*.*

FIELD_ERROR_INDICATOR

This option indicates if field level error indicators are to be displayed when an error occurs. The error message is the alt text of the image and is available as a tool-tip when the mouse is hovered over the image. The feature only applies to text input and date-time fields. Also, this feature only applies to web-tier generated messages (data-type validation, mandatory fields etc.), it does not apply to messages generated from server side code since there is no way to associate a server exception with a client side field.

```
<FIELD_ERROR_INDICATOR>true</FIELD_ERROR_INDICATOR>
```

Example 3.15 Field Error Indicators Example

Please note if the FIELD_ERROR_INDICATOR element is not specified, it defaults to FALSE.

SECURITY_CHECK_ON_PAGE_LOAD

All server functions used on a Cúram screen are checked for authorization

when the page is initially loaded. If a user fails authorization for *any* of the server functions, an authorization error message will be displayed and the user will be prevented from viewing the page.

The `SECURITY_CHECK_ON_PAGE_LOAD` setting in `curam-config.xml` allows this functionality to be disabled and defers the authorization check to the server. For example, on an edit page that has both `DISPLAY` and `ACTION` server interfaces, the user must have authorization rights for the `DISPLAY` server interfaces at a minimum. If they do not have authorization rights for the `ACTION` server interfaces, the page will display, but they will get an authorization error message when they submit the page. To disable authorization on page load add the following to the `curam-config.xml`:

```
<SECURITY_CHECK_ON_PAGE_LOAD>false</SECURITY_CHECK_ON_PAGE_LOAD>
```

Example 3.16 Security Check on Page Load Example

Please note if the `SECURITY_CHECK_ON_PAGE_LOAD` element is not specified, it defaults to `TRUE`.

ENABLE_SELECT_ALL_CHECKBOX

The multi-select check-box `WIDGET` described here displays a column of check-boxes used to select items in a `LIST`. The following configuration setting causes a check-box to be displayed in the column header that can be used to select or de-select all of the check-boxes at once.

```
<ENABLE_SELECT_ALL_CHECKBOX>true</ENABLE_SELECT_ALL_CHECKBOX>
```

Example 3.17 Enable Select All Check-box Example

Please note if the `ENABLE_SELECT_ALL_CHECKBOX` element is not specified, it defaults to `FALSE`.

TRANSFER_LISTS_MODE

When set to `true` all multiple selection controls in an application are displayed as Transfer List widgets.

```
<TRANSFER_LISTS_MODE>true</TRANSFER_LISTS_MODE>
```

Example 3.18 Transfer Lists Mode Example

Please note if the `TRANSFER_LISTS_MODE` element is not specified, it defaults to `FALSE`.

HIDE_CONDITIONAL_LINKS

When set to `true` all conditional links that evaluate to `false` are not displayed. When set to `false` all conditional links that evaluate to `false` are displayed as disabled links.

```
<HIDE_CONDITIONAL_LINKS>true</HIDE_CONDITIONAL_LINKS>
```

Example 3.19 Hide Conditional Links

Please note if the `HIDE_CONDITIONAL_LINKS` element is not specified, it defaults to `TRUE`.

DISABLE_AUTO_COMPLETE

When set to `true` auto complete on all input fields is disabled. When set to `false` auto complete on all input fields is enabled.

```
<DISABLE_AUTO_COMPLETE>true</DISABLE_AUTO_COMPLETE>
```

Example 3.20 Disable Auto Complete

Please note if the `DISABLE_AUTO_COMPLETE` element is not specified, it defaults to `FALSE`.

SCROLLBAR_CONFIG

The `SCROLLBAR_CONFIG` element allows a vertical scrollbar to appear on a `LIST` or `CLUSTER` element after a maximum height is reached. It can contain two or less `ENABLE_SCROLLBARS` elements. The `ENABLE_SCROLLBARS` element has the following attributes:

- `TYPE` : Specifies the element in which vertical scrollbars are to be enabled. Can only be set to `LIST` or `CLUSTER`.
- `MAX_HEIGHT` : Specifies the maximum height a `CLUSTER` or `LIST` can reach before a vertical scrollbar is displayed.

```
<SCROLLBAR_CONFIG>
  <ENABLE_SCROLLBARS TYPE="LIST" MAX_HEIGHT="150" />
  <ENABLE_SCROLLBARS TYPE="CLUSTER" MAX_HEIGHT="100" />
</SCROLLBAR_CONFIG>
```

Example 3.21 Scrollbar Configuration

Please note if the `SCROLLBAR_CONFIG` element is not specified no `LIST` or `CLUSTER` element will display a vertical scrollbar.

PAGINATION

This element configures the `LIST` pagination options for the whole application. Individual lists can override the global settings.

```
<PAGINATION ENABLED="true">
  <DEFAULT_PAGE_SIZE>15</DEFAULT_PAGE_SIZE>
  <PAGINATION_THRESHOLD>15</PAGINATION_THRESHOLD>
</PAGINATION>
```

Example 3.22 Sample Pagination Configuration

Option Name	Required	Default	Description
ENABLED	No	true	Enables the ability to page through lists displayed in Cúram pages. Any LIST longer than the configured minimum size will display only the first "page" of data and the pagination controls will be displayed below the list.
DE-FAULT_PAGE_SIZE	No	15	Specifies the page size the list will get by default. The page size can be then changed at runtime by the user.
PAGINATION_THRESHOLD	No	Based on the DE-FAULT_PAGE_SIZE value.	Specifies the minimum list size at which pagination will be enabled. For shorter lists there will be no pagination, even if otherwise pagination is switched on.

Table 3.2 Pagination configuration options

Customizing Configuration Settings

The core component contains a copy of the `curam-config.xml` file, but you are free to augment and override the settings by including your own `curam-config.xml` file in your custom component. All of the individual `curam-config.xml` files will be merged into one at generation. The effect of this merging depends on each particular setting.

Some entries are global settings for the application and so must only appear once in the final output. These entries are as follows:

- HELP
- ERROR_PAGE
- APPEND_COLON
- ADMIN
- POPUP_PAGES/CLEAR_TEXT_IMAGE
- MULTIPLE_POPUP_DOMAINS/CLEAR_TEXT_IMAGE
- STATIC_CONTENT_SERVER

If you define one of these in a custom component, it will completely override that of the core component.

The other entries will be merged. This applies to the following elements:

- MULTIPLE_POPUP_DOMAINS
- POPUP_PAGES
- MULTIPLE_SELECT
- FILE_DOWNLOAD_CONFIG
- PAGINATION
- ADDRESS_CONFIG

Note, however, that particular address formats can be overridden. So, for example, if the core component had the following address format definition:

```
<ADDRESS_FORMAT NAME="US" COUNTRY_CODE="US">
  <ADDRESS_ELEMENT NAME="ADD1"
    LABEL="Core.Label.Address.1"
    MANDATORY="true"/>
  <ADDRESS_ELEMENT NAME="ADD2"
    LABEL="Core.Label.Address.2" />
  <ADDRESS_ELEMENT NAME="CITY"
    LABEL="Core.Label.City" />
  <ADDRESS_ELEMENT NAME="STATE"
    LABEL="Core.Label.State"
    CODETABLE="AddressState"
    MANDATORY="true"/>
  <ADDRESS_ELEMENT NAME="ZIP"
    LABEL="Core.Label.Zip" />
</ADDRESS_FORMAT>
```

Example 3.23 Extract from **curam-config.xml** File (1)

and if your custom component had the following address format definition:

```
<ADDRESS_FORMAT NAME="US" COUNTRY_CODE="US">
  <ADDRESS_ELEMENT NAME="ADD1"
    LABEL="Custom.Label.Address.1"
    MANDATORY="true"/>
  <ADDRESS_ELEMENT NAME="ADD2"
    LABEL="Custom.Label.Address.2" />
  <ADDRESS_ELEMENT NAME="CITY"
    LABEL="Custom.Label.City" />
  <ADDRESS_ELEMENT NAME="STATE"
    LABEL="Custom.Label.State"
    CODETABLE="AddressState"
    MANDATORY="true"/>
  <ADDRESS_ELEMENT NAME="ZIP"
    LABEL="Custom.Label.Zip" />
</ADDRESS_FORMAT>
```

Example 3.24 Extract from **curam-config.xml** File (2)

then it is the second one (i.e., the custom definition) that will appear in the final merged **curam-config.xml** file. This is because both address formats have the same name (“US”).

Dividing the Configuration File

The **curam-config.xml** file can be divided into manageable chunks. If you like, you can take one part of the configuration and save it in a file with a different name. Taking the previous address format configuration as an ex-

ample, you can create a file with the following contents:

```
<APP_CONFIG>
  <ADDRESS_CONFIG>
    <LOCALE_MAPPING LOCALE="en_US"
      ADDRESS_FORMAT_NAME="US">
      <ADDRESS_FORMAT NAME="US" COUNTRY_CODE="US">
        <ADDRESS_ELEMENT NAME="ADD1"
          LABEL="Custom.Label.Address.1"
          MANDATORY="true" />
        <ADDRESS_ELEMENT NAME="ADD2"
          LABEL="Custom.Label.Address.2" />
        <ADDRESS_ELEMENT NAME="CITY"
          LABEL="Custom.Label.City" />
        <ADDRESS_ELEMENT NAME="STATE"
          LABEL="Custom.Label.State"
          CODETABLE="AddressState"
          MANDATORY="true" />
        <ADDRESS_ELEMENT NAME="ZIP"
          LABEL="Custom.Label.Zip" />
      </ADDRESS_FORMAT>
    </ADDRESS_CONFIG>
  </APP_CONFIG>
```

Example 3.25 Sample `address-config.xml` File

You would then save this with a file name that ends with `-config.xml` anywhere within your component, say, `address-config.xml`. Note that the file must have the same `APP_CONFIG` root element as the full `curam-config.xml` file. As long as you follow these conventions, all of your configuration files will be merged into a single `address-config.xml` file at build time.



Configuration File Names

Two naming patterns are used for most configuration files. Some use the pattern `XConfig.xml` and others `X-config.xml`, where “X” is some prefix. For example, `ImageMapConfig.xml` and `address-config.xml`. The former pattern indicates a standalone configuration file that is not related to other configuration files. The latter pattern indicates that the file is really just part of the `curam-config.xml` file.

3.12.14 Custom Resources

Arbitrary files can be included in the web application by doing the following:

1. At the root of a component, created a folder called `WebContent`, for example `<client-dir>/components/MyComponent/WebContent`.
2. Place files in this folder using any folder structure you wish.
3. When you run the **client** build target these files will be copied directly to the `<client-dir>/WebContent` which represents the root of the web application. The folder structure will be maintained during the copy.

Files included in the application in this way take precedence over the merging and overriding process as described in previous sections for other resources. For example, if you include a CSS file in this way, the contents of the file will not be included in the CSS overriding process described in Section 3.12.11, *Cascading Stylesheets*. The copying of custom resources occurs after other source artifacts are built and merged, so it is possible to replace existing resources. Care should be taken in this case. For example, it would be possible to have a component with a file in `WebContent/WEB-INF/struts-config.xml` that would completely replace the Struts configuration file generated by the client build and therefore break the application. Finally, when multiple components have a `WebContent` folder they are copied based on component priority, but the copy is timestamp based. The copy command always uses verbose output for these files so the developer can see exactly what files are being copied.

Notes

¹Due to classloader issues with Log4j, the web-tier does not currently provide a configurable logging system in the same way as the server-tier.

Chapter 4

Localization

4.1 Objective

This chapter will introduce you to the various files that need to be updated when translating a Cúram application to a new language.

4.2 Prerequisites

You should be familiar with the basic concepts of Cúram CDEJ development (see Chapter 2, *Concepts*).

4.3 Introduction

Cúram is designed to support an application running simultaneously in as many languages as required. To simplify the translation process, the language-specific parts of the application are separated out from the application code.

4.4 Numbers

Numbers are language-specific and so a Cúram application treats numbers in a locale-specific manner depending on the preferred language of the user. For example, a decimal number can be represented as 7,99 or 7.99 depending on whether the user's locale is French or English.

4.5 File Encoding

OOTB Cúram supports the development of applications localized into many languages. The Cúram CDEJ generators support files encoded in the various character encodings appropriate for those languages. There are a number of

ways to define the encoding for a file, this is dependent on the type of file. The following sections describe how the encoding is set for the different types of files.

4.5.1 XML Files

The encoding for XML-format files is declared explicitly within the XML file itself, where the first line, the XML declaration, may look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

This tells the XML parser that the file uses the ISO-8859-1 encoding, a typical encoding for Western European languages. If the XML declaration is omitted, the parser will assume UTF-8 encoding, which covers most modern languages and many others, besides being based on the Unicode standard. It is very important that the XML declaration matches the actual file encoding. The declaration does not determine the encoding, it only identifies it; changing the declaration does not automatically change the file encoding. If you use a specialized XML editor application, then it will probably recognize the declaration and change the file encoding for you. Most plain-text editors will not do this, so you must ensure that you select the correct encoding in your editor before saving the file.

4.5.2 Java properties files

For Java properties files (used in the application, for example, to define the text strings that appear on client screens), there is no equivalent of the explicit XML declaration. The client generator must assume an encoding for the client properties files. The assumption the generator makes is that Java properties files are encoded in the default system encoding of the machine that the build is running on. This is a reasonable assumption given that the files themselves were likely created on the same machine or a machine of similar type in the same country. On a *Microsoft® Windows®* machine in Western Europe, for example, the system encoding is probably Cp1252, the Windows variant of ISO-8859-1. This encoding will handle the accented characters of Western European languages but does not cover, say, Cyrillic or Chinese characters.

If, for some reason, you are building on a machine that does not share its system encoding with the files that are being processed, you must indicate this by setting the `ENCODING` environment variable. For example, to build a Chinese language web client application on an English language Microsoft Windows machine, you might choose to save your properties files in the UTF-8 encoding, so you would set the `ENCODING` environment variable to UTF-8. During the build, you can see that the generator overrides its normal default setting:

```
System encoding is Cp1252.  
Using encoding UTF-8 to read properties files.
```

The Java Runtime Environment will always assume that properties files use the ISO-8859-1 encoding. This is not very helpful if you want to create properties files using the UTF-8 encoding for localization to, say, Chinese. To overcome this limitation, the Cúram CDEJ will automatically translate properties files from your preferred encoding (either the system default encoding, or the encoding specified via the `ENCODING` environment variable) into the encoding required by Java. This is performed automatically during the build process and your original properties files will not be affected.



Troubleshooting

Where a properties file has been saved in UTF-8 encoding, and this does not match the system encoding, build failures can occur. The build failure will report a `PageGenerationException`, where the build could not find a property even though the property exists in the relevant file. This happens where the properties file has been saved by a UTF-8 editor which adds the Byte Order Mark (BOM) at the beginning of the file. The property reported in the error will be the first property in the file. To resolve the issue the file should be saved in the correct encoding, ensuring the BOM character has been removed.

4.5.3 Non-XML Files

The non-XML files in the Cúram Reference Application are encoded in the ASCII encoding. ASCII has the useful property of being a subset of most other common file encodings. This means you do not generally need to convert the English language files that ship with the OOTB Cúram application in a new encoding in order to build them in a different language environment.

4.6 Locales

A Java locale identifier has three parts:

Language

A lower-case, two-letter, ISO-639 code.

See <http://www.unicode.org/onlinedat/languages.html>.

Country

An upper-case, two-letter, ISO-3166 code.

See <http://www.unicode.org/onlinedat/countries.html>.

Variant

A vendor-specific or browser-specific code.

The language code is required, but the other parts are optional. The individual parts are separated by an underscore character. Some examples of valid locales are: “en” (English language), “en_US” (English language for

the United States), zh_HK (Chinese language for Hong Kong). This system is used within the Cúram application to identify locales. Most locale-specific information in the application are contained in properties files.

4.6.1 Non JavaScript property files

When localizing an application (see Section 4.6.2, *JavaScript property files* for details on localizing JavaScript), you will need to create new properties files for each locale. The files for the default locale are named simply as `SomeFile.properties`. The files for other locales are identified by appending the locale identifier to the end of the file name after a separating “_” (underscore) character (i.e., between the name of the page and the `.properties` extension). For example, `SomeFile_es.properties` would be the name of the Spanish language version of `SomeFile.properties`.

It is useful to note that if a particular property is not found by the application in `SomeFile_es.properties`, the properties file for the default locale, i.e. `SomeFile.properties`, will be searched. This is particularly handy in the case of `Image.properties`, described below, where only some of your images contain text and thus need to be localized. Properties for the other images can be defined once in the default locale properties file and they will be picked up in all locales.

Once done adding localized `.properties` files, update the `LOCALE_LIST` environment variable as appropriate (this variable defines the list of locales the client will be built for), for example, set it to “en,es” for a default English language application and a Spanish language application. See Section 3.9, *Application Locales* for more details on this setting.

The merging of localized properties files from different components happens in exactly the same way as it does for default locale properties files. See Section 3.12.4, *Externalized Strings* for more details on the merging of properties files.

4.6.2 JavaScript property files

When localizing JavaScript files in the application, you will need to create new JavaScript property files for each locale. The files for the default locale are named simply as `*.js.properties`. The files for other locales are identified by appending the locale identifier - after a separating “_” (underscore) character - between the `.js` extension and the `.properties` extension. For example, `SomeJSFile.js_es.properties` would be the name of the Spanish language version of `SomeJSFile.js.properties` file. This file will be automatically processed by a client build. Similar to the non JavaScript property files, if a particular property is not found by the application in `SomeJSFile.js_es.properties` file, then the property from the default properties file (`SomeJSFile.js.properties`) will be used.

4.7 UIM Externalized Strings

As described in Section 3.12.4, *Externalized Strings*, all string values in UIM files are externalized to `.properties` files.

If `MyPage.uim` is the UIM file, then `MyPage.properties` is the corresponding properties file. To add localized properties files, please see Section 4.6, *Locales*.

The strings are stored in a properties file in the same folder as the page or view file. This file must have the same name as the page or view file but with the extension `.properties`. For example, if the page is stored in a file called `MyPage.uim`, the strings will be stored in the file `MyPage.properties` in the same folder. Similarly, views will see the `.vim` extension changed to `.properties`.

While UIM documents in the highest priority component override those in all other components, properties files in different components are merged together. Individual properties override those with the same property name defined in lower priority components. Also, when a UIM page includes a UIM view (a `.vim` file), all of the properties defined for both the page and the view are merged and the properties for the page override those defined for the view where they share the same property name. These two merging steps happen separately with the component order applied first for each properties file and the page-view order applied on the resulting properties. A property defined for a page will override a property of the same name defined for a view, even if the property for the view was defined in a higher priority component.

4.8 JavaScript Externalized Strings

As described in Section 3.12.4, *Externalized Strings*, all string values in JavaScript files should be externalized to JavaScript property files (`.js.properties` files).

By convention the name of the resource file for your JavaScript must be derived from name of the `.js` file itself. For example if your JavaScript file is called `SomeJSFile.js` then related localizable resources should be placed in `SomeJSFile.js.properties` file. A `*.js.properties` file can be placed anywhere in the component directory, but by convention it should be in the same directory as the related `*.js` file.

The exception to this is that a `*.js` file within a `WebContent` directory cannot have its associated `*.js.properties` file within the same directory. The associated `*.js.properties` file must be placed within a directory outside of the `WebContent` directory. To add localized JavaScript properties files, please see Section 4.6, *Locales*.

JavaScript Properties files with the same name across all components will be merged together during processing. Any property with the same name

will be overwritten by the highest component in the component order.

The use of placeholders within a property value is supported. The placeholders must be in the format `%ns` or `'%ns'` where `n` represents an integer from `1..n`, and `n` must be within a defined range. The range is defined by the number of of placeholders used within a property value. For example, if there are three placeholders within a property value then the placeholders must be numbered from 1 to 3 (e.g. `%1s`, `%2s`, `%3s`) and anything outside of this range is not supported.

4.8.1 Accessing properties in JavaScript

There are three requirements for accessing a JavaScript property.

```
// 1.
dojo.requireLocalization("curam.application", "SomeJSFile");

// 2.
dojo.require("curam.util.ResourceBundle");
var bundle = new curam.util.ResourceBundle("SomeJSFile");

// 3.
var localizedMessage = bundle.getProperty("myPropertyKey");
var localizedMessageWithSubstitutions
    = bundle.getProperty("my.sub.key", ["a", "b"]);
```

`curam.application` is the default package into which all localizable resources are placed by the Curam infrastructure. `SomeJSFile` is derived from the name of the related JavaScript properties file.

Example 4.1 Accessing a property

1. **Load the resources using `dojo.requireLocalization()`.** Refer to comment 1 in Example 4.1, *Accessing a property* for an example of this.
2. **Create an instance of the `curam.util.ResourceBundle` object.** This is required in order to be able to access the localized resources. Refer to comment 2 in Example 4.1, *Accessing a property* for an example of this.
3. **Access a property.** The `getProperty()` method can be used to access a property on the instantiated `ResourceBundle`. Refer to comment 3 in Example 4.1, *Accessing a property* for an example of how to get a property and a substituted (2 substitutions) property respectively.

4.9 Image.properties

The `Image.properties` file (see Section 3.12.5, *Images*) can be localized as per other properties files, please see Section 4.6, *Locales* for more information on localizing properties files. Once the localized properties file is created, place this beside the `Image.properties` file.

It is useful to note that if the application does not find a particular property in a localized properties file, it will check the default locale properties file. This is generally true for all properties files but it is particularly useful in the case of `Image.properties`. You might find that some of your images can be used no matter what language is displayed, whereas other images contain text and thus must be altered. It is only these latter images that need to be mentioned in the localized properties file.

4.10 Infrastructure Widget Properties Files

The following is a list of `.properties` files associated with Infrastructure widgets, e.g. the `AgendaPlayer.properties` file is associated with the `AgendaConfig.xml` file, which defines the Agenda Player widget.

- `AgendaPlayer.properties`
- `BarChart.properties`
- `Calendar.properties`
- `ComparedEvidence.properties`
- `DateTimeSelector.properties`
- `DecisionMatrixAddMessage.properties`
- `DisplayEvidence.properties`
- `EvidenceComparison.properties`
- `EvidenceReview.properties`
- `EvidenceTabContainer.properties`
- `FrequencyPatternSelector.properties`
- `GanttChart.properties`
- `IEGPlayer.properties`
- `Logon.properties`
- `MeetingView.properties`
- `PaymentStatement.properties`
- `RatesTable.properties`
- `Rules.properties`
- `TypicalPictureEditor.properties`
- `Workflow.properties`
- `WordFileEdit.properties`



Note

The names of the properties files associated with infrastructure widgets are reserved names and must not be used for the name of any other client properties file. No warning is printed to the console in this scenario, therefore care must be taken when naming other properties files.

To customize a widget properties file, create a new version under the `web-client/components/custom` component folder, where the default content for the file can be found in the corresponding sample widget properties file located in the `<cdej-dir>/doc/defaultproperties/` folder. For each entry in Cúram's version of the file you wish to change, add a corresponding entry to your custom file. These properties files can be localized as per Section 4.6, *Locales*.

4.10.1 Frequency Pattern Selector Localization

The Frequency Pattern Selector infrastructure widget is used to construct frequency patterns such as:

```
the first day of every 1 month(s)
```

This sentence is made up of fixed text from its associated `FrequencyPatternSelector.properties` file as well as values selected by a user from an input field and two drop-downs in the widget, refer to this example frequency pattern in Figure 8.1, *Frequency Pattern Selector Pop-up*.

Because of the grammar differences between different languages, the construction of this example frequency pattern sentence can be dramatically changed in other languages, like the values selected by a user can be re-ordered in it. Therefore, the placeholders are introduced to represent these user selected values so that we can localize every frequency pattern as "whole" into every single property in the properties file.

Here is the property entry from the `FrequencyPatternSelector.properties` for this example frequency pattern:

```
Text.monthly.freq.type.two= The %ordinal% %dayOfWeekExtended%  
of every %monthInterval% month(s)
```

The strings `%ordinal%`, `%dayOfWeekExtended%` and `%monthInterval%` in this property entry are the placeholders that map to the values that will be selected from two drop-downs and one input field in the widget. The detailed explanation of these three placeholders will be covered later in a table.

In order to use these placeholders properly, you need to stick to the following two rules:

- **The placeholders control the layout of the widget.** Any change of the location of a placeholder in a localized text for a certain frequency pat-

tern would cause the change of the layout of this frequency pattern to be displayed on the Frequency Pattern Selector widget.

- **The placeholders that can be used for every frequency pattern are fixed .** You could not change, add or reduce placeholders used for a certain frequency pattern. It will cause this widget failing to work.

A description of all these placeholders used in the properties file of this widget is listed as follows:

Placeholder Name	Description
<code>%dayInterval%</code>	A day interval. It maps to an input field where you can enter a number for a day interval for a frequency pattern.
<code>%weekInterval%</code>	A week interval. It maps to an input field where you can enter a number for a week interval for a frequency pattern.
<code>%dayOfWeek%</code>	A set of days in a week. It maps to a collection of check boxes where you can multi select the days in a week for a frequency pattern.
<code>%dayOfWeekExtended%</code>	It is an extension of the values represented by <code>%dayOfWeek%</code> , which also includes the weekday, weekend day and day value. It maps to a drop-down where you can select one of those day values for a frequency pattern.
<code>%monthInterval%</code>	A month interval. It maps to an input field where you can enter a number for a month interval for a frequency pattern.
<code>%ordinal%</code>	an ordinal, e.g. first, second. It maps to a drop-down where you can select an ordinal for a frequency pattern.
<code>%dayIntervalOne%, %dayIntervalTwo%</code>	Two day intervals in a frequency pattern. They should be used together and map to two input field where you can enter a number for a day interval respectively for a frequency pattern.
<code>%ordinalOne%, %ordinalTwo%</code>	Two ordinals in a frequency pattern. They should be used together and map to two drop-downs where you can select an ordinal respectively for a frequency pattern.
<code>%monthOfYear%</code>	A month in a calendar year. It maps to a drop-down where you can select a month for a frequency pattern.

Table 4.1 Placeholders used in Frequency Pattern Selector

As stated in the second rule above, the placeholders used for every frequency pattern are fixed. So you need to take care that you have used them properly when localizing the properties in this widget properties file. As long as you keep this in mind, the customization of this widget properties file is also no difference from other infrastructure widgets. The following table lists all the properties and the placeholders they contain for every frequency pattern sentence displayed on the Frequency Pattern Selector.

Property Name	Placeholders it contains
<code>Text.daily.freq.type.one</code>	<code>%dayInterval%</code>
<code>Text.daily.freq.type.two</code>	None.
<code>Text.weekly.freq.type</code>	<code>%weekInterval%, %dayOfWeek%</code>
<code>Text.monthly.freq.type.one</code>	<code>%dayInterval%, %monthInterval%</code>
<code>Text.monthly.freq.type.two</code>	<code>%ordinal%, %dayOfWeekExtended%, %monthInterval%</code>
<code>Text.bimonthly.freq.type.one</code>	<code>%dayIntervalOne%, %dayIntervalTwo%</code>
<code>Text.bimonthly.freq.type.two</code>	<code>%ordinalOne%, %ordinalTwo%, %dayOfWeek%</code>
<code>Text.yearly.freq.type.one</code>	<code>%monthOfYear%, %dayInterval%</code>
<code>Text.yearly.freq.type.two</code>	<code>%ordinal%, %dayOfWeekExtended%, %monthOfYear%</code>

Table 4.2 Properties used for the Frequency Pattern Selector

4.11 CDEJResources.properties

This properties file can be localized as per Section 4.6, *Locales*. Images defined in this file can also be customized per locale.

4.12 ApplicationConfiguration.properties

This properties file does not, in itself, need to be localized but there are a couple of settings within this file which are related to the localization of date and address formatting. See Section 3.11.2, *Configuring the Application* for details.

4.13 Application-wide Menu

The contents of the application-wide menu (that normally appears in the top-right of the screen) are defined in `curam-config.xml`. It is possible to put the text that will appear on screen directly into this file, in the `LABEL` attribute of the `LINK` element. That approach, however, is not suitable if the application should be viewable in multiple languages, so the application will first check if the `LABEL` attribute is actually a key into the `CDEJResources.properties` file. If it finds the key, it will use the corresponding value in the menu. To localize the menu, therefore, simply include the same key in the localized version of `CDEJResources.properties`. This properties file can be localized as per Section 4.6, *Locales*.

4.14 Tabbed Configuration Artifacts

Each tabbed configuration artifact will have a corresponding properties file for any text that may be localizable. To localize this text for a specific language, you must add the locale-specific properties file beside its associated tabbed configuration artifact in your `<custom>` component. These properties file can be localized as per Section 4.6, *Locales*.

4.15 Runtime Messages

The Cúram CDEJ runtime messages can be localized or customized by creating a `RuntimeMessages.properties` file within the `curam/omega3/i18n` folder below the web application project's `JavaSource` folder, i.e. the `<client-dir>/JavaSource` folder. The default content for this file can be found in the `<cdej-dir>/doc/defaultproperties/` folder. Any messages present in this file will override the corresponding messages from the `RuntimeMessages.properties` shipped with the Cúram CDEJ. The standard file naming convention for Java properties files can be used to add locale-specific messages. For example, to create a Spanish version, a file `RuntimeMessages_es.properties` would be created.

It is not necessary to copy all of the messages into the custom message catalog when customizing only some of them. Only the messages that are customized need to be defined in the custom message catalog; the other messages will be loaded from the default message catalog.

When resolving error messages, the custom message catalog is checked first and all the locale fall-backs are applied. If a message is not found, then the default message catalog (from the Cúram CDEJ) is checked. Therefore, a message in a custom message catalog will take precedence over one in a default catalog even if the locale of the default catalog is more specific.

When customizing a message, the message argument placeholders cannot be changed. The message argument placeholders have the form `%ns` where `n` is the argument number. The message arguments can be moved around and their order changed, but no new arguments may be added and none may be

removed. The meaning of each message argument for every error message is provided in the Cúram Web Client Error Message Guide.

Chapter 5

UIM Reference

5.1 Objective

This chapter provides you with all the information about UIM required to develop Cúram web application pages.

5.2 Prerequisites

You should be familiar with the basic concepts of Cúram CDEJ development (see Chapter 2, *Concepts*) and web application development. You should also have some knowledge of the basic format of XML documents.

5.3 Introduction

UIM is the Cúram User Interface Meta-data format used to specify the contents of the Cúram web application client pages. UIM is an XML dialect and all UIM files are well-formed XML. The Cúram CDEJ will translate UIM files into JSP files that can be deployed to your web application server.

5.4 Creating UIM Documents

You can use any text editor to write UIM documents, but it is usually easier if a specialized XML editor is used. The CDEJ includes an XML Schema file defining the syntax of a UIM document and when this is combined with a schema-aware XML editor, you will have access to many time-saving facilities such as auto-completion, syntax checking, etc.

5.5 UIM Document Types

When creating UIM documents, there are four root elements that are valid:

PAGE, VIEW, PAGE_GROUP and APPLICATIONS. These root elements are used to create the two types of UIM document:

PAGE

This defines a UIM page that will be translated into a JSP page. The file name must be the same as the value of the PAGE_ID attribute of the root element. The file extension to use is `.uim`. UIM pages can be organized arbitrarily into sub-folders within a component folder for convenience in managing a large number of files. Ultimately, all UIM pages are generated into JSP pages in a single folder, so the PAGE_ID attribute of the PAGE element and consequently the file names of all the `.uim` files must be unique within a component.

VIEW

This defines a portion of a page that can be included into a PAGE element in another UIM document. This allows common sequences of elements to be reused. The file name is not restricted. The file extension to use is `.vim`. Like UIM pages, views can be organized into an arbitrary folder structure within a component folder, but the file names must be unique within that component.

5.6 UIM Pages

Chapter 2, *Concepts* covered the basic concepts behind UIM pages and what clusters, lists, action sets, action controls, containers, and fields are, so this information will not be repeated here.

The elements in a page must follow a strict order imposed by the XML Schema definition of UIM. However, this order is only imposed when editing using a schema-aware XML editor. The JSP generator does not check the ordering at present. The order in which elements are presented in the child element tables in this reference is the order in which the elements should be used in the UIM documents unless otherwise indicated. There is no specific ordering for attribute values.

5.7 UIM Views

A PAGE element can contain an INCLUDE element anywhere at the top level that allows commonly used fragments of UIM to be inserted at that point during translation. The included elements are defined in a UIM document called a *view*. The view document uses VIEW as the root element. Elements included from a view must be valid in the context in which they have been included. For example, a PAGE element that already contains a PAGE_TITLE element, cannot include a view that also defines a PAGE_TITLE element. Similarly, the schema rules governing the order of elements in a page must be observed when elements are included from a view.

Views are similar to pages in what they can contain, the only differences are

as follows:

- A view cannot contain an `INCLUDE` element to include another view.
- A view does not have any `PAGE_ID` attribute, this is defined in the page that includes the view.

All other elements that are valid in a `PAGE` element at the top level, are also valid in a `VIEW`.

When including views, the name of the view file must be specified. Regardless of where in the component the file including the view is, only the name of the view file is required, not its path.

5.8 Externalized Strings

All string values and image references in UIM documents must be externalized, i.e., the actual values are stored in files separated from the UIM. This aids maintenance and allows the application to be localized.

See Section 3.12.4, *Externalized Strings* for details on externalizing strings.

5.9 UIM Reference for Pages and Views

5.9.1 Introduction

This section describes the `PAGE` and `VIEW` elements and all of the child elements that they can contain with the exception of `WIDGET` elements. These are treated in the next section.

Most elements have a list of attributes that can be used in any order. Some attributes are optional and have default values when omitted. Others can have one of a range of values. Boolean attributes can only have the values `true` and `false` (case-sensitive).

Many elements can have child elements and these are listed in the order in which they must be added and include details on their cardinality. Cardinalities use “0” to indicate that the element is optional, “1” to indicate that it can appear only once, and “n” to indicate that it can be appear any number of times. The “..” indicates the range of the cardinality. For example, “0..1” indicates that the element can appear zero or one times in this location, i.e., it is optional, while “1..n” indicates that an element must appear at least once, but can appear any number of times thereafter.

5.9.2 Connection Types

UIM pages use connections for associating components on a page with actual data. The connection type is reflected in the connection tag name and is roughly equivalent to data direction. The three types of connection available are `SOURCE`, `TARGET` and `INITIAL` (see Section 5.9.30, *SOURCE*, Sec-

tion 5.9.32, *TARGET*, and Section 5.9.15, *INITIAL*, respectively).

Connection endpoints are further distinguished by the setting of the *NAME* attribute. The value of this attribute may be the name of the server interface used, *TEXT*, *CONSTANT* or *PAGE*. These values designate objects which supply or consume data. *TEXT* or *CONSTANT* can only be used when *TARGET* has a server interface defined in the *ACTION* phase.

```
<PAGE PAGE_ID="APage" >
  <PAGE_TITLE>
    <CONNECT>
      <SOURCE NAME="TEXT" PROPERTY="Page.Title.Static"/>
    </CONNECT>
  </PAGE_TITLE>

  <SERVER_INTERFACE NAME="DISPLAY_SI"
    CLASS="sourceClass"
    OPERATION="read"
    PHASE="DISPLAY"/>
  <SERVER_INTERFACE NAME="ACTION_SI"
    CLASS="targetClass"
    OPERATION="modify"
    PHASE="ACTION"/>

  <PAGE_PARAMETER NAME="P_PARAM" />

  <CONNECT>
    <SOURCE NAME="CONSTANT"
      PROPERTY="From.Constants.Props" />
    <TARGET NAME="ACTION_SI"
      PROPERTY="aProperty" />
  </CONNECT>

  <ACTION_SET BOTTOM="true" TOP="false">
    <ACTION_CONTROL TYPE="SUBMIT" LABEL="Button.Submit">
      <LINK PAGE_ID="APage">
        <CONNECT>
          <SOURCE NAME="DISPLAY_SI" PROPERTY="PARAM" />
          <TARGET NAME="PAGE" PROPERTY="P_PARAM" />
        </CONNECT>
      </LINK>
    </ACTION_CONTROL>
  </ACTION_SET>

  <CLUSTER NUM_COLS="1" SHOW_LABELS="false">
    <FIELD LABEL="Label.Text">
      <CONNECT>
        <SOURCE NAME="DISPLAY_SI" PROPERTY="sourceField"/>
      </CONNECT>
      <CONNECT>
        <TARGET NAME="ACTION_SI" PROPERTY="targetField"/>
      </CONNECT>
    </FIELD>
  </CLUSTER>
</PAGE>
```

Example 5.1 Connection Types Example

Most frequent is a connection to a server interface. Here, the *NAME* attribute corresponds to an existing (i.e. declared on the page) *SERVER_INTERFACE NAME* attribute value (*DISPLAY_SI* and *ACTION_SI* in the example above).

A value of *TEXT* means data is sourced from a properties file. The *PROPERTY* attribute in this case contains the name of an externalized string in a page-specific property file. In the example, the file *APage.uim* has a page

title which references the `Page.Title.Static` property in the associated `APage.properties` file.

A value of `CONSTANT` provides similar functionality to `TEXT` but the externalized string is component-specific rather than page-specific and is sourced from a file called `Constants.properties`. In the example, there is a page level connection to a `From.Constants.Props` property.

A connection might also source its data from a page parameter (i.e., a variable declared on a page, `P_PARAM` in the example). In this case `PAGE` is used as the value of the `NAME` attribute.

There are limitations and restrictions on the use of the various connection types in various contexts. The UIM element descriptions below detail these limitations where they arise.

5.9.3 ACTION_CONTROL

The `ACTION_CONTROL` element defines a link (text based), button or file download link that the user can activate on a page.

File Downloads

An `ACTION_CONTROL` with the `TYPE` set to `FILE_DOWNLOAD` results in the generation of a hyperlink on the page. Clicking on the hyperlink invokes a special `FileDownload` servlet included in the Cúram CDEJ that returns the contents of a file from the database. The `FileDownload` servlet is configured with the server interface to call to get the file contents and the parameters to pass to identify that file. The configuration is performed in the `curam-config.xml` file. A single server interface can be configured for each page of the application that includes file download action controls. An example configuration is shown in Example 5.2, *Example Configuration for File Download*, below:

A `WIDGET` with the `TYPE` set to `FILE_DOWNLOAD` can also be used to generate a hyperlink to download a file. You should use the `ACTION_CONTROL` element when the hyperlink text is the fixed `LABEL` value. The `FILE_DOWNLOAD WIDGET` allows the hyperlink text to be a dynamic value retrieved from a server interface property.

```
<APP_CONFIG>
  <FILE_DOWNLOAD_CONFIG>
    <FILE_DOWNLOAD PAGE_ID="FileDownload"
      CLASS="curam.interfaces.FilePkg.File_read_TH">
      <INPUT PAGE_PARAM="fileID" PROPERTY="key$fileID"/>
      <FILE_NAME PROPERTY="dtls$fileName"/>
      <FILE_DATA PROPERTY="dtls$fileData"/>
    </FILE_DOWNLOAD>
  </FILE_DOWNLOAD_CONFIG>
</APP_CONFIG>
```

Example 5.2 Example Configuration for File Download

Each configuration for downloading files is contained in a `FILE_DOWNLOAD` element within the `FILE_DOWNLOAD_CONFIG` ele-

ment in the configuration file. There should be one `FILE_DOWNLOAD` element for each page that contains file download action controls.

The `FILE_DOWNLOAD` element takes two attributes: `PAGE_ID` for the identifier of the page containing the action controls to which this configuration will be applied, and `CLASS` containing the name of the server interface that will be called by the `FileDownload` servlet when the generated hyperlink is invoked.

The `FILE_DOWNLOAD` element can contain zero or more `INPUT` elements specifying the key values to set before the server interface is called. These `INPUT` elements associate page parameters with properties of the server interface. The `PAGE_PARAM` attribute specifies the name of the page parameter whose value will be used as a key value, and the `PROPERTY` attribute specifies the key property of the server interface that must be set to identify the file. The page parameters are set by the `LINK` element within the `ACTION_CONTROL`, as you will see below.

The other three elements, `FILE_NAME` and `FILE_DATA`, and `CONTENT_TYPE` all have `PROPERTY` attributes that indicate the properties of the server interface that will contain; the name of the file, the contents of the file, and the content type of the file respectively, after the server interface is called. This data is returned to the client in response to the activation of the hyperlink and the user's browser will present them with the download dialog box prompting them to save or open the file.

Where property names are specified, the names must be written in full and cannot be abbreviated like they can in UIM documents.

Attributes

The `ACTION_CONTROL` element has the following attributes. The `LABEL` attribute must be present.

Attribute Name	Required	Default	Description
<code>LABEL</code>	Yes		A reference to an externalized string containing the label text for this action control. If the <code>TYPE</code> is <code>ACTION</code> , this will be the text of the hyperlink. If the <code>TYPE</code> is <code>SUBMIT</code> , this will be caption of the submit button.
<code>LA-BEL_ABBREVIATION</code>	No		A reference to an externalized string containing the label abbreviation text for this action control. This label abbreviation is placed only on table headers in a <code>LIST</code> .
<code>TYPE</code>	No	<code>ACTION</code>	The type of action control to create. There are six types: <code>ACTION</code>

Attribute Name	Required	Default	Description
			(the default) defines a link to another page, SUBMIT forwards the page's form data to the action phase for processing, DISMISS closes a pop-up page, SUBMIT_AND_DISMISS combines a submit with closing a pop-up page (see Section 8.21, <i>Pop-up Pages</i> for details on working with pop-up pages), FILE_DOWNLOAD defines a link that triggers the download of a file from the server, and CLIPBOARD places a predefined value to the system clipboard.
STYLE	No		The class name of the CSS style to use when formatting the action control. Supported by action controls in action sets only.
CONFIRM	No		Use the CONFIRM attribute of ACTION_CONTROL to force a confirmation dialog when the action control is activated. The value of the CONFIRM attribute is a reference to the confirmation message in the page properties file.
DEFAULT	No	false	If there is more than one submit action on a page, it is useful to specify which one is executed when the user hits the <i>Enter</i> key. This is especially recommended when the submitting action controls are contained within the different action sets as in this case the default action could be different than the first submit action declared on the page. The default action can be specified by setting this attribute to true. Note that only one submit action on a page can have a DEFAULT value of true.
ACTION_ID	No		A custom identifier for action controls of TYPE=SUBMIT. It is used

Attribute Name	Required	Default	Description
			<p>in conjunction with ACTION_ID_PROPERTY attribute of SERVER_INTERFACE element to inform the server side code which action control was used to make the server call.</p> <p>This attribute is only valid on action controls of TYPE= SUBMIT.</p> <p>The value of this attribute among the action controls within the page must be unique.</p> <p>The value of this attribute must be in the format suitable for the domain associated with the property specified in the ACTION_ID_PROPERTY attribute of SERVER_INTERFACE.</p> <p>This attribute must be either specified on all action controls within the page or not specified on any of them.</p> <p>If this attribute is specified then the ACTION_ID_PROPERTY attribute of SERVER_INTERFACE must also be specified.</p>
IMAGE	No		<p>The value of this attribute refers to an externalized string which maps to a specific icon or graphic in the application. An action control with this attribute can only be used within a CONTAINER element.</p>
ALIGNMENT	No	RIGHT	<p>When contained in a page level ACTION_SET of a Modal Dialog, the ALIGNMENT attribute is supported. This will define the individual horizontal alignment of the action control. It can be set to LEFT or RIGHT. The default is to right aligned.</p>

Table 5.1 Attributes of the ACTION_CONTROL Element

Child Elements

The ACTION_CONTROL element can contain the following child elements:

Element Name	Cardinality / Description
LINK	<p>0..1. An action control with a TYPE of ACTION that has no LINK element will create a link to the previous page in the history that had SAVE_LINK set to true on the link that led to this page (this is typically used for <i>Cancel</i> buttons). However this type of ACTION_CONTROL should not be present on a page that is directly referenced by any tabbed configuration artifact. Also, if this type of ACTION_CONTROL is preceded by another ACTION_CONTROL of the same type in the page history, there is the potential of a circular reference between these pages.</p> <p>An action control with a TYPE of SUBMIT that has no LINK element will submit the field values to the action phase and then return to the previous page in the history that had SAVE_LINK set to true on the link that led to this page.</p> <p>An action control with a TYPE of FILE_DOWNLOAD only requires a link if it must provide the page parameter values specified in the INPUT elements of its configuration. Each CONNECT element in the link can contain a SOURCE element to specify the value and a TARGET element specifying the page parameter to which to map the value. The PROPERTY attribute value of the page parameter must match the PAGE_PARAM attribute value of the INPUT element in the configuration.</p>
CONNECT	<p>0..1. A CONNECT element specifying a single SOURCE end-point. As a direct child it is used only for an action control with a TYPE of CLIPBOARD. Such an action control places predefined textual data into the system clipboard when clicked.</p> <p>Text to be copied to clipboard can be sourced from the server, the request or a properties file.</p> <p>The CONNECT element used can only contain a SOURCE element with a NAME property of PAGE, TEXT or the name of a server interface defined within the page.</p>

Element Name	Cardinality / Description
SCRIPT	0..n. A script element associated with an action control. For a detailed description of this element see Section 5.9.28, <i>SCRIPT</i> . SCRIPT elements are not supported on ACTION_CONTROL elements with a type of CLIPBOARD.
CONDITION	0..1. Affects whether or not the ACTION_CONTROL is displayed.

Table 5.2 Child Elements of the ACTION_CONTROL Element

When linking to another page, the link must specify all page parameters declared on the target page.

5.9.4 ACTION_SET

The ACTION_SET element groups a number of ACTION_CONTROL elements together. Depending on the context in which the action set is defined, the action controls will be displayed in differing ways.

At the page level, action controls are displayed at the left side of the page title bar, see the Page Level Action Control in User Interface Element 10 of Figure 2.1, *Application User Interface Overview*. If the action set contains two or less action controls, then each link is displayed side by side with a new item icon to the left of it. The SEPARATOR child element has no affect.

If three or more action controls exist at the page level, then a drop down menu will display each action control as a menu item. In this case, the SEPARATOR element inserts a gray separator into the drop down menu at the position indicated in the UIM file.

At the list level, all action controls will be displayed in a menu drop down. The SEPARATOR element inserts a gray separator into the drop down menu.

For action sets defined at the cluster or list level, the action controls can be displayed above and/or below the element with which the action set is associated and are aligned horizontally.

In all scenarios, conditional links that evaluate to false will not display if HIDE_CONDITIONAL_LINKS attribute is set to true, otherwise the conditional link displays but is disabled.

Attributes

The ACTION_SET element has the following attributes:

Attribute Name	Required	Default	Description
TOP	No	true	Defines whether the action controls will be displayed

Attribute Name	Required	Default	Description
			above the associated element. Can be set to <code>true</code> (the default) or <code>false</code> .
BOTTOM	No	<code>true</code>	Defines whether the action controls will be displayed below the associated element. Can be set to <code>true</code> (the default) or <code>false</code> .
ALIGNMENT	No	DEFAULT	Defines the horizontal alignment of the set of action controls with respect to the associated element. Can be set to LEFT, RIGHT, CENTER, or DEFAULT. The value DEFAULT corresponds to the CSS class <code>ac_default</code> in <code>curam_common.css</code> . The default is to be left aligned. In addition, for a page level ACTION_SET in a Modal Dialog, LEFT, RIGHT and DEFAULT values are supported.
TYPE	No	DEFAULT	Defines the location of the action set. This can be set to LIST_ROW_MENU or DEFAULT. LIST_ROW_MENU is applicable where the ACTION_SET is contained within a LIST. It indicates that the action set should be displayed as a list actions menu within each list row entry.

Table 5.3 Attributes of the ACTION_SET Element

**Note**

An ACTION_SET of type LIST_ROW_MENU should not be used to open a Pop-up search dialog.

Child Elements

The ACTION_SET element can contain the following child element:

Element Name	Cardinality / Description
ACTION_CONTROL	1..n. See the description of ACTION_SET's parent element to see what ACTION_CONTROL elements are valid in each context.
CONDITION	0..1. Affects whether or not the ACTION_SET is displayed.
SEPARATOR	0..n. allows the for ability to add a visual separator between action controls that display in the page action drop down menu.

Table 5.4 Child Elements of the ACTION_SET Element

5.9.5 CLUSTER

The CLUSTER element defines a group of input and/or output fields containing data from any data source (server interface property values, externalized string values, or page parameter values) and supplying data to other data targets (server interface properties, or page parameters). Clusters generally show the fields with labels to the left and these label/field pairs in a number of columns. Clusters can also include other clusters and lists in place of fields to allow more complex layouts.

Attributes

The CLUSTER element has the following attributes:

Attribute Name	Required	Default	Description
TITLE	No		A reference to an externalized string containing the title string for this cluster.
NUM_COLS	No	1	The number of columns to display in the cluster (a cluster column includes both the label and field).
TAB_ORDER	No	COLUMN	Indicates the order to layout elements in a multi-column cluster. The elements can be ordered by ROW or COLUMN (default). Please note, if a CLUSTER has NUM_COLS set to 2 or above and is going to contain a mix of LIST and FIELD elements, the TAB_ORDER must be set to ROW.
SHOW_LABELS	No	true	Can be set to true (the default) to show labels beside the field values or false to show no la-

Attribute Name	Required	Default	Description
			Labels at all.
LAYOUT_ORDER	No	LABEL	Labels can be displayed to the left or to the right of their associated fields. Set the attribute value to LABEL to show labels to the left (this is the default behavior). Set the attribute value to FIELD to show labels to the right.
WIDTH	No	100	The percentage of the width of the containing area that the cluster should occupy.
STYLE	No		The class name of the CSS style to associate with this cluster for formatting.
DESCRIPTION	No		A reference to an externalized string that provides more details about the cluster than the title alone. This will be displayed below the title on the page.
LABEL_WIDTH	No		<p>The percentage of the width of a cluster column that the label should occupy. By default, the web browser will determine the widths as appropriate.</p> <p>This attribute has an effect even if SHOW_LABELS is set to false. It is possible, say, to use action controls in place of text labels. You might want to control the width of these action control columns and you can do that by setting the LABEL_WIDTH attribute. The specified width will be applied to every other column. Whether this starts with the first or second column depends on the LAYOUT_ORDER attribute.</p> <p>The LABEL_WIDTH attribute will not apply to codetable hierarchy fields when SHOW_LABELS is set to false or the FIELD attribute CONFIG has a value of CT_DISPLAY_LABELS. See the CONFIG attribute in Sec-</p>

Attribute Name	Required	Default	Description
			tion 5.9.11, <i>FIELD</i> for more information on code table hierarchies.
BEHAVIOR	No	EXPANDED	Collapsible clusters can be initially displayed expanded or collapsed on a page. Set the attribute value to EXPANDED to display a collapsible cluster fully expanded. Set the attribute to COLLAPSED to display a collapsible cluster collapsed. To remove the collapsible functionality from a cluster set the attribute to NONE. Note that this attribute is only applicable when the property ENABLE_COLLAPSIBLE_CLUSTERS is not set or is set to true in curam_config.xml. For details see Section 3.12.13, <i>General Configuration</i> . This feature is currently not supported on clusters containing Charts, Evidence Review Widgets, Evidence Comparison Widgets, or Evidence Tab Containers.
SUMMARY	No		A reference to an externalized string containing the summary of this cluster. The SUMMARY attribute describes the purpose and/or structure of a cluster.
SCROLL_HEIGHT	No		Specifies in pixels the desired maximum height of a scrollable cluster.

Table 5.5 Attributes of the CLUSTER Element

Child Elements

The CLUSTER element must contain one of the following elements; ACTION_SET, FIELD, WIDGET, CONTAINER, CLUSTER or LIST.

Element Name	Cardinality / Description
CONDITION	0..1. Affects whether or not the cluster is displayed.
TITLE	0..1. The TITLE element will be displayed above the CLUSTER.

Element Name	Cardinality / Description
DESCRIPTION	0..1 The DESCRIPTION element has the same behavior as the DESCRIPTION attribute but allows the description to be built up from a number of sources. If both are specified, this element takes precedence over the corresponding attribute.
ACTION_SET	0..1. The action set can contain ACTION_CONTROL elements of any type. The action controls will be displayed above or below the entire cluster.
FIELD	0..n. The FIELD, CONTAINER, WIDGET, CLUSTER, and LIST elements can be freely intermingled.
WIDGET	0..n. The FIELD, CONTAINER, WIDGET, CLUSTER, and LIST elements can be freely intermingled.
CONTAINER	0..n. The FIELD, CONTAINER, WIDGET, CLUSTER, and LIST elements can be freely intermingled.
CLUSTER	0..n. The FIELD, CONTAINER, WIDGET, CLUSTER, and LIST elements can be freely intermingled.
LIST	0..n. The FIELD, CONTAINER, WIDGET, CLUSTER, and LIST elements can be freely intermingled.

Table 5.6 Child Elements of the CLUSTER Element

5.9.6 CONDITION

The CONDITION element represents the condition under which an ACTION_SET, ACTION_CONTROL, LIST, or a CLUSTER is displayed. If a condition evaluates to true, then the parent element will be displayed; if the condition evaluates to false, then the parent element is not displayed with the following exception: An ACTION_SET or ACTION_CONTROL element will display disabled links if the condition evaluates to false and the HIDE_CONDITIONAL_LINKS attribute on the PAGE element or in thecuram_config.xml file has been set to false. Conditional ACTION_SETS and ACTION_CONTROLS are mutually exclusive from one another and therefore the CONDITION element should be set for either one (depending on the requirements) but not both.

Finally, if the condition equates to false for those conditional action sets or action controls which appear as drop down menu items, then a single disabled menu item titled, 'No Contents' is displayed (upon selecting the drop down menu icon).

Attributes

The CONDITION element has no attributes.

Child Elements

The `CONDITION` element must contain either an `IS_TRUE` element or an `IS_FALSE` element. It must not be empty and it must not contain more than one element.

Element Name	Cardinality / Description
<code>IS_TRUE</code>	0..1 If the property referenced by the <code>IS_TRUE</code> element returns true then the condition is true.
<code>IS_FALSE</code>	0..1 If the property referenced by the <code>IS_FALSE</code> element returns false then the condition is true.

Table 5.7 Child Elements of the `CONDITION` Element.

For Agenda Player specific use, see Section 8.22, *Agenda Player*

5.9.7 CONNECT

The `CONNECT` element defines a data connection between two connection end points such as server interface bean properties, page parameters, screen controls, localized string values, etc.

Attributes

The `CONNECT` element has no attributes.

Child Elements

The `CONNECT` element must contain at least one of the child elements from the table below, but the details of how these elements are used depends on the context in which the `CONNECT` element is defined. See the specific parent or child element's description for more details.

Element Name	Cardinality / Description
<code>INITIAL</code>	0..1. This element is only valid in <code>CONNECT</code> elements contained within <code>FIELD</code> elements.
<code>SOURCE</code>	0..1. Within a <code>FIELD</code> element, the <code>SOURCE</code> is the source of the value displayed in the field control (unless <code>INITIAL</code> is used).
<code>TARGET</code>	0..1. Within a <code>FIELD</code> element, the <code>TARGET</code> is the property to which the value in the field control will be assigned.

Table 5.8 Child Elements of the `CONNECT` Element

5.9.8 CONTAINER

The CONTAINER element groups FIELD, ACTION_CONTROL and IMAGE elements so that they can be used in a single cell of a CLUSTER or LIST element.

Attributes

The CONTAINER element has the following attributes:

Attribute Name	Required	Default	Description
LABEL	No		A reference to an externalized string that should be used as the associated label for this container.
LA-BEL_ABBREVIATION	No		A reference to an externalized string containing the associated label abbreviation text for this container. This label abbreviation is placed only on table headers in a LIST.
WIDTH	No	100	The percentage of the width of the field value cell in the cluster or list that the container should occupy.
ALIGNMENT	No	DEFAULT	Defines the horizontal alignment of the elements within the container. Can be set to LEFT, RIGHT, CENTER, or DEFAULT. The value DEFAULT corresponds to the CSS class default in curam_common.css. Currently the default is to be left aligned.
SEPARATOR	No		A reference to an externalized string to use as the separator between the elements within the container.
STYLE	No		A CSS class to be applied to this container.

Table 5.9 Attributes of the CONTAINER Element

Child Elements

The CONTAINER element can contain the following child elements. It must

contain at least one element.

Element Name	Cardinality / Description
FIELD	0..n. The FIELD, ACTION_CONTROL, IMAGE and WIDGET elements can be freely intermingled.
IMAGE	0..n. The FIELD, ACTION_CONTROL, IMAGE and WIDGET elements can be freely intermingled.
ACTION_CONTROL	0..n. The FIELD, ACTION_CONTROL, IMAGE and WIDGET elements can be freely intermingled.
WIDGET	0..n. The FIELD, ACTION_CONTROL, IMAGE and WIDGET elements can be freely intermingled.

Table 5.10 Child Elements of the CONTAINER Element

5.9.9 DETAILS_ROW

The DETAILS_ROW element is used within a LIST element to enable each row to be expanded to show more details about the row. Child elements of DETAILS_ROW define the content that is displayed when the row is expanded. Currently only the INLINE_PAGE element is supported as a child.

When a page a page containing a list with expanded rows is submitted to self or refreshed after a dialog submit, the rows will be re-expanded after the page loads again. This functionality is based on page parameters to the corresponding INLINE_PAGE and the following limitations apply:

- The INLINE_PAGE must take page parameters and they must uniquely identify each row within the list.
- The functionality is supported for pages submitted to self or refreshed after a dialog submit. In all other cases all rows after refresh are reset to default - collapsed.
- If the list contains duplicate items, only the first of them will retain the expanded state after refresh.
- If an edit operation in a dialog changes values that are used in the INLINE_PAGE parameters, this row will be collapsed after refresh.
- If an expanded row is expandable conditionally and it is no longer expandable after the page is refreshed, its state will be always set to collapsed.

Note that DETAILS_ROW element is not allowed in a list using the SCROLL_HEIGHT attribute.

Attributes

The DETAILS_ROW element has the following attribute.

Attribute Name	Required	Default	Description
MINIMUM_EXPANDED_HEIGHT	No	30px	Specifies minimum height in pixels of an expanded row for this list. To be used for in-line pages that are expected to contain nested lists with long actions menus which would not fit to the default expanded row height.

Table 5.11 Attributes of the DETAILS_ROW Element

Child Elements

The DETAILS_ROW element contains the following child elements.

Element Name	Cardinality / Description
INLINE_PAGE	1..1 This defines the page to be shown when the list row is expanded. Currently this is the only supported element, hence it's 1..1 cardinality.
CONDITION	0..1. Affects whether or not the details row is displayed.

Table 5.12 Child Elements of the INFORMATIONAL Element

5.9.10 DESCRIPTION

The DESCRIPTION element defines the description associated with a PAGE_TITLE, CLUSTER or LIST element. A DESCRIPTION is constructed by concatenating a number of connection sources together.

Attributes

The DESCRIPTION element has the following attributes:

Attribute Name	Required	Description
SEPARATOR	No	A reference to an externalized string to use as the separator between the elements within the container.

Table 5.13 Attributes of the DESCRIPTION Element

Child Elements

The DESCRIPTION element can contain child elements as follows:

Element Name	Cardinality / Description
CONNECT	1..n. Only CONNECT elements containing SOURCE elements can be included (one SOURCE per CONNECT). Sources can be server interface properties or, with the NAME attribute set to TEXT, references to strings in a properties file.

Table 5.14 Child Elements of the DESCRIPTION Element

5.9.11 FIELD

The FIELD element specifies a data value to be displayed in a CLUSTER, a value to be retrieved from the user via an input control in a CLUSTER, or a list of data values to be displayed in a LIST column. FIELD elements can also be aggregated within CONTAINER elements so that they fill a single cell of a CLUSTER or LIST element.

Attributes

The FIELD element has the following attributes:

Attribute Name	Required	Default	Description
LABEL	No		A reference to an externalized string that should be used as the associated label for this field. The LABEL attribute is mandatory when a CONNECT element exists, that contains a TARGET.
LA-BEL_ABBREVIATION	No		A reference to an externalized string containing the associated label abbreviation text for this field. This label abbreviation is placed only on table headers in a LIST.
DESCRIPTION	No		A reference to an externalized string that is displayed below the label text.
ALT_TEXT	No		A reference to an externalized string that is used as the altern-

Attribute Name	Required	Default	Description
			ate text for the field. This is only applicable when the field has a target connection, i.e. it is an input field. If this attribute is not specified the LABEL is used. Browsers supported by the Cúram application display alternate text when the mouse is hovered over the input control.
WIDTH	No		Specifies the width of the field value within its cluster or list cell.
WIDTH_UNITS	No	PERCENT	The units in which the width is interpreted. This can be PERCENT to indicate the percentage of the space available to the field, or CHARS to indicate the number of visible characters wide the field will be.
HEIGHT	No	1	For input fields that resolve to a text input control, this specifies the number of visible lines of text that the control will display. For input fields that resolve to a selection list, this specifies the number of entries that are initially displayed. i.e. a scrollable selection list is displayed instead of a drop-down selection list.
ALIGNMENT	No	DEFAULT	Defines the horizontal alignment of the field value. Can be set to LEFT, RIGHT, CENTER, or DEFAULT. The value DEFAULT corresponds to the CSS class default in curam_common.css. Currently the default is to be left aligned. In a CLUSTER, only input fields are aligned. In a LIST, all fields are aligned.
USE_DEFAULT	No	true	If set to true (the default) and the field has no SOURCE connection, then if a sensible default value for the field can be determined automatically, it

Attribute Name	Required	Default	Description
			<p>will be displayed.</p> <p>For example, numeric fields will display a zero, string fields will be empty, date fields will default to the current date, etc.</p>
USE_BLANK	No	false	<p>If the field source is a code-table based property, or a server interface list property, it will be displayed in a list. If this attribute is set to <code>true</code>, an extra blank value will be added to the top of the list.</p>
CONTROL	No	DEFAULT	<p>The CONTROL attribute can take one of a number of values:</p> <p>DEFAULT: the field behaves in the standard fashion.</p> <p>SUMMARY, DYNAMIC, DYNAMIC_FULL_TREE and FAILURE: these settings only apply to rules fields. See Section 8.9, <i>Rules Trees</i> for further details.</p> <p>SKIP: indicates that the field is only present to occupy space in a CLUSTER to balance the layout. No label or value will be displayed. The label background will still be presented, however.</p> <p>TRANSFER_LIST: Enables a list on a page to be displayed as a transfer list widget. This mode is only applicable and supported for list controls with multiple selection capability.</p> <p>CT_HIERARCHY_HORIZONTAL displays a list as a horizontal code table hierarchy.</p> <p>CT_HIERARCHY_VERTICAL displays a list as a vertical code table hierarchy. Consult the <i>Cúram Server Developers Guide</i> for more information on</p>

Attribute Name	Required	Default	Description
			code table hierarchies.
CONFIG	No		<p>Identifies configuration details for this FIELD instance. This attribute can only be used in conjunction with a FIELD whose CONTROL attribute is for a widget that supports configuration. For example, if the CONTROL attribute is DYNAMIC for a FIELD of the RESULT_TEXT domain then the CONFIG attribute should match an ID on a config element in the RulesDecisionConfig.xml file. See Section 8.9.5, <i>Dynamic Rules View</i> for further details on configuration.</p> <p>CT_DISPLAY_LABELS: Displays labels for each code table in a code table hierarchy. See the CONTROL attribute in Section 5.9.11, <i>FIELD</i> for further information regarding code table hierarchies.</p>
INITIAL_FOCUS	No	false	<p>A FIELD element whose INITIAL_FOCUS attribute is set to true will get focus when the page is displayed. In other words, the cursor will be placed in that field ready for data entry. If no FIELD requests the initial focus, the cursor will be placed in the first input field on the page. It is not allowed to have more than one FIELD with the INITIAL_FOCUS attribute set to true specified on a page.</p>
PROMPT	No	false	<p>The setting of this attribute will allow for prompt to appear in the text field if the text field is blank. On focus, the prompt will disappear to allow for data entry.</p>

Table 5.15 Attributes of the FIELD Element

Child Elements

The FIELD element can contain the following child elements:

Element Name	Cardinality / Description
CONNECT	<p>0..3. A field can contain up to three CONNECT elements. The SOURCE connection defines the initial value for the field (this will be the static value shown if there is no target end-point, or the initial value of an input control if there is a target end-point). The TARGET end-point defines the property that will be set from the field value during the action phase. If a TARGET end-point is specified the SOURCE end-point can only be from a server interface property. This is because domain information is required to correctly format the value for display in the input control.</p> <p>If an INITIAL end-point is used and the property is not a list value, it specifies the visible value of the field (which will be read-only). The SOURCE value will be hidden, and the pair of values can only be changed via a pop-up search page. The TARGET end-point will be supplied with the hidden value.</p> <p>If an INITIAL end-point is used and the property is a list value, it specifies the visible values in a drop-down list. The INITIAL element's HIDDEN_PROPERTY specifies the corresponding list of hidden values that will be supplied to the TARGET end-point. In this instance, the SOURCE end-point specifies one of the hidden values in the list that should be used as the initial list selection (the corresponding visible value is displayed).</p>
LINK	<p>0..1. Only valid for output fields (those with no TARGET connection end-point). The value of the output field will be used as the text for the hyperlink specified by this LINK element.</p> <p>If the field is based on a domain which requires a pop-up window then the LINK element can be used to supply parameters to the pop-up page. In this case the LINK element must not have a PAGE_ID attribute specified. See Section 8.21.3, <i>Using the Pop-up Page</i> for further details.</p>
LABEL	<p>0..1. Allows the label for a FIELD to be constructed from a number of sources. If both a LABEL attribute and LABEL child element are specified, the element takes precedence. See Section 5.9.21, <i>LABEL</i> for more details.</p>

Element Name	Cardinality / Description
SCRIPT	0..n. A script file associated with this FIELD that contains JavaScript code to be activated in response to the specified event on the field control. See Section 5.9.28, <i>SCRIPT</i> for more details and limitations on this element usage.

Table 5.16 Child Elements of the FIELD Element

5.9.12 FOOTER_ROW

The FOOTER_ROW element is used to define a single footer row at the end of a list. A list can have multiple footer rows.

A FOOTER_ROW element may only contain FIELD elements. The number of FIELD elements must match the number of columns in the parent list.

There are two CSS classes associated with footer row fields. A FIELD with a TEXT SOURCE connection is output with the footerheader CSS class. All other SOURCE connections are output with the footervalue CSS class. Both of these classes are defined in curam_common.css and can thus be customized.

Spanning column widths are supported through the use of skip fields. For instance, if one normal field and two skip fields are used in a FOOTER_ROW element, this normal field will span three columns. Example code is shown below.

```
<LIST TITLE="List.Title.One" DESCRIPTION="List.Description.One">
  <FIELD LABEL="Field.Title.BankId" WIDTH="40">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="dtls$entitlement"/>
    </CONNECT>
  </FIELD>
  <FIELD LABEL="Field.Title.Name" WIDTH="35">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="dtls$date"/>
    </CONNECT>
  </FIELD>
  <FIELD LABEL="Field.Title.VersionNo" WIDTH="25">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="dtls$total"/>
    </CONNECT>
  </FIELD>

  <FOOTER_ROW>
    <FIELD CONTROL="SKIP"/>
    <FIELD WIDTH="40" LABEL="Field.Title.Footer" >
      <CONNECT>
        <SOURCE NAME="TEXT" PROPERTY="Footer.Text.Entitlement"/>
      </CONNECT>
    </FIELD>
    <FIELD>
      <CONNECT>
        <SOURCE NAME="DISPLAY" PROPERTY="dtls$entitlement"/>
      </CONNECT>
    </FIELD>
  </FOOTER_ROW>
```

```

</LIST>
<LIST>
  <FIELD WIDTH="40">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="dtls$deduction"/>
    </CONNECT>
  </FIELD>
  <FIELD WIDTH="35">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="dtls$date"/>
    </CONNECT>
  </FIELD>
  <FIELD WIDTH="25">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="dtls$total"/>
    </CONNECT>
  </FIELD>

  <FOOTER_ROW>
    <FIELD CONTROL="SKIP"/>
    <FIELD WIDTH="40" LABEL="Field.Title.Footer" >
      <CONNECT>
        <SOURCE NAME="TEXT" PROPERTY="Footer.Text.Deductions"/>
      </CONNECT>
    </FIELD>
    <FIELD>
      <CONNECT>
        <SOURCE NAME="DISPLAY" PROPERTY="dtls$subTotal"/>
      </CONNECT>
    </FIELD>
  </FOOTER_ROW>

  <FOOTER_ROW>
    <FIELD CONTROL="SKIP"/>
    <FIELD>
      <CONNECT>
        <SOURCE NAME="TEXT" PROPERTY="Footer.Text.Payment"/>
      </CONNECT>
    </FIELD>
    <FIELD>
      <CONNECT>
        <SOURCE NAME="DISPLAY" PROPERTY="dtls$payment"/>
      </CONNECT>
    </FIELD>
  </FOOTER_ROW>
</LIST>

```

Example 5.3 Example of a FOOTER_ROW in a List.

Attributes

The FOOTER_ROW element has no attributes.

Child Elements

The FOOTER_ROW element contains the following child elements.

Element Name	Cardinality / Description
FIELD	1..n Each FOOTER_ROW must contain the same number FIELD elements as there are columns in the parent LIST.

Table 5.17 Child Elements of the FOOTER_ROW Element

5.9.13 IMAGE

The `IMAGE` element inserts an image into a `CONTAINER`.

Attributes

The `IMAGE` element has attributes as follows:

Attribute Name	Required	Default	Description
<code>IMAGE</code>	Yes		A reference to an entry in the <code>Image.properties</code> file.
<code>LABEL</code>	Yes		The entry in the UIM's associated properties file which is used as the alternate (or "alt") text of the image.
<code>STYLE</code>	No		A CSS style to associate with the image.

Table 5.18 Attributes of the `IMAGE` Element

Child Elements

The `IMAGE` element has no child elements.

5.9.14 INCLUDE

The `INCLUDE` element indicates that the elements within an external UIM view document should be included at this position in the page.

Attributes

The `INCLUDE` element has attributes as follows:

Attribute Name	Required	Default	Description
<code>FILE_NAME</code>	Yes		The file name of the UIM view document to be included. No path to the file should be specified. The file name alone is sufficient to identify the document.

Table 5.19 Attributes of the `INCLUDE` Element

Child Elements

The INCLUDE element has no child elements.

5.9.15 INITIAL

This element is only valid within a CONNECT element contained in a FIELD element. Use of this connection type is described in further detail in the following sections:

- For pop-up pages see Section 8.21, *Pop-up Pages*
- For selection lists populated from server interface properties see Section 8.7, *Selection Lists*

Attributes

The INITIAL element has the following attributes:

Attribute Name	Re- quired	Default	Description
NAME	Yes		The name of the SERVER_INTERFACE instance to use as the source of the property value.
PROPERTY	Yes		The source of the data to be displayed in the visible field. This can be a list or a non-list field type.
HIDDEN_PROPERTY	No		The source of the list data that has a one-to-one mapping (based on the list indexes) to the list property specified in the PROPERTY attribute.

Table 5.20 Attributes of the INITIAL Element

Child Elements

The INITIAL element contains no child elements.

5.9.16 INFORMATIONAL

The INFORMATIONAL element is used to display informational messages returned from the server. These are different to error messages in that the server call completes successfully. The messages are created in server side code using the SDEJ Informational Manager API (see the *Cúram Server Developers Guide* for more details). This API allows a developer to assign

messages to an output list field(s). This field must then be referenced using child `CONNECT` elements. The message will be displayed at the top of the page in the same area as error messages and this may not be on the page on which the `INFORMATIONAL` element was defined. It could be on the following page or on the parent page in the case of modal dialogs. Finally, messages will never be displayed within the context panel of the application, but will instead will always be displayed within the main content area of the page.

Attributes

The `INFORMATIONAL` element has no attributes.

Child Elements

The `INFORMATIONAL` element contains the following child elements.

Element Name	Cardinality / Description
<code>CONNECT</code>	1..n Each <code>CONNECT</code> element specifies a single <code>SOURCE</code> end-point. This is a field of a bean which contains informational messages.

Table 5.21 Child Elements of the `INFORMATIONAL` Element

5.9.17 `INLINE_PAGE`

The `INLINE_PAGE` element is used to display the contents of one UIM page *in-line* in another. Currently this is only supported within the `DETAILS_ROW` element of a `LIST` to support displaying extra content when a list row is expanded.

Attribute

The `INLINE_PAGE` element has the following attributes:

Attribute Name	Re-quired	Default	Description
<code>PAGE_ID</code>	Yes		The ID of the UIM page to display. Circular dependencies must not be introduced. If a page is used inline, it is not allowed for it to be mapped to a tab at the same time.
<code>URI_SOURCE_NAME</code>	No		The name of the <code>SERV-ER_INTERFACE</code> instance to use as the source of the URI. This attribute is paired with

Attribute Name	Re- quired	Default	Description
			URI_SOURCE_PROPERTY. Note that a URI can only be sourced from a server interface. This attribute cannot be used to specify page parameters or properties files as a source for the URI. The server interface reference must be called during the “display-phase” and the parent ACTION_CONTROL must be of type ACTION when this property is used.
URI_SOURCE_PROPE RTY	No		The name of the property to use as the source of the URI.

Table 5.22 Attributes of the INLINE_PAGE Element

Child Elements

The INLINE_PAGE element contains the following child elements.

Element Name	Cardinality / Description
CONNECT	0..n. Connections on this element define the parameters to be exported to the page targeted by the INLINE_PAGE elements PAGE_ID attribute. The CONNECT should contain both a SOURCE and a TARGET element and the TARGET element should have the NAME attribute set to PAGE and the PROPERTY attribute set to the name of the page parameter.

Table 5.23 Child Elements of the INLINE_PAGE Element

Restrictions on usage

The UIM page opened in an expanded row is intended for only viewing additional information about the row. It should not be used for editing information about that row. Instead a modal dialog should be launched from the page when an edit is required.

As these pages are for viewing information only, the following rules/restrictions should be noted for these "in-line" pages.

- The "in-line" pages displayed in an expanded row must not be used for editing information.
- The "in-line" pages displayed in an expanded row should not display very complex widgets that require a "full screen". This includes the following domain specific controls and UIM elements:
 - Decision Assist: The Decision Matrix Widget
 - Decision Assist: Typical Picture Editor Widget
 - Decision Assist: Evidence Review Widget
 - Agenda Player
 - Batch Function View
 - The Rules Simulation Editor
 - The Rates Table
 - The Meeting View Widget
 - The FILE_EDIT Widget
 - The Calendar
 - Rules Trees



Note

There are no validations in place for these restrictions and it is the responsibility of the developer to ensure they don't use unsupported widgets in an expandable list.

5.9.18 IS_FALSE

A Boolean test to evaluate if the parent CONDITION succeeds or fails. This element evaluates to true when the referenced property value is false.

Attributes

The IS_FALSE element has the following attributes:

Attribute Name	Re-quired	Default	Description
NAME	Yes		The name of the SERVER_INTERFACE instance to use as the source of the property value.
PROPERTY	Yes		The name of the property being accessed. It must be a Boolean value.

Table 5.24 Attributes of the IS_FALSE Element

See Section 5.9.19.1, *Attributes* for more details on the use of this element to access the values of action-phase server interface properties.

Child Elements

The IS_FALSE element contains no child elements.

5.9.19 IS_TRUE

A Boolean test to evaluate if the parent CONDITION succeeds or fails. This element evaluates to true when the referenced property value is true.

Attributes

The IS_TRUE element has the following attributes:

Attribute Name	Re- quired	Default	Description
NAME	Yes		The name of the SERVER_INTERFACE instance to use as the source of the property value.
PROPERTY	Yes		The name of the property being accessed. It must be a Boolean value.

Table 5.25 Attributes of the IS_TRUE Element

In the majority of cases the NAME and PROPERTY combination will reference a display-phase server interface property. However when a page submits to itself using an ACTION_CONTROL with a child LINK element that has the PAGE_ID set to THIS (e.g., a search page), properties of the action-phase server interface can be referenced. When the page is first displayed the action-phase server interface will not be in scope and the property is treated as if its value is false. When the page is submitted, the action-phase server interface will be in scope and the referenced property will be evaluated as normal.

Child Elements

The IS_TRUE element contains no child elements.

5.9.20 JSP_SCRIPTLET

The JSP_SCRIPTLET element defines JSP scriptlet code that should be inserted into the page at that point relative to any other LIST or CLUSTER

elements. Any TextHelper beans declared by a SERVER_INTERFACE element to be in the DISPLAY phase are available to the scriptlet by getting the attribute of the page context with the same name as the NAME attribute of the SERVER_INTERFACE element. An example is shown in Example 5.4, *Example JSP_SCRIPTLET Accessing a TextHelper* below.

```
<SERVER_INTERFACE NAME="MyBeanName" CLASS="MyClass"
  OPERATION="getMyData" />
<JSP_SCRIPTLET>
  <![CDATA[
    curam.omega3.texthelper.TextHelper th =
      pageContext.findAttribute("MyBeanName");
    String myValue = th.getFieldValue("myPropertyName");
    out.print("VALUE: " + myValue);
  ]]>
</JSP_SCRIPTLET>
```

Example 5.4 Example JSP_SCRIPTLET Accessing a TextHelper

As the code within the JSP_SCRIPTLET element may contain reserved XML characters¹, you can either replace these characters with the appropriate XML character entity or enclose the contents of the element in the CDATA (“character data”) block as shown above which will prevent the XML parser from trying to interpret the contents of the block.

A common use of the JSP_SCRIPTLET element is to write code that will redirect the current page to another page. Example 5.5, *Example JSP_SCRIPTLET Redirecting to a Page*, shows an example of this.

```
<PAGE PAGE_ID="Activity_resolveAttendeeHome">
  <JSP_SCRIPTLET>
    <![CDATA[
      curam.omega3.request.RequestHandler rh
        = curam.omega3.request.RequestHandlerFactory
          .getRequestHandler(request);
      String context = request.getContextPath() + "/";
      context += curam.omega3.user.UserPreferencesFactory
        .getUserPreferences(pageContext.getSession())
          .getLocale() + "/";
      String url = context + "UserCalendarPage.do?"
        + "startDate=&calendarViewType=CVT3";
      url += "&" + rh.getSystemParameters();
      response.sendRedirect(response.encodeRedirectURL(url));
    ]]>
  </JSP_SCRIPTLET>
</PAGE>
```

Example 5.5 Example JSP_SCRIPTLET Redirecting to a Page

This demonstrates the API used to access the system parameters that control an application's ability to return to previous pages. The information about the previous page is stored in the system parameters accessible via the RequestHandler.getSystemParameters() method. By adding the system parameters, any *Cancel* button on the following page will return to the expected page when clicked. The RequestHandlerFactory.getRequestHandler() method is passed the JSP request object and will return the appropriate request handler. The system parameters should be appended to the redirect URL and just require a separating “&” character as they are already formatted in

value pairs.

When using a `JSP_SCRIPTLET` to redirect to another page, the `JSP_SCRIPTLET` should be the only child element of the `PAGE` element. When this is the case, no HTML content will be generated for the page: it will not be displayed, so no HTML is required. If other elements are present, then HTML content will be generated. This can include the page header, navigation menus, footer, title, etc. If this HTML content exceeds the size of the buffer on the web container serving the page, then the content will be transmitted to the web browser. Once any content is transmitted in this way, the redirect operation will have no effect. Therefore, ensuring that the page contains a single `JSP_SCRIPTLET` element and no other elements will ensure that the redirect operation works as expected.

If you need to access a `TextHelper` instance from a JSP scriptlet that redirects to another page, then you cannot use the `SERVER_INTERFACE` element to declare the `TextHelper` as shown in Example 5.4, *Example JSP_SCRIPTLET Accessing a TextHelper*, as this extra element would cause HTML content to be generated. Instead, you must declare the `TextHelper` instance within the scriptlet code as shown below.

It should be noted that, when using `JSP_SCRIPTLET`, there is limited error handling capability. Thus, code should not make calls to secured server interface methods. Instead, the target page of any `JSP_SCRIPTLET` should be secured appropriately.

```
<PAGE PAGE_ID="Activity_resolveApplicationHome">
  <JSP_SCRIPTLET>
    <![CDATA[
      curam.omega3.request.RequestHandler rh
        = curam.omega3.request.RequestHandlerFactory
          .getRequestHandler(request);
      String context = request.getContextPath() + "/";
      context += curam.omega3.user.UserPreferencesFactory
        .getUserPreferences(pageContext.getSession())
          .getLocale() + "/";
      String activityID = request.getParameter("ID");
      String eventType = request.getParameter("TYPE");
      String url = context;

      curam.interfaces.ActivityPkg.Activity_readDescription_TH
        th = new curam.interfaces.ActivityPkg
          .Activity_readDescription_TH();
      th.setFieldValue(
        th.key$activityDescriptionKey$activityID_idx,
        activityID);
      th.callServer();

      String description = th.getFieldValue(
        th.result$activityDescriptionDetails$description_idx);
      if (eventType.equals("AT1")) {
        url = "Activity_viewUserRecurringActivityPage.do?";
      } else {
        url = "Activity_viewUserStandardActivityPage.do?";
      }
      url += "activityID=" + activityID;
      url += "&description="
        + curam.omega3.request.RequestUtils.escapeURL(
          description);
      url += "&" + rh.getSystemParameters();
      response.sendRedirect(response.encodeRedirectURL(url));
    ]]>
  </JSP_SCRIPTLET>
```

```
</PAGE>
```

Example 5.6 Example JSP_SCRIPTLET Redirecting and Accessing a TextHelper

When adding parameters to the parameter list, care must be taken if the parameter value may contain non-ASCII characters. Values containing non-ASCII characters must be escaped before they are added to the parameter list to ensure that the characters are preserved correctly. The `RequestUtils.escapeURL(String)` method can be used to perform the escaping. An example of the Java code to perform this escaping is shown in the example above. Code following that pattern should be included within your JSP scriptlet.

Attributes

The `JSP_SCRIPTLET` element has no attributes.

Child Elements

The `JSP_SCRIPTLET` element contains no child elements. The body of the element must only contain the JSP scriptlet code to be inserted into the page.

5.9.21 LABEL

The `LABEL` element can be used as a child element of `FIELD` to construct a label by concatenating multiple values. An example of the field and label data is shown in Example 5.7, *Example of a Dynamic LABEL*, below.

```
<CLUSTER TITLE="Cluster.Title">
  <FIELD>
    <LABEL>
      <CONNECT>
        <SOURCE NAME="TEXT" PROPERTY="Label.Text" />
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="DISPLAY" PROPERTY="personName" />
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="TEXT" PROPERTY="Label.Separator" />
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="DISPLAY" PROPERTY="dateOfBirth" />
      </CONNECT>
    </LABEL>

    <CONNECT>
      <TARGET NAME="ACTION" PROPERTY="fieldName" />
    </CONNECT>
  </FIELD>
</CLUSTER>
```

Example 5.7 Example of a Dynamic LABEL

Attributes

The LABEL element has no attributes:

Child Elements

The LABEL element can contain the following child elements.

Element Name	Cardinality / Description
CONNECT	1..n. A CONNECT element specifying a single SOURCE end-point. Action-phase server interfaces cannot be used in the SOURCE end-point.

Table 5.26 Child Elements of the LABEL Element

5.9.22 LINK

The LINK element specifies the page to go to after an action phase. Alternatively, a LINK element can specify any external web page or certain resource. Links can contain CONNECT elements to map values to parameters to be added to the link.

Attributes

The LINK element has the following attributes. Note that the PAGE_ID, PAGE_ID_REF, URL, URI, and URI_REF attributes are mutually exclusive as well as the pair of attributes URI_SOURCE_NAME and URI_SOURCE_PROPERTY.

Please note that attributes that have the ability to link to external web pages or resources (i.e mailto: links) will have their link back functionality stripped away. This link back functionality keeps a link to the previous page. An example of where this is needed is with cancel buttons where if they are used, the page will link back to the previous page. In order to keep this, the link will have to be to an internal Curam page. In order to mark a link as being a link to an internal Curam page, the keyword 'curam:' needs to be added before the link text.

Attribute Name	Required	Default	Description
PAGE_ID	No		<p>The unique identifier of the page to be opened. This is the value of the PAGE_ID attribute of the PAGE element in the required UIM page document.</p> <p>If this attribute is set to the PAGE_ID of the current page, the page will be re-opened with all the input fields reset to their default state.</p> <p>If the link is on an action control</p>

Attribute Name	Required	Default	Description
			with a <code>TYPE</code> set to <code>SUBMIT</code> and this attribute is set to the value <code>THIS</code> , the link will return to the current page after the action phase and the input fields will not be reset to their default state. This is useful for search pages where the search criteria need to be preserved.
<code>PAGE_ID_REF</code>	No		A <code>PAGE_ID</code> can alternatively be specified by reference to an entry in the <code>Curam-Links.properties</code> file. This allows many links to refer to the same target page yet all can be updated by changing the entry in the <code>Curam-Links.properties</code> file.
<code>URL</code>	No		It is recommended to use the new <code>URI</code> attribute which is described below. The <code>URL</code> attribute is maintained for backward compatibility.
<code>URI</code>	No		Rather than link to another page in the application, the <code>URI</code> attribute allows the creation of a link to any <code>URI</code> whatsoever. This can be used to link to pages or other resources completely outside of the application. Parameters must be supplied by <code>CONNECT</code> elements within the <code>LINK</code> to ensure correct encoding.
<code>URI_REF</code>	No		A <code>URI</code> (or <code>URL</code>) can alternatively be specified by reference to an entry in the <code>Curam-Links.properties</code> file. This allows many links to refer to the same target yet all can be updated by changing the entry in the <code>CuramLinks.properties</code> file. The file can be placed in any component in the application.
<code>URI_SOURCE_NAME</code>	No		The name of the <code>SERV-ER_INTERFACE</code> instance to use

Attribute Name	Required	Default	Description
			as the source of the URI. This attribute is paired with <code>URI_SOURCE_PROPERTY</code> . Note that a URI can only be sourced from a server interface. This attribute cannot be used to specify page parameters or properties files as a source for the URI. The server interface reference must be called during the “display-phase” and the parent <code>ACTION_CONTROL</code> must be of type <code>ACTION</code> when this property is used.
<code>URI_SOURCE_PROPERTY</code>	No		The name of the property to use as the source of the URI.
<code>OPEN_NEW</code>	No	<code>false</code>	When set to <code>true</code> , this flag indicates that the linked page should be opened in a new window. When set to <code>false</code> (the default) the linked page will be opened in the current window. This setting is only supported for links to external sites.
<code>SAVE_LINK</code>	No	<code>true</code>	This attribute indicates that the page containing the link should be returned to if an action control on the target page is configured to return to the previous page. An action control without a <code>LINK</code> child element will return the user to the previous page. If there is a sequence of pages and any one of them needs to go back to a “starting” page, then each page in the sequence should set this attribute to <code>false</code> so that subsequent pages do not return to their immediate previous page in the chain.
<code>SET_HIERARCHY_RETURN_PAGE</code>	No	<code>false</code>	This attribute is no longer used but has been retained in the UIM schema to avoid upgrade impact.
<code>USE_HIERARCHY_RETURN_PAGE</code>	No	<code>false</code>	This attribute is no longer used but has been retained in the UIM schema to avoid upgrade impact.

Attribute Name	Required	Default	Description
HOME_PAGE	No		<p>If this attribute is set to <code>true</code>, the link will take a user directly to their home page. During development the home page can be configured by setting the “application code” field of the Cúram “users” table. This value of this field corresponds to an entry on the APPLICATION_CODE code-table. At runtime, the Cúram Administration application allows the home page to be set when creating or editing a user.</p> <p>Note, that in the development environment Java EE security is not enabled. Therefore, since a user name is not available the home page link cannot be displayed.</p>
OPEN_MODAL	No	"false"	<p>If this attribute is set to <code>true</code>, the link will open the referenced page in a new window. The new window is modal, meaning that while it is open the parent window cannot be accessed. When a user navigates from the original page in the modal dialog, either by submitting a form or clicking a link, the modal dialog is closed, and the parent page that spawned it is sent to the new location. This behavior is only supported in Internet Explorer, all other browsers will simply open a normal pop-up window.</p>
DISMISS_MODAL	No	"true"	<p>If this attribute is set to <code>false</code>, the link will open the referenced page in the same pop-up window, modal or normal depending on what the browser supports.</p>
WINDOW_OPTIONS	No	"width=800,height=450"	<p>The size of each modal dialog is configurable using this parameter. The value of the attribute is a comma separated list of name value pairs. The currently supported options are <code>width</code> and</p>

Attribute Name	Required	Default	Description
			height, both of which take an integer value, which is translated directly to a pixel value. Any other parameters will cause an exception to be thrown. This attribute should only be set when OPEN_MODAL is set to true on the same LINK tag.

Table 5.27 Attributes of the LINK Element

Child Elements

The LINK element can contain the following child elements:

Element Name	Cardinality / Description
CONNECT	0..n. Connections on a link define the parameters to be exported to the page targeted by the link. The CONNECT should contain both a SOURCE and a TARGET element and the TARGET element should have the NAME attribute set to PAGE and the PROPERTY attribute set to the name of the page parameter. Any type of SOURCE element can be used with the following exceptions. TEXT and where the LINK is inside an ACTION_CONTROL with TYPE=SUBMIT. In the last scenario, the SOURCE must have an ACTION phase bean, a page parameter or a CONSTANT. The reason being the URL is generated in the action class and the DISPLAY bean is not accessible at the stage.
CONDITION	0..1. Affects whether or not the link is displayed.

Table 5.28 Child Elements of the LINK Element

Modal Dialogs

A Modal Dialog is similar to a Pop-up Page, in that it opens a dialog box to display a page on top of the main application content. However, modal dialog is different in a number of ways.

- When a modal dialog is open, its parent page cannot be accessed. The parent page is grayed-out and ignores any user action.

- Changing the page in the Modal Dialog, either by submitting a form or by clicking a hyperlink, causes it to close, and the parent page to be changed to the changed page, with the following exceptions
 - If the page linked to has the same id as the current modal page (e.g. a 'save & new' button/link), then the page will be refreshed within the same modal window
 - If the link clicked has the attribute `DISMISS_MODAL` set to `false`, the page linked to will be opened within the same modal window
 - If the link clicked has the attribute `OPEN_MODAL` set to `true`, it will open in a new modal window
- The usage of Modal Dialogs is different to that of Pop-up pages. It is considerably less complex, consisting of using either one or two optional attributes on the `LINK` tag.

Using Modal Dialogs

A `LINK` tag is made to open in a Modal Dialog, rather than the default action of opening a new page in the same window, by setting the `OPEN_MODAL` attribute to `true`.

```
<LINK PAGE_ID="MultiSelectWidgetResult" OPEN_MODAL="true" />
```

Note in the example the use of the `OPEN_MODAL` attribute on the `LINK` tag. Setting `OPEN_MODAL` on a `LINK` that is inside an `ACTION_CONTROL` of type `SUBMIT` has no effect. Setting `OPEN_MODAL=true` on a link implies also having `DISMISS_MODAL=false` on that link, and setting `DISMISS_MODAL=true` on it is ignored. Setting `DISMISS_MODAL=false` implies `OPEN_MODAL=false`, so there is no need to set it.

Configuring Modal Dialogs

Modal Dialogs can be individually configured by setting the `WINDOW_OPTIONS` attribute on a `LINK` tag which also has the `OPEN_MODAL` attribute set to `true`. Multiple options can be set using this attribute, which is formatted as a comma separated list of name value pairs. The currently supported parameters are

- `width` - sets the width of the Modal Dialog, measured in pixels. This parameter takes an integer value.
- `height` - sets the height of the Modal Dialog, measured in pixels. This parameter takes an integer value.

```
<LINK PAGE_ID="MultiSelectWidgetResult" OPEN_MODAL="true"
WINDOW_OPTIONS="width=600,height=500" />
```

Note in the example above the use of the `WINDOW_OPTIONS` attribute. The values specified for `width` and `height` are simple integers and do not have any alphabetic characters appended. A default width of 600 pixels is

used if no `width` parameter is specified. If no `height` parameter is specified the height will be automatically calculated to accommodate the page contents. If an unsupported parameter is placed in the `WINDOW_OPTIONS`, a build exception will be thrown.

If the `WINDOW_OPTIONS` attribute is also specified on the `PAGE` element of the page the `LINK` points to, it will take precedence over the value specified on the `LINK` itself.

The minimum required height for modal dialogs can be configured using the property `modal.dialogs.minimum.height` that is located in the `ApplicationConfiguration.properties` file.

Controlling Modal Dialogs from custom JavaScript

Modal Dialogs can be controlled from custom JavaScript by using the provided `curam.util.UimDialog` API. For details see the full API documentation in HTML format, accessible by opening `<cdej-dir>\doc\JavaScript\index.html` in a Web browser.

Loading custom non-UIM pages in a Modal Dialog

Custom non-UIM pages must hook into a specific set of API functions in order to work correctly in a Modal Dialog. These functions are provided by the `curam.util.Dialog` API. The details are available in the full API documentation: `<cdej-dir>\doc\JavaScript\index.html`.

5.9.23 LIST

The `LIST` element defines the layout of a control used to display lists of data. Each field or action control becomes a column and data values are then tabulated.

Attributes

The `LIST` element has the following attributes:

Attribute Name	Required	Default	Description
<code>TITLE</code>	No		A reference to an externalized string containing the title string for this list. See also note below.
<code>STYLE</code>	No		The class name of the CSS style to associate with this list for formatting.
<code>DESCRIPTION</code>	No		A reference to an externalized string that provides more details about the list than the title alone. This will be displayed below the title on the page.

Attribute Name	Required	Default	Description
<code>SORTABLE</code>	No	<code>true</code>	Lists can be sorted by clicking on the appropriate headers. This is set by default to be enabled without the use of the attribute. This attribute allows this feature to be controlled with <code>false</code> disabling the feature and <code>true</code> enabling it.
<code>SUMMARY</code>	No		A reference to an externalized string containing the summary of this list. The <code>SUMMARY</code> attribute describes the purpose and/or structure of a list.
<code>SCROLL_HEIGHT</code>	No		Specifies in pixels the desired fixed height of a scrollable list. A vertical scrollbar is provided once the list exceeds the scroll height. The scrollbar is only applied to the list body and the list's column headers remain fixed. Scroll height is independent of the list contents and therefore an empty list will still be set to the height specified.
<code>BEHAVIOR</code>	No		<p>Optional attribute which controls the display and behavior of the toggle button used to expand or collapse the list.</p> <p>Three value options are available for this attribute:</p> <ul style="list-style-type: none"> <code>NONE</code> which prevents the toggle button from being displayed in the list header. <code>EXPANDED</code>: the toggle button is displayed and the list is initially expanded. <code>COLLAPSED</code>: the toggle button is displayed and the list is initially collapsed. <p>When the <code>BEHAVIOR</code> is not set for a list, its default value of <code>EXPANDED</code> is implied.</p> <p>Note that this attribute is only applicable when the property <code>ENABLE_COLLAPSIBLE_CLUSTER</code></p>

Attribute Name	Required	Default	Description
			RS is not set or is set to <code>true</code> in <code>curam_config.xml</code> . For details see Section 3.12.13, <i>General Configuration</i> .
PAGINATED	No	true	Enables the ability to page through lists displayed in Cúram pages. Any LIST longer than the configured minimum size will display only the first "page" of data and the pagination controls will be displayed below the list.
DE-FAULT_PAGE_SIZE	No	Based on the global configured value, usually 15.	Specifies the page size the list will get by default. The page size can be then changed at runtime by the user.
PAGINATION_THRESHOLD	No	Based on the global configured value, usually same as DE-FAULT_PAGE_SIZE.	Specifies the minimum list size at which pagination will be enabled. For shorter lists there will be no pagination, even if otherwise pagination is switched on.

Table 5.29 Attributes of the LIST Element

**Note**

Lists on search pages now display the number of items found as a result of the search. The number of items will be displayed beside the list title.

The text used to display the number of items can be customized by setting the following property in the `CDEJResources.properties` file, for example:

```
record.number.message=Items found:
```


The actual number of items will be displayed after the text.

This feature only applies to search pages and must be enabled by adding the following to the curam-config.xml file:

```
<LIST_ROW_COUNT>true</LIST_ROW_COUNT>
```

Child Elements

The LIST element can contain the following child elements. It must contain at least one ACTION_CONTROL, FIELD, or CONTAINER element. SOURCE connections can be made to list or non-list properties. Within a table all list properties must belong to the same list structure defined in the server interface model. This ensures that they are all the same length. The number of rows in the list will be equal to the number of elements in the list properties. The value of a non-list property is simply repeated on each row.

Element Name	Cardinality / Description
TITLE	0..1. The TITLE element will be displayed above the LIST.
DESCRIPTION	0..1 The DESCRIPTION element has the same behavior as the DESCRIPTION attribute but allows the description to be built up from a number of sources. If both are specified, this element takes precedence over the corresponding attribute.
ACTION_SET	0..1. The action set can contain ACTION_CONTROL elements of any type. The action controls will be displayed above and/or below the entire list.
FIELD	0..n. The FIELD, CONTAINER, and ACTION_CONTROL elements can be freely intermingled. Only output fields can be used (i.e., fields with no target connection.)
CONTAINER	0..n. The FIELD, CONTAINER, and ACTION_CONTROL elements can be freely intermingled. Within the container, only output fields can be used (i.e., fields with no target connection.)
CONDITION	0..1. Affects whether or not the list is displayed.
FOOTER_ROW	0..n. This should be defined after all other child elements.

Table 5.30 Child Elements of the LIST Element

5.9.24 MENU

The MENU element is used to define six types of menus in a Cúram client application. The menu types are:

- **STATIC:** The menu is made up of ACTION_CONTROL elements that will appear on the page menu. The ACTION_CONTROL elements must have the TYPE of ACTION.
- **NAVIGATION:** The menu is made up of ACTION_CONTROL elements that will be appended to the “Navigation” menu. The ACTION_CONTROL elements must have the TYPE of ACTION.
- **DYNAMIC:** The menu is driven by XML data constructed on the server application.
- **INTEGRATED_CASE:** The menu is driven by XML data constructed on the server application. This menu is specific to the Cúram-style Integrated Case user interface and is rendered as a set of tabs.
- **IN_PAGE_NAVIGATION:** The menu is made up of ACTION_CONTROL elements that will appear on the in-page-navigation menu at the top of the main content area.
- **WIZARD_PROGRESS_BAR:** This is another specific type of menu rendered as a button bar on the top of the content area in a modal dialog for displaying a sequence of related pages in the wizard manner. The menu is driven by a resource stored in the server application.

Attributes

The MENU element has the following attribute:

Attribute Name	Required	Default	Description
MODE	No	STATIC	<p>The type of menu to create. The mode can be STATIC (the default), NAVIGATION, DYNAMIC, INTEGRATED_CASE, IN_PAGE_NAVIGATION or WIZARD_PROGRESS_BAR.</p> <p>Static, navigation and in-page-navigation menus contain one or more ACTION_CONTROL elements that represent links to other pages. The static menu normally appears just above the main content area of the page. Navigation menu items will be appended to the navigation menu, normally on the left of the page. In-page-navigation menu items ap-</p>

Attribute Name	Required	Default	Description
			<p>pear at the top of the main content area and the wizard progress bar appears at the top of the modal dialog content area.</p> <p>Dynamic menus of both types (DYNAMIC and INTEGRATED_CASE) are created from data retrieved from the server and contain a single CONNECT element specifying a SOURCE end-point to a server interface property.</p>

Table 5.31 Attributes of the MENU Element

Child Elements

The MENU element can contain the following child elements. Note that the ACTION_CONTROL and CONNECT elements are mutually exclusive.

Element Name	Cardinality / Description
ACTION_CONTROL	1..n. Only action controls with a TYPE of ACTION can be used.
CONNECT	1. A CONNECT element specifying a single SOURCE end-point.

Table 5.32 Child Elements of the MENU Element

DYNAMIC and INTEGRATED_CASE type menus

The data for both DYNAMIC and INTEGRATED_CASE menu's are driven by the same XML format. An example of the menu data sent by the application server is shown below.

```

<DYNAMIC_MENU>
  <LINK PAGE_ID="CaseHome"
    DESC="2:field1:curam.omega3.myMessages:info_menu1:()"
    TYPE="case" >
    <PARAMETER NAME="caseID" VALUE="1234" />
  </LINK>
  <LINK PAGE_ID="ProductHome"
    DESC="2:field1:curam.omega3.myMessages:info_menu2:()"
    TYPE="product" >
    <PARAMETER NAME="productID" VALUE="5678" />
    <PARAMETER NAME="caseID" VALUE="1234" />
  </LINK>
</DYNAMIC_MENU>

```

Example 5.8 Example of Dynamic MENU Data

All the menu links are contained within the `DYNAMIC_MENU` root element. Each entry on the menu is specified by a `LINK` element. The `LINK` element has the following attributes:

- `PAGE_ID`: Specifies the target page for the link.
- `DESC`: Specifies the server message catalog entry to be looked up and used as the text for the link. The Cúram SDEJ provides an API to create the string representation of a message catalog entry shown in the example above. Consult the Cúram Server Developers Guide for details on using message catalogs.
- `TYPE`: specifies a value that is looked up in appropriate menu configuration file (described below) to identify the icon that should be associated with the link.

Each `LINK` element can contain a number of `PARAMETER` elements that specify additional parameters that will be added to the link from the menu. The `PARAMETER` element has the following attributes:

- `NAME`: The parameter name.
- `VALUE`: The parameter value.

The configuration files for the `DYNAMIC` and `INTEGRATED_CASE` menu's are `DynamicMenuConfig.xml` and `ICDynamicMenuConfig.xml` respectively. The following are examples each configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<DYNAMIC_MENU_CONFIG>
  <SEPARATOR IMAGE="Images/separator.gif"
    TEXT="Dyn.Menu.Separator"/>
  <LINK TYPE="case" IMAGE="Images/case.gif"
    TEXT="Dyn.View.Case"/>
  <LINK TYPE="product" IMAGE="Images/product-delivery.gif"
    TEXT="Dyn.View.Product"/>
</DYNAMIC_MENU_CONFIG>
```

Example 5.9 Example of a **DYNAMIC** Menu Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<INTEGRATED_CASE_MENU_CONFIG>
  <LINK TYPE="case" IMAGE="Images/case.gif"
    TEXT="Dyn.View.Case"/>
  <LINK TYPE="product" IMAGE="Images/product-delivery.gif"
    TEXT="Dyn.View.Product"/>
</DYNAMIC_MENU_CONFIG>
```

Example 5.10 Example of an **INTEGRATED_CASE** Menu Configuration File

The differences to note are the root elements, `DYNAMIC_MENU_CONFIG` and `INTEGRATED_CASE_MENU_CONFIG`, and the `SEPARATOR` element which is not used in an `INTEGRATED_CASE` because of its very specific look and feel.

The `SEPARATOR` element describes an image or a piece of text used to sep-

arate the menu items and has the following attributes:

- **IMAGE:** Specifies an image to use as the separator.
- **TEXT:** Specifies an entry in the `CDEJResources.properties` file. This attribute is mandatory. If an image is specified this will be used as the alternate text for the image, if not, then the text will be displayed.

The **LINK** element has the following attributes.

- **TYPE:** This must match the **TYPE** attribute of the **LINK** element returned from the server application.
- **IMAGE:** Specifies an image to use in the link. This attribute is mandatory.
- **TEXT:** Specifies an entry in the `CDEJResources.properties` file. This attribute is mandatory. It will be used as the alternate text for the image.

The **IN_PAGE_NAVIGATION** type menu

The in-page navigation menu, see User Interface Element 9 of Figure 2.1, *Application User Interface Overview*, allows for the addition of a set of links which will be displayed as tabs embedded within a UIM page. Each UIM page in the set must define the same **MENU** element. The currently selected UIM page, aka tab, is identified by the `STYLE="in-page-current-link"` attribute. This will differ on each of the UIM pages in the set and should be set on the **ACTION_CONTROL** that matches the UIM page the **MENU** is contained in.

```
<PAGE PAGE_ID="InPageNav">
  <PAGE_TITLE>
    <CONNECT>
      <SOURCE NAME="TEXT" PROPERTY="Title.Text"/>
    </CONNECT>
  </PAGE_TITLE>
  <MENU MODE="IN_PAGE_NAVIGATION">
    <ACTION_CONTROL LABEL="Label.page1">
      <LINK PAGE_ID="Page1" SAVE_LINK="false"/>
    </ACTION_CONTROL>
    <ACTION_CONTROL
      LABEL="Page2.Label"
      STYLE="in-page-current-link" >
      <LINK PAGE_ID="Page2" SAVE_LINK="false" />
    </ACTION_CONTROL>
  </MENU>
  .....
</PAGE>
```

Example 5.11 Example of the **IN_PAGE_NAVIGATION** menu in UIM

WIZARD_PROGRESS_BAR menu

The wizard progress menu bar is inserted on a page by including a **MENU**

element which has a `MODE` attribute set to `WIZARD_PROGRESS_BAR`. It binds a number of pages, allowing for the sequential navigation through them. For instance, in a modal dialog which contains a wizard progress menu bar, pages can be navigated through by clicking the previous or next button. At the same time, the wizard progress menu bar presented on the top of it will indicate its progress.

The UIM wizard pages

There are some specifics regarding the UIM pages used with the `WIZARD_PROGRESS_BAR` menu:

- The wizard pages should open in the modal dialog. The wizard progress bar functionality should not be used in standard non-modal UIM pages.
- Each page in the wizard flow is implemented as standard UIM with a wizard progress bar widget placed at the top of each page.
- The pages should have action controls for advancing through the wizard (back and forward buttons as required by the scenario). The `LINK` elements of these action controls should have `DISMISS_MODAL` attribute set to `false` (except for the controls supposed to close the wizard). Additionally, the `SAVE_LINK` attribute should also be set to `false`.

```
<PAGE PAGE_ID="Sample_PageOne">
  <MENU MODE="WIZARD_PROGRESS_BAR">
    <CONNECT>
      <SOURCE
        NAME="DISPLAY" PROPERTY="resourceID" />
    </CONNECT>
  </MENU>
  <PAGE_TITLE>
    <CONNECT>
      <SOURCE NAME="TEXT"
        PROPERTY="PageTitle" />
    </CONNECT>
  </PAGE_TITLE>
  <SERVER_INTERFACE
    CLASS="WizardSample"
    NAME="DISPLAY" OPERATION="getResourceID"
    PHASE="DISPLAY" />
  <ACTION_SET ALIGNMENT="CENTER" TOP="false">
    <ACTION_CONTROL
      LABEL="ActionControl.Label.Cancel" />
    <ACTION_CONTROL
      LABEL="ActionControl.Label.Next">
      <LINK PAGE_ID="Sample_PageTwo"
        SAVE_LINK="false"
        DISMISS_MODAL="false" />
    </ACTION_CONTROL>
  </ACTION_SET>
  .....
</PAGE>
```

Example 5.12 An example of wizard-type menu UIM

In the example above the connection in the `MENU` provides the identifier of the server-side resource describing this wizard (see below).

Wizard menu configuration

The text required by the wizard progress bar items come from a property resource whose identifier must be provided to the wizard progress bar menu.

```
Number.Wizard.Pages=2
Sample_pageOne.Wizard.Item.Text=Child
Sample_pageOne.Wizard.Page.Title=Step 1: Child Details
Sample_pageOne.Wizard.Page.Desc=Capture some details
Wizard.PageID.1=Sample_pageOne

Sample_pageTwo.Wizard.Item.Text=Parent
Sample_pageTwo.Wizard.Page.Title=Step 2: Parent Details
Sample_pageTwo.Wizard.Page.Desc=Capture some details 1
Wizard.PageID.2=Sample_pageTwo
```

Example 5.13 Example of the required properties in the resource store property file

Property Name	Description
Number.Wizard.Pages	The value of this property defines the number of items to be rendered for the wizard progress bar. The value must be a numeric whole number greater than zero.
<PageID>.Wizard.Item.Text	Defines the text to be displayed within the wizard progress bar item for each page of the wizard. There must be one of these properties defined for each page in the wizard. The property is uniquely identified for each wizard page by the <PageID> prefix which represents the actual identifier of that UIM page in the wizard flow.
<PageID>.Wizard.Page.Title	Defines the title to be displayed within the wizard progress bar for the current page of the wizard. There must be one of these properties defined for each page in the wizard. The property is uniquely identified for each wizard page by the <PageID> prefix which represents the actual identifier of that UIM page in the wizard flow.
<PageID>.Wizard.Page.Desc	Defines the description to be displayed within the wizard progress bar for the current page of the wizard. There must be one of these properties defined for each page in the wizard. The property is uniquely identified for each wizard page by the <PageID> prefix which represents the actual identifier of that UIM page in the wizard flow.

Property Name	Description
Wiz- ard.PageID.<PageNum >	Defines the position of the page within the wizard flow. The widget uses this information to style the bar items correctly. There must be one of these properties defined for each page in the wizard. This property is uniquely identified for each wizard page by the <PageNum> suffix which represents the position of each page within the list of wizard menu pages.

Table 5.33 Properties in the wizard defining resource

The order of the properties declaration in the resource is important as the associated menu widget will draw the wizard items for the progress bar in that order. The page title and description are added by the widget for the current page of the wizard.

5.9.25 PAGE

The PAGE element is the root element of a UIM document that describes the data to be included in a generated JSP page.

Attributes

The PAGE element has the following attributes:

Attribute Name	Required	Default	Description
PAGE_ID	Yes		An identifier for the page used when referencing the page from LINK elements. This identifier must be unique within a project. The file name of the document must be the same as the value of this attribute and have the extension .uim.
POPUP_PAGE	No	false	Indicates that this page is a pop-up that will be opened from a parent page. Pop-up pages do not include the side-bar, header and footer of standard pages. The value can be set to true or false. The attribute must only be used for pages configured according to Section 8.21, <i>Pop-up Pages</i> (i.e., search pop-up pages).
SCRIPT_FILE	No		The name of the script file con-

Attribute Name	Required	Default	Description
			<p>taining the JavaScript functions that are specified in the ACTION attribute of any SCRIPT elements on the page. If no SCRIPT_FILE attribute is set on a particular SCRIPT element within a FIELD or ACTION_CONTROL the PAGE script file is used by default. The script file should be added in a component. If another script file has the same name in another component, the version in the highest priority component will be used. Each SCRIPT can specify its own script file if required, or share this common script file.</p>
APPEND_COLON	No		<p>Set to true to automatically append colons to FIELD and CONTAINER labels within CLUSTER elements. This overrides the value of the APPEND_COLON element in the curam-config.xml file for that individual page (see Section 3.12.13.8, APPEND_COLON).</p>
WINDOW_OPTIONS	No	"width=600,height=auto-calculated"	<p>The size of the page when displayed in a modal dialog is configurable using this parameter. The value of the attribute is a comma separated list of name value pairs. The currently supported options are width and height, both of which take an integer value, which is translated directly to a pixel value. Only a width needs to be specified however as the height will be dynamically calculated. Any other parameters will cause an exception to be thrown.</p>
TYPE	No	DEFAULT	<p>Used to define specific types of UIM pages. Two types are supported, DETAILS and SPLIT_WINDOW. SPLIT_WINDOW enables the use</p>

Attribute Name	Required	Default	Description
			<p>of frames within the page. If the attribute is not present or is set to <code>DEFAULT</code> then frames are not used. See Section 8.22, <i>Agenda Player</i> for an example of use.</p> <p><code>DETAILS</code> defines a UIM page that will be used as a context panel page. For more information see Section 6.8.3, <i>Context Panel UIM</i>.</p>
<code>HIDE_CONDITIONAL_LINKS</code>	No	TRUE	<p>Set to <code>true</code> to hide conditional links that evaluate to false. Set to <code>false</code> to show a disabled conditional link that evaluate to false. This overrides the value of the <code>HIDE_CONDITIONAL_LINKS</code> element in the <code>curam-config.xml</code> file for that individual page (see Section 3.12.13.8, <i>APPEND_COLON</i>).</p>

Table 5.34 Attributes of the PAGE Element

Child Elements

The PAGE element can contain child elements as follows:

Element Name	Cardinality / Description
<code>INCLUDE</code>	0..1. This element can be used before any other child element of a PAGE element.
<code>PAGE_TITLE</code>	0..1
<code>DESCRIPTION</code>	0..1
<code>SHORTCUT_TITLE</code>	0..1
<code>SERVER_INTERFACE</code>	0..n. Multiple <code>SERVER_INTERFACE</code> elements are supported, however it is recommended that only one <code>SERVER_INTERFACE</code> with the <code>PHASE</code> attribute set to <code>ACTION</code> is defined per <code>PAGE</code> element. See Section 5.9.29, <i>SERVER_INTERFACE</i> for more information.
<code>INFORMATIONAL</code>	0..1
<code>MENU</code>	0..2. The page can contain one optional static and one optional dynamic menu as well as append extra items to the navigation menu.

Element Name	Cardinality / Description
ACTION_SET	0..1. In this context, the action set defines the set of action controls that will appear around the page's main content area.
PAGE_PARAMETER	0..n
CONNECT	0..n. In this context, the connections can copy values directly from the properties of source server interfaces to properties of the target server interfaces. Each CONNECT element should contain both a SOURCE and a TARGET element.
JSP_SCRIPTLET	0..n. JSP_SCRIPTLET, CLUSTER and LIST can be intermingled freely and the order in UIM will be preserved in the generated page.
CLUSTER	0..n. JSP_SCRIPTLET, CLUSTER and LIST can be intermingled freely and the order in UIM will be preserved in the generated page.
LIST	0..n. JSP_SCRIPTLET, CLUSTER and LIST can be intermingled freely and the order in UIM will be preserved in the generated page.
SCRIPT	0..n. A script associated with the PAGE that will be activated in response to the specified event. See Section 5.9.28, <i>SCRIPT</i> for more details.

Table 5.35 Child Elements of the PAGE Element

Where a page is configured to contain a large number of scrollable list and cluster elements (approximately 15), it may cause JSP compile issues in Weblogic. This is due to a Weblogic system limitation in how big a page can be rendered at run time. To overcome this restriction, arrange the display of the required scrollable lists and clusters over a number of pages.

5.9.26 PAGE_PARAMETER

The PAGE_PARAMETER element declares a parameter to the current page. Once a parameter is declared, it can be used as the source of a connection by setting the connection source bean NAME attribute to PAGE.

Attributes

The PAGE_PARAMETER element has the following attributes:

Attribute Name	Required	Default	Description
NAME	Yes		The name of the parameter to use in SOURCE connection end-points.

Table 5.36 Attributes of the PAGE_PARAMETER Element

Child Elements

The PAGE_PARAMETER element contains no child elements.

5.9.27 PAGE_TITLE

The PAGE_TITLE element defines the title that appears at the top of a page's main content area. A title is constructed by concatenating a number of connection sources together. These can include localized strings and data from server interfaces.



Note

The PAGE_TITLE element defines the text for the tab title bar where the UIM page is used as a context panel page. See Section 6.8.3, *Context Panel UIM* for more information.

Attributes

The PAGE_TITLE element has the following attributes:

Attribute Name	Required	Default	Description
DESCRIPTION	No		A reference to a localized string that provides a more detailed description of the page than the title alone. This will be displayed with the title in the page's main content area.
STYLE	No		The name of the CSS class to use when displaying the title on the page.
ICON	No		A reference to an entry in the <code>Image.properties</code> file specifying the image file to use beside the title in the main content area.

Table 5.37 Attributes of the PAGE_TITLE Element

Child Elements

The PAGE_TITLE element can contain child elements as follows:

Element Name	Cardinality / Description
CONNECT	1..n. Only CONNECT elements containing SOURCE elements can be included (one SOURCE per CONNECT). Sources can be server interface properties or, with the NAME attribute set to TEXT, references to strings from a properties file.
DESCRIPTION	0..1 The DESCRIPTION element has the same behavior as the DESCRIPTION attribute but allows the description to be built up from a number of sources. If both are specified, this element takes precedence over the corresponding attribute.

Table 5.38 Child Elements of the PAGE_TITLE Element

5.9.28 SCRIPT

The SCRIPT element defines an exit point to allow the invocation of a script (JavaScript) in response to an event. Scripts are supported for pages, read-write fields and action controls. These elements are not applicable and not supported for fields within a LIST or read-only fields.

Attributes

The SCRIPT element has the following attributes:

Attribute Name	Required	Default	Description
EVENT	Yes		<p>The JavaScript name of the event as defined in the W3C HTML recommendations.</p> <p>JavaScript events are valid within the PAGE, FIELD or ACTION_CONTROL elements, with the exception of FIELD elements within a LIST or read-only FIELD elements.</p> <p>Note that the ONCLICK event will be ignored for ACTION_CONTROL with a TYPE of CLIPBOARD (for further information see Section 5.9.3, ACTION_CONTROL.).</p> <p>In addition, please note that by default when a link is clicked in the Cúram application the link is pro-</p>

Attribute Name	Required	Default	Description
			cessed by Cúram specific code. If you are adding some scripting to a link and do not want this default processing to occur, the event should be stopped using the JavaScript APIs available.
ACTION	Yes		The JavaScript to be invoked if the event occurs. This must be a function call including parameters, if any. For example; <code>someFunction()</code> or <code>someFunction(someParam)</code> where <i>someParam</i> may be a global variable defined in script file.
SCRIPT_FILE	No		The name of the script file containing the JavaScript functions that are specified in the ACTION attribute of the SCRIPT element. If no SCRIPT_FILE attribute is set on a particular SCRIPT element within a FIELD or ACTION_CONTROL the PAGE script file is used by default. The script file should be added in a component. If another script file has the same name in another component, the version in the highest priority component will be used. If not specified, the SCRIPT will expect to find the functions in the page-level script file specified with the PAGE element's SCRIPT_FILE attribute.

Table 5.39 Attributes of the SCRIPT Element

Child Elements

The SCRIPT element contains no child elements.

5.9.29 SERVER_INTERFACE

The SERVER_INTERFACE element defines a server interface to which other elements of the page can connect.

Attributes

The `SERVER_INTERFACE` element has the following attributes:

Attribute Name	Required	Default	Description
<code>NAME</code>	Yes		A unique name for this instance of the server interface on this page.
<code>CLASS</code>	Yes		The name of the server interface class.
<code>OPERATION</code>	Yes		The name of the server interface operation on the class.
<code>PHASE</code>	No	<code>DISPLAY</code>	<p>The phase of the page in which the server interface is called. This can be <code>DISPLAY</code> (the default) or <code>ACTION</code>. Server interfaces set to the <code>DISPLAY</code> phase are called as the page is displayed (i.e., the execution of the JSP page).</p> <p>Server interfaces set to the <code>ACTION</code> phase are only called in response to the activation of an <code>ACTION_CONTROL</code> with a <code>TYPE</code> of <code>SUBMIT</code>. It is recommended that only one <code>SERVER_INTERFACE</code> is set to the <code>ACTION</code> phase per <code>PAGE</code>.</p>
<code>AC-TION_ID_PROPERTY</code>	No		<p>Specifies a name of the server access bean property that will be populated with <code>ACTION_ID</code> of the action control used to make the server call. The value of of this attribute must be a valid property name of the corresponding server access bean. The use of shorthand notation is allowed (for example specify <code>theProperty</code> instead of the fully qualified <code>dtls\$theProperty</code>).</p> <p>This attribute is only valid on server interfaces with <code>PHASE= ACTION</code> and must be specified on all server interfaces within the page or not specified on any of them.</p> <p>If multiple server interfaces spe-</p>

Attribute Name	Required	Default	Description
			<p>cify ACTION_ID_PROPERTY with different domains the value of ACTION_ID on all action controls within the page must be suitable for all of the domains. Failing to comply with this rule will lead to error at runtime when the corresponding action control is activated.</p> <p>If this attribute is specified then the ACTION_ID attribute of ACTION_CONTROL element must also be specified.</p>

Table 5.40 Attributes of the SERVER_INTERFACE Element

**Note**

It is technically possible to specify multiple SERVER_INTERFACE elements set to the ACTION phase. However, this is not recommended. Each SERVER_INTERFACE is essentially a separate transaction and when an invocation fails, no further invocations of other server interfaces are made and completed transactions are not rolled back.

For example, three SERVER_INTERFACE elements are defined, each set to the ACTION phase. When the page is executed, the first server interface invocation succeeds and the second fails. In this scenario, the third server interface is never invoked and the action of the first will not be rolled back.

Child Elements

The SERVER_INTERFACE element contains no child elements.

5.9.30 SOURCE

The SOURCE element defines the source end-point of a data connection. The source can be the value of a server interface property, the value of a parameter to the page (which must be declared via the PAGE_PARAMETER element), or the value of an externalized string.

Attributes

The SOURCE element has the following attributes:

Attribute Name	Re-quired	Default	Description
NAME	Yes		The name of the SERV-ER_INTERFACE instance to use as the source of the property value, or PAGE, if the source is the value of a page parameter, or TEXT (or CONSTANT) if the source is the value of an externalized text string. TEXT or CONSTANT can only be used when TARGET has a server interface defined in the ACTION phase.
PROPERTY	Yes		The name of the server interface property, the name of the input page parameter, or the string reference to the externalized string whose value is required.

Table 5.41 Attributes of the SOURCE Element

Child Elements

The SOURCE element contains no child elements.

5.9.31 TAB_NAME

The TAB_NAME element defines the text used for the tab in the tab bar, where the UIM page is used as a context panel UIM page. The text is constructed by concatenating a number of connection sources together. These can include localized strings and data from server interfaces.

This element only applies where the TYPE attribute of the PAGE element is set to DETAILS. See Section 6.8.3, *Context Panel UIM* for more information.

Child Elements

The TAB_NAME element can contain child elements as follows:

Element Name	Cardinality / Description
CONNECT	1..n. Only CONNECT elements containing SOURCE elements can be included (one SOURCE per CONNECT). Sources can be server interface properties or, with the NAME attribute set to TEXT, references to strings from a properties file.
DESCRIPTION	0..1 The DESCRIPTION element has the

Element Name	Cardinality / Description
	same behavior as the DESCRIPTION attribute but allows the description to be built up from a number of sources. If both are specified, this element takes precedence over the corresponding attribute.

Table 5.42 Child Elements of the TAB_NAME Element

5.9.32 TARGET

The TARGET element defines the target end-point of a data connection. The target can be the value of a server interface property or the value of a parameter to be exported from the page.

Attributes

The TARGET element has the following attributes:

Attribute Name	Required	Default	Description
NAME	Yes		The name of the SERVER_INTERFACE instance to use as the target of the property value, or PAGE, if the target is the value of a page parameter.
PROPERTY	Yes		The name of the server interface property, or the name of the output page parameter whose value is to be set.

Table 5.43 Attributes of the TARGET Element

Child Elements

The TARGET element contains no child elements.

5.9.33 TITLE

The TITLE element defines the title that appears at the top of a CLUSTER or LIST element. A TITLE is constructed by concatenating a number of connection sources together. These can include localized strings and data from server interfaces.

Attributes

The TITLE element has the following attributes:

Attribute Name	Required	Description
SEPARATOR	No	A reference to an externalized string to use as the separator between the elements within the container.

Table 5.44 Attributes of the TITLE Element

Child Elements

The TITLE element can contain child elements as follows:

Element Name	Cardinality / Description
CONNECT	1..n. Only CONNECT elements containing SOURCE elements can be included (one SOURCE per CONNECT). Sources can be server interface properties or, with the NAME attribute set to TEXT, references to strings in a properties file.

Table 5.45 Child Elements of the TITLE Element

5.9.34 VIEW

The VIEW element is the root element of a UIM document that defines elements to be included in a UIM page document. A view cannot include other views using the INCLUDE element.

Attributes

The VIEW element has no attributes.

Child Elements

The VIEW element can contain child elements as follows:

Element Name	Cardinality / Description
PAGE_TITLE	See the PAGE element.
SHORTCUT_TITLE	See the PAGE element.
SERVER_INTERFACE	See the PAGE element.
MENU	See the PAGE element.
ACTION_SET	See the PAGE element.
PAGE_PARAMETER	See the PAGE element.
CONNECT	See the PAGE element.

Element Name	Cardinality / Description
JSP_SCRIPTLET	See the PAGE element.
CLUSTER	See the PAGE element.
LIST	See the PAGE element.
SCRIPT	See the PAGE element.

Table 5.46 Child Elements of the VIEW Element

5.10 UIM Reference for Widgets

5.10.1 Introduction

Widgets are used when the handling of data in the client application is too complicated to do with the automatic domain definition recognition of the FIELD element. Widgets allow several different sources of data to be connected to a control that can then supply data to several different targets.

There are a number of predefined types of WIDGET element. Each type of WIDGET can contain one or more WIDGET_PARAMETER elements. The configuration of these WIDGET_PARAMETER elements depends on the type of the widget. These are described in the sections below.

Most widget types can only be defined within CLUSTER elements (exceptions to this are described below). There may also be restrictions on how many widgets of a particular type can be included in a single UIM document.

5.10.2 WIDGET

The WIDGET element is used to define the type of widget to include and it holds the WIDGET_PARAMETER elements that configure the widget.

Attributes

The WIDGET element has the following attributes:

Attribute Name	Required	Default	Description
TYPE	Yes		<p>The type of WIDGET. This can be one of the following:</p> <ul style="list-style-type: none"> EVIDENCE_COMPARE FILE_EDIT FILE_UPLOAD MULTISELECT

Attribute Name	Required	Default	Description
			<ul style="list-style-type: none"> • SINGLESELECT • RULES_SIMULATION_EDITOR • FILE_DOWNLOAD • IEG_PLAYER
LABEL	No		A reference to an externalized string that should be used as the associated label string for this widget.
WIDTH	No		The width of the control specified in the appropriate units.
WIDTH_UNITS	No	PERCENT	The units in which the width is interpreted. This can be PERCENT to indicate the percentage of the space available to the widget, or CHARS to indicate the number of visible characters wide the widget will be.
HEIGHT	No	1	A HEIGHT value that may be used by the widget.
ALIGNMENT	No	DEFAULT	Defines the horizontal alignment of the widget. Can be set to LEFT, RIGHT, CENTER, or DEFAULT. The value DEFAULT corresponds to the CSS class default in curam_common.css. Currently the default is to be left aligned.
HAS_CONFIRM_PAGE	No	false	Attribute to be used only on widget of type of MULTISELECT. Used to specify that the widget selection data is to be submitted to the confirmation page. Can be true or false. See Section 5.10.8.1, <i>Confirmation Pages</i> .

Table 5.47 Attributes of the WIDGET Element

Child Elements

The WIDGET element can contain the following child element:

Element Name	Cardinality / Description
WIDGET_PARAMETER	1..n. The parameters depend on the type of widget.

Table 5.48 Child Elements of the WIDGET Element

5.10.3 WIDGET_PARAMETER

The WIDGET_PARAMETER element is used to define the properties of an individual widget. In particular, the WIDGET_PARAMETER elements allow connections to be made between named properties of the widget and various source and target data end-points.

Attributes

The WIDGET_PARAMETER element has the following attribute:

Attribute Name	Required	Default	Description
NAME	Yes		The name of the property on the WIDGET that this element configures.

Table 5.49 Attributes of the WIDGET_PARAMETER Element

Child Elements

The WIDGET_PARAMETER element can contain the following child element:

Element Name	Cardinality / Description
CONNECT	<p>A WIDGET_PARAMETER can be connected in one of two ways depending on the specification for the particular WIDGET. The first way is similar to that of FIELD elements:</p> <p>1..n. The parameter can contain multiple CONNECT elements. Usually (the FILE_DOWNLOAD WIDGET is an exception to this) a WIDGET_PARAMETER contains up to three CONNECT elements, SOURCE, TARGET, and INITIAL connection end-points. The valid types of source or target depend on the individual parameter.</p> <p>The second way to connect a parameter is similar to the CONNECT elements in a LINK element.</p>

Element Name	Cardinality / Description
	1..n. CONNECT elements that each connect a SOURCE end-point to a TARGET end-point.

Table 5.50 Child Elements of the WIDGET_PARAMETER Element

5.10.4 The EVIDENCE_COMPARE Widget

The EVIDENCE_COMPARE widget displays the differences between two sets of evidence. These differences are high-lighted using the following colors: evidence items that have changed are shown in red; new items are shown in green; deleted items are shown in gray.

This widget should be the sole element in a CLUSTER. Its TYPE should be set to EVIDENCE_COMPARE and its WIDGET_PARAMETER elements should be set as follows:

Parameter Name	Required	Description and Connections
OLD_EVIDENCE	Yes	This parameter must include a single CONNECT element that must specify a SOURCE end-point. The SOURCE end-point should specify a property of the EVIDENCE_TEXT domain that contains the original evidence.
NEW_EVIDENCE	Yes	This parameter must include a single CONNECT element that must specify a SOURCE end-point. The SOURCE end-point should specify a property of the EVIDENCE_TEXT domain that contains the new evidence.

Table 5.51 Parameters to the EVIDENCE_COMPARE Widget

5.10.5 The FILE_EDIT Widget

The FILE_EDIT widget allows a user to edit a Microsoft *Word*® document on their local computer and then save it to the server. A document can be created automatically from a template where the template details can be

set before the document is presented to the user for editing.

A UIM page containing the `FILE_EDIT` widget will only operate in the main content panel of the application. If such page is opened in a modal window then the modal will close immediately and the page will be loaded in the main content panel.

The widget uses a Java applet to manage the interaction between the user's browser and *Word*. Only the source and target documents and the template details are required. If key details, or other data, are required by the server interfaces that handle the document, these should be provided by page parameters and page-level connections.

Each time the document is saved in *Word*, the submit button for the page is activated automatically. This triggers the `ACTION` phase but returns to the same page rather than opening the page linked to by the submit button. Only when the *Word* document is closed will the next page be opened. This behavior requires that the server interface for the `ACTION` phase allows multiple invocations for the same editing session and that it saves the document to the database on each invocation.

The first time the *Word* document is loaded successfully with the template details, it is automatically saved to the server before further editing.

When editing the document, the user has the option to save it. This triggers the normal saving behavior and the page will not be changed when the `ACTION` phase completes. After the document has been closed, the `ACTION` phase will be triggered again to open the next page, but this time the server interface will not be invoked and the document (which has already been saved) will not be saved again. Because the server interface is not invoked, it is not permitted to use any property of the `ACTION` phase server interface in a `SOURCE` connection of the submit button's `LINK` element. Typically, the submit button will return to the previous page and will not need a `LINK` element, so this limitation should have little impact.

Using the `FILE_EDIT` widget is simple. The `WIDGET` element should have the `TYPE` attribute set to `FILE_EDIT`. Two `WIDGET_PARAMETER` elements are required:

Parameter Name	Required	Description and Connections
<code>DOCUMENT</code>	Yes	Defines the source document (usually a template) and the target to which to write the saved document. The parameter must contain a <code>CONNECT</code> element with a <code>SOURCE</code> set from a <code>DISPLAY</code> phase sever interface and a <code>TARGET</code> set from an <code>ACTION</code> phase sever interface. Both fields should be

Parameter Name	Required	Description and Connections
		<p><i>Word</i> documents.</p> <p>The data-type for both the source and target document must be SVR_BLOB.</p>
DETAILS	Yes	<p>The template details that should be set in the document before presenting it to the user for editing. The parameter must contain a CONNECT element with a SOURCE set from a DISPLAY phase sever interface. The details are in XML format, described below.</p> <p>The data-type for the template details must be SVR_BLOB.</p>

Table 5.52 Parameters to the FILE_EDIT Widget

The template details must be provided in a simple XML format. An example of the format is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<FIELDS>
  <FIELD NAME="personName" VALUE="John Smith"/>
  <FIELD NAME="AddressLine1" VALUE="1 Main Street"/>
  <FIELD NAME="AddressLine2" VALUE="Newtown"/>
  <FIELD NAME="AddressLine3" VALUE="Erehwon"/>
</FIELDS>
```

Example 5.14 Sample Template Details

It is recommended that your XML uses UTF-8 encoding to handle multi-byte characters. To preserve the correct encoding it is important that any code that manipulates the XML honors the encoding of the document. If the encoding is not honored, this can lead to characters being displayed incorrectly when opened in *Microsoft®Word*.

Each FIELD element identifies the name of a field in the document template and the value to which it should be set.

While editing the document in *Word*, navigation within the originating browser window is disabled. An alert message will be displayed if any attempt is made to navigate from the page. If the originating browser window is closed, the *Word* document will stay open, but the editing session will be terminated. Any unsaved changes will not be persisted in database.

User Machine Configuration

On first use of a new version of the integration applet the user will be presented with a popup dialog window to confirm if the code from publisher "IBM Corporation" should be allowed to run. The checkbox "Always trust the content from this publisher" should be selected and dialog confirmed, which will ensure the widget executes successfully and the prompt is not displayed again on subsequent uses. New versions of the widget will be downloaded to the user's machine automatically when the Cúram application is upgraded to a new version.

When a user attempts to edit a *Word* document, execution of the integration applet may be blocked depending on security settings of the Java browser plugin on that particular machine. This causes the editing session to fail. If you experience this kind of issues issues, please check the following:

- *Microsoft Word* 2002 or higher should be installed on the user's machine.
- Word installation should be working as expected on the user's machine when started manually.
- The Web browser Popup blocker feature on the user's machine should be disabled.
- For supported browsers other than Internet Explorer if you are getting a message about the missing Java plugin even though it is installed on the machine, verify the following option is enabled: Control Panel -> Java -> Advanced -> Default Java for browsers-> Mozilla family
- Generally if you are getting message about the missing Java plugin even though it is installed on the machine, check if a slide-down message is displayed in the small popup window that opens when you attempt to edit a Word document. If so, then confirm that you want to always run code from this publisher and reload the application in the browser.

5.10.6 The FILE_UPLOAD Widget

The FILE_UPLOAD widget is a type of widget used to allow a user to specify a file on their local computer to be uploaded to the server. It will appear as a text field with a *Browse...* ² button beside it. The user can click on the button to open a file dialog box with which they can select their file.

The normal widget attributes WIDTH and WIDTH_UNITS are not applicable for the FILE_UPLOAD widget. Some browsers do not allow width of the filename entry box to be set for security reasons (it could be set to zero width and thus be hidden while remaining active).



File Size Validation

There are settings to limit the maximum size of a file that is allowed to be uploaded. The validations for these settings are carried out on the server side after the file is fully uploaded to a temporary directory. Therefore, it should be kept in mind that large files could be up-

loaded consuming a large amount of disk space. We recommend checking the file upload folder at intervals to ensure disk space usage meets requirements.

There are three application-level configuration settings for the `FILE_UPLOAD` widget. These control how the web-server handles the incoming files. Default settings are already present, but the default values can be overridden by adding configuration settings to the `ApplicationConfiguration.properties` file. The settings follow the same `name = value` format of all the other entries there. The settings are as follows:

uploadMaximumSize

This is the maximum size of a file that can be uploaded to the server. The number is specified in bytes. If the number is negative, there is no limit to the file size. By default, the value is `-1` (no limit).

uploadThresholdSize

This is maximum number of bytes of the file's content that the web-server will hold in memory while the file is being uploaded. Once the number of bytes uploaded exceeds this limit, the web-server will begin to store the file on disk to save memory. By default, the value is `1024`.

uploadRepositoryPath

This is the path to the folder on the disk in which the files will be stored as they are uploaded if they exceed the threshold size. By default, the value is the JVM defined temp folder, so this folder must be present on your system. If it is not on your system, you can create it or explicitly set the `uploadRepositoryPath` to a folder of your choice.

The `WIDGET` element should have the `TYPE` attribute set to `FILE_UPLOAD`. The widget supports the following `WIDGET_PARAMETER` elements:

Parameter Name	Required	Description and Connections
<code>CONTENT</code>	Yes	<p>This parameter indicates the target connection for the actual content of the uploaded file.</p> <p>A single <code>CONNECT</code> element with a <code>TARGET</code> that connects to a property of an <code>ACTION</code> phase server interface is required.</p>
<code>FILE_NAME</code>	No	<p>This parameter represents the name of the file to be uploaded. The parameter can be set to provide a default name for the file to be uploaded, and can also supply the name of the file chosen by the user.</p> <p>If present, the parameter can include <code>CONNECT</code> elements for either or both</p>

Parameter Name	Required	Description and Connections
		<p>end-points: a SOURCE end-point for the initial name of the file, and a TARGET end-point for the file that was actually chosen. The SOURCE end-point can specify a property of a DISPLAY phase server interface. The TARGET end-point can specify a property of an ACTION phase server interface.</p> <p><i>Note: Many browsers do not allow a default value for the name of a file to be uploaded. In this case, setting a SOURCE connection will have no effect.</i></p>
CONTENT_TYPE	No	<p>This parameter indicates the target connection for the content type of the uploaded file. The content type describes the format of the uploaded data. For example, a simple text file would have a content type of “text/plain” and a <i>Microsoft Word</i> document would have a content type of “application/msword”.</p> <p>A single CONNECT element with a TARGET that connects to a property of an ACTION phase server interface is required.</p>
ACCEPT- ABLE_CONTENT_TY- PES	No	<p>A HTML page only allows certain types of content to be uploaded by default (the actual default types are dependent on the browser). This parameter can specify the types of content that the page will accept. The value of the parameter should be a comma-separated list of content types. If there is more than one FILE_UPLOAD widget on a page, the acceptable content types of all widgets are pooled together and define what is acceptable for that page (this is a limitation of the HTML specification.)</p> <p>A single CONNECT element with a SOURCE that connects to a TEXT property is allowed.</p>

Table 5.53 Parameters to the FILE_UPLOAD Widget

5.10.7 The FILE_DOWNLOAD Widget

A WIDGET with the TYPE set to FILE_DOWNLOAD results in the generation of a hyperlink on the page. Clicking on the hyperlink invokes a special FileDownload servlet included in the Cúram CDEJ that returns the contents of a file from the database. The FileDownload servlet is configured with the server interface to call to get the file contents and the parameters to pass to identify that file. The configuration is performed in the curam-config.xml file. An example configuration is shown in Section 5.9.3.1, *File Downloads*. A single server interface can be configured for each page of the application that includes a file download widget. An example configuration is shown in Example 5.2, *Example Configuration for File Download*.

An ACTION_CONTROL with the TYPE set to FILE_DOWNLOAD can also be used to generate a hyperlink to download a file. You should use the ACTION_CONTROL element when the hyperlink text is a fixed value retrieved from the page's corresponding properties file. The FILE_DOWNLOAD WIDGET allows the hyperlink text to be a dynamic value retrieved from a server interface property.

The FILE_DOWNLOAD widget can also be utilized within the Actions menu of the Context Panel. The menu item TYPE must be set to FILE_DOWNLOAD. The menu item PAGE-ID must match the PAGE_ID attribute of the FILE_DOWNLOAD widget configuration. The file identifier must be available as a page parameter in the respective .tab file for the menu. This page parameter must match the PAGE_PARAM attribute of the FILE_DOWNLOAD widget configuration.

The WIDGET element should have the TYPE attribute set to FILE_DOWNLOAD. The widget supports the following WIDGET_PARAMETER elements:

Parameter Name	Required	Description and Connections
LINK_TEXT	Yes	This parameter indicates the source connection for sourcing content of the link text which will appear on the screen. A single CONNECT element with a SOURCE that connects to a property of a DISPLAY phase server interface is required. If you want to use a fixed text value, you should use an ACTION_CONTROL with the TYPE set to FILE_DOWNLOAD instead of a WIDGET.
PARAMS	No	This optional parameter supplies the

Parameter Name	Required Description and Connections
	<p>FileDownload servlet with the necessary parameters.</p> <p>The parameter can include CONNECT elements with a SOURCE end-point for the page parameter supplying a value for the FileDownload servlet, and a TARGET end-point for specifying the servlet parameter to supply the value to. The SOURCE end-point should refer to a parameter on the page declared by a corresponding PAGE_PARAMETER element. The TARGET end-point can specify a parameter whose name corresponds to a configured FileDownload servlet parameter name. Thus both end-points should have a NAME attribute set to PAGE.</p>

Table 5.54 Parameters to the FILE_DOWNLOAD Widget

5.10.8 The MULTISELECT Widget

The MULTISELECT widget allows you to specify that the first column in a LIST should contain a check-box on each row and to allow several rows to be selected. A “Select All” feature can be enabled which displays a check-box in the column header. See Section 3.12.13.14, *ENABLE_SELECT_ALL_CHECKBOX* for further details.

Each check box can represent multiple entities in the row. For each check box that is selected, the fields on that row will be compiled into a “|” delimited string and each row will be tab delimited and passed as a page parameter when a specific type of page link is activated.

The UIM document in Example 5.15, *MULTISELECT Example* is an example of a page with multiple rows with check boxes. When the form is submitted, a single string, containing multiple fields for each selected row, is passed to the `in$tabbedString` field on the target page. Following the UIM is a detailed description of each relevant part of the UIM that implement this functionality.

```
<PAGE PAGE_ID="MultiSelectWidgetTest"
      xsi:noNamespaceSchemaLocation="CuramUIMSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SERVER_INTERFACE NAME="DISPLAY" CLASS="MyBean"
                    OPERATION="Display" PHASE="DISPLAY"/>
  <SERVER_INTERFACE NAME="ACTION" CLASS="MyBean"
                    OPERATION="Submit" PHASE="ACTION"/>
</PAGE>
```

```

<LIST TITLE="List.Title">
  <ACTION_SET BOTTOM="false">
    <ACTION_CONTROL TYPE="SUBMIT">
      <LINK PAGE_ID="MultiSelectWidgetResult">
        <CONNECT>
          <SOURCE NAME="ACTION"
                PROPERTY="in$tabbedString"/>
          <TARGET NAME="PAGE"
                PROPERTY="referenceNumTabString"/>
        </CONNECT>
      </LINK>
    </ACTION_CONTROL>
  </ACTION_SET>
  <CONTAINER LABEL="List.Multiselect.Header" WIDTH="5"
            ALIGNMENT="CENTER">
    <WIDGET TYPE="MULTISELECT"
            HAS_CONFIRM_PAGE="true">
      <WIDGET_PARAMETER NAME="MULTI_SELECT_SOURCE">
        <CONNECT>
          <SOURCE PROPERTY="personID" NAME="DISPLAY"/>
        </CONNECT>
        <CONNECT>
          <SOURCE PROPERTY="caseID" NAME="DISPLAY"/>
        </CONNECT>
      </WIDGET_PARAMETER>
      <WIDGET_PARAMETER NAME="MULTI_SELECT_TARGET">
        <CONNECT>
          <TARGET PROPERTY="in$tabbedString" NAME="ACTION"/>
        </CONNECT>
      </WIDGET_PARAMETER>
      <WIDGET_PARAMETER NAME="MULTI_SELECT_INITIAL">
        <CONNECT>
          <SOURCE NAME="DISPLAY" PROPERTY="out$tabString"/>
        </CONNECT>
      </WIDGET_PARAMETER>
    </WIDGET>
  </CONTAINER>
  <FIELD LABEL="Field.Title.ReferenceNumber" WIDTH="35">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="personID"/>
    </CONNECT>
  </FIELD>
  <FIELD LABEL="Field.Title.Forename" WIDTH="30">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="firstName"/>
    </CONNECT>
  </FIELD>
  <FIELD LABEL="Field.Title.Surname" WIDTH="30">
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="surname"/>
    </CONNECT>
  </FIELD>
</LIST>
</PAGE>

```

Example 5.15 MULTISELECT Example

The main points to note in the above UIM example are:

- The WIDGET of TYPE equal to MULTISELECT is a child node of a CONTAINER element. The container's label will be used as the column header unless the select all check box is enabled in curam-config.xml. See Section 3.12.13.14, *ENABLE_SELECT_ALL_CHECKBOX* for further details.
- Up to three WIDGET_PARAMETER elements are allowed within the WIDGET element. MULTI_SELECT_SOURCE and

MULTI_SELECT_TARGET are mandatory and MULTI_SELECT_INITIAL is optional.

- The MULTI_SELECT_SOURCE can have multiple CONNECT elements, each with one SOURCE element. Each SOURCE is added to the “|” delimited string. If only one SOURCE element is specified the string will not contain any “|” delimiters. Then each select row will be delimited by a tab character.
- The MULTI_SELECT_TARGET element must contain only one CONNECT element with only one TARGET element. This TARGET element specifies the field on the action phase bean that the “|” and tab-delimited string will be assigned to when the page is submitted.
- The MULTI_SELECT_INITIAL contains only one CONNECT element with a single SOURCE element. This contains a “|” and tab-delimited string which specifies the rows that are selected when the page is loaded.
- In the LIST element the ACTION_SET has one ACTION_CONTROL element.
- Optional HAS_CONFIRM_PAGE attribute is used to indicate that the page with MULTISELECT widget submits to a confirmation page, where user selection is re-displayed for confirmation. See Section 5.10.8.1, *Confirmation Pages*

Below is an example of the delimited string passed as a parameter to the specified page.

101|case121 102|case122 103|case123

Parameter Name	Required	Description and Connections
MULTI_SELECT_SOURCE	Yes	This parameter can include multiple CONNECT elements that must specify a SOURCE end-point. The SOURCE end-point must be a list property containing the key data for the row.
MULTI_SELECT_TARGET	Yes	This parameter must include one CONNECT element that must specify a TARGET end-point. The TARGET end-point must be a string property containing the key data for selected rows.
MULTI_SELECT_INITIAL	No	This parameter must include

Parameter Name	Required Description and Connections
	<p>one CONNECT element that must specify a SOURCE end-point.</p> <p>The SOURCE end-point must be a string property containing the key data for the rows that are initially check when page is loaded.</p>

Table 5.55 Parameters to the MULTISELECT Widget

Confirmation Pages

MULTISELECT widget has a specific mechanism allowing for confirming user selection on a separate page. This confirmation page is supposed to re-display values selected by an user on the MULTISELECT widget offering a choice to review these values and confirm them or re-visit the previous page to refine the selection.

Confirming user selection can become a problem where there is a lot of selected values from a big MULTISELECT widget to be passed to the confirmation page. There are request length limitations in place, so in order to pass bigger amounts of data possible in this case different request mechanism (request forwarding) has to be used.

MULTISELECT widget with the selection to be confirmed is specified by HAS_CONFIRM_PAGE optional attribute on the WIDGET element. The attribute is to be set to true. It is only valid for a widget of TYPE of MULTISELECT.

Some things to keep in mind with confirmation pages:

- As request forwarding is used to carry the data in this case, the URL for the confirmation page will not be displayed with the forwarding page URL shown instead.
- Even though the mentioned attribute is set on a MULTISELECT widget, the setting applies to the whole page (as there is only one form per page). So, in case where multiple submit buttons exist on a page with MULTISELECT widget to be confirmed, a confirmation step should be assumed for all of these buttons (i.e., there is no way to have a submit with confirmation and another without confirmation on that page).
- The confirmation is to be the immediate step carried out on submitting the form with user selection; no resolve page should be used in the middle.
- It is recommended to have a read-only page for user selection confirma-

tion, allowing user to cancel and return to the previous page if the selection is to be refined.

5.10.9 The SINGLESELECT Widget

The SINGLESELECT widget allows you to specify that the first column in a LIST should contain a radio button on each row. This widget functions in same way as the MULTISELECT widget, except you are limited to selecting a single item via radio buttons instead of check boxes. See Section 5.10.8, *The MULTISELECT Widget* for further details.

Parameter Name	Required	Description and Connections
SELECT_SOURCE	Yes	<p>This parameter must include multiple CONNECT elements that must specify a SOURCE end-point.</p> <p>The SOURCE end-point must be a list property containing the key data for the rows to be displayed.</p>
SELECT_TARGET	Yes	<p>This parameter must include one CONNECT element that must specify a TARGET end-point.</p> <p>The TARGET end-point must be a string property containing the key data for selected row.</p>
SELECT_INITIAL	No	<p>This parameter must include one CONNECT element that must specify a SOURCE end-point.</p> <p>The SOURCE end-point must be a string property containing the key data for the row that is initially checked when page is loaded.</p>

Table 5.56 Parameters to the SINGLESELECT Widget

5.10.10 The RULES_SIMULATION_EDITOR Widget

The RULES_SIMULATION_EDITOR widget is used to edit or create data used when simulating the execution of a rule-set. The widget generates

clusters of fields that correspond to the fields of Rules Data Objects (RDO). A normal cluster is used to display the fields of a basic RDO and a multi-column cluster is used for a list RDO. A standard list is not used, as a list RDO with many fields would result in a list that had too many columns to be displayed on the screen.

The user can enter or modify values on the page corresponding to the RDO fields and, for list RDOs displayed in a multi-column cluster, press a button to create additional columns for field values.

The `WIDGET` element should have the `TYPE` attribute set to `RULES_SIMULATION_EDITOR`. The parameters to the widget are as follows:

Parameter Name	Required	Description and Connections
<code>VALUES</code>	Yes	<p>The simulation data values. A previous set of values can be displayed and edited or a new set of values can be created.</p> <p>The parameter should contain a <code>CONNECT</code> element with a <code>SOURCE</code> set to a <code>DISPLAY</code> phase bean field containing the values and a <code>TARGET</code> set to an <code>ACTION</code> phase bean field that will receive the edited values. If the <code>SOURCE</code> has no values set, the editor will create them.</p>
<code>META_DATA</code>	Yes	<p>The simulation meta-data. The meta-data contains details about the structure of the RDOs necessary to generated the input fields.</p> <p>The parameter should contain a <code>CONNECT</code> element with a <code>SOURCE</code> set to a <code>DISPLAY</code> phase bean field containing the meta-data.</p>
<code>ADD_BUTTON_CAPTION</code>	Yes	<p>The caption to use on the button displayed at the bottom of each multi-column cluster and used to add a new column of extra data to a list RDO. If an image is also specified, this caption is used as the “alt” text of the image.</p>

Parameter Name	Required	Description and Connections
		The parameter should contain a CONNECT element with a SOURCE that gets a localized string from a TEXT source.
ADD_BUTTON_IMAGE	No	<p>The path to the image file to use if an image button is to be used in place of a standard button. The path is relative to the WebContent folder.</p> <p>The parameter should contain a CONNECT element with a SOURCE that gets a localized string from a TEXT source.</p>

Table 5.57 Parameters to the RULES_SIMULATION_EDITOR Widget

The widget should be placed in a CLUSTER element. The clusters for the RDOs will be rendered within that cluster. The SHOW_LABELS attribute should be set to false. The LABEL_WIDTH attribute of the CLUSTER element will be inherited by the clusters that are generated by the widget, so it can be used to control the layout. An ACTION_CONTROL element in the cluster or on the page should be added to save and process the simulation data created by the widget in the usual manner.

When a widget is not supplied with any simulation data values, it will display empty fields. For list RDOs, a single empty column of fields will be displayed; values can be entered and more columns added as needed. If values are supplied, they will be displayed. In a multi-column cluster, pressing the defined “add” button will add a single empty column to the right of any existing columns. All other empty columns will be removed at this time, so deleting the values in one or more columns has the effect of removing those columns from the multi-column cluster.

5.10.11 The IEG_PLAYER Widget

Consult the *Cúram Intelligent Evidence Gathering (IEG)* guide for details.

5.11 Dynamic UIM Cross Reference

Dynamic UIM as its name implies, is UIM that is cached in the resource store - rather than static UIM (described in earlier sections) which resides on the file system - so that the server and client do not have to be rebuilt in order for a page to be displayed in an application. All string values in dynamic

UIM documents must be externalized in properties files, which must also be cached in the resource store.

When creating a dynamic UIM document, only the `PAGE` element is a valid root element. All the UIM features (elements and attributes) referenced in Section 5.9, *UIM Reference for Pages and Views* are supported for dynamic UIM, except for those which are listed in Appendix A, *Unsupported Dynamic UIM features*.

Refer to Appendix B, *Maintaining Dynamic UIM Pages* on details about how to maintain dynamic UIM pages in the Resource Store.

5.12 Dynamic UIM System Initialization

There are two ways in which the Dynamic UIM system can be initialized; when the application is started, or the first time that there is a request for a Dynamic UIM page in the running application. By default the Dynamic UIM system is initialized when the application is started. In order to override the default initialization of the Dynamic UIM system - so that it is initialized when a Dynamic UIM page is first requested - a configuration setting can be added to the `ApplicationConfiguration.properties` file. This settings follows the same *name = value* format of all the other entries there. It should be set as follows:

dynamicUIMInitModelOnStart

This value should be set to `false` in order to override the default setting.

If a developer intends to access dynamic UIM pages in the application, then the default initialization of the dynamic UIM system must be used. Otherwise, if the developer is not using dynamic UIM pages and finds their Tomcat start-up time is too slow, the default initialization of the dynamic UIM should be overridden, as described above.

Notes

¹The reserved characters in XML are “'”, “””, “&”, “<”, and “>”. The respective XML character entities are “'”, “"”, “&”, “<”, and “>”.

²The actual appearance of the button depends on the browser being used and may be different from this. The button is created by the browser and there is no control over its appearance.

Chapter 6

Application Configuration

6.1 Objective

This chapter provides you with all the information about application configuration files required to develop Cúram web client applications.

6.2 Prerequisites

You should be familiar with the basic concepts of Cúram CDEJ development, as outlined in Chapter 2, *Concepts*, in addition to the *Cúram User Experience Guidelines*. You should also have some knowledge of the basic format of XML documents.

In addition, the *Working with the Cúram User Interface* guide is a companion guide to this document and illustrates the usage of the features outlined in this chapter using concrete examples.

6.3 Introduction

An application in the Cúram user interface is a collection of user interface elements, predominantly based on UIM.¹ pages, combined to create specific content for a particular user or role. An application comprises of an application banner and one or more application sections. Each section, contains an optional section shortcut panel and one or more tabs. A tab represents a business object or logical grouping of information.

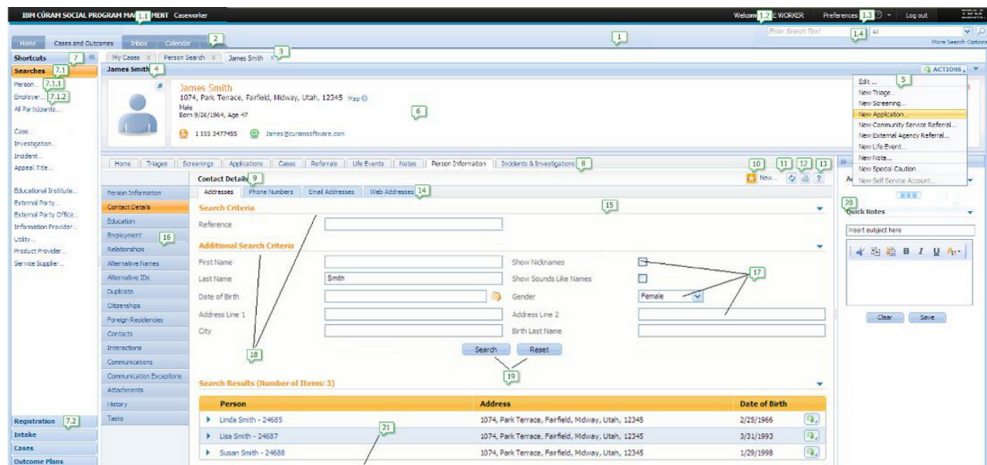


Figure 6.1 Application User Interface Overview

Figure 6.1, *Application User Interface Overview* illustrates a functional overview of the User Interface Elements within a sample application page.

The following sections of this chapter outline how to develop an application, using the relevant XML configuration files.

6.4 Configuration Files

Applications, sections, tabs and their relevant elements are defined using XML based configuration files. These files are located in the <server-dir>\components\<component-name>\clientapps directory. Section 3.12.12, *Application Configuration Files* should be consulted for more information on the clientapps directory, and best practices for working with application configuration files.

Each configuration file has a specific extension and an associated schema file detailing the supported attributes. A summary of the file extensions and related schema files is available in Table 6.1, *Configuration Files*.

File Extension	Schema File	Description
.app	application-view.xsd	Configuration file to define an application, including the application banner, referenced sections and application search.
.sec	section.xsd	Configuration file to define the referenced tabs and section shortcut panel in a section.
.ssp	section-shortcut-panel.xsd	Configuration file to define the contents of a section shortcut panel.

File Extension	Schema File	Description
.tab	tab.xsd	Configuration file to define a tab, including the context panel and referenced navigation and actions menu.
.nav	navigation.xsd	Configuration file to define the content of a tab navigation bar.
.mnu	menubar.xsd	Configuration file to define the content of a tab actions menu.

Table 6.1 Configuration Files

The schema files are all located in the `<sdej-dir>\lib` directory and can be used during development for validation in any XML editor.

The configuration files for applications, sections and tabs are processed as part of the **database** target and stored on the database for use at runtime. A standalone target, **inserttabconfiguration**, is also available for processing the configuration files only. This command is useful during development because it is more efficient than the full database target. For more information on these targets please consult the *Cúram Server Developers Guide*.

The **inserttabconfiguration** validates all the configuration files, ensuring that they conform to the XML schema, in addition to ensuring that all mandatory elements and attributes are specified. All files are processed before the build fails, listing all validation errors.

6.5 Applications

6.5.1 Introduction

An application is a particular view of the Cúram client defined for a specific user or role. The application definition file details the application banner and a reference to the sections that are part of the application.

An application banner provides the user with the context of the application they are currently accessing. The banner contains the following elements:

- The name of the application. Refer to User Interface Element 1.1 in Figure 6.1, *Application User Interface Overview* to see an example of an application name configured in the User Interface.
- The role of the user that this application is intended for.
- A welcome message for the user. Refer to User Interface Element 1.2 in Figure 6.1, *Application User Interface Overview* to see an example of

a welcome message configured in the User Interface.

- An application menu, which includes links to the User Preferences dialog, application help, the about box, and to logout of the application. Refer to User Interface Element 1 . 3 in Figure 6.1, *Application User Interface Overview* to see an example of an application menu configured in the User Interface.
- A quick search facility for the application. Refer to User Interface Element 1 . 4 in Figure 6.1, *Application User Interface Overview* to see an example of an application search configured in the User Interface.

The application search is an optional addition to the application banner which provides a quick search facility. The application search supports:

- A text entry field where the user can enter their search criteria.
- An optional search type combo box, which lists the types of object which can be searched on.
- A search button to trigger the actual search.
- An optional link to more search options.

Refer to User Interface Element 1 . 4 in Figure 6.1, *Application User Interface Overview* to see an example of a fully configured application search in the User Interface. This example has both the optional search type combo box, and optional link with more search options enabled

6.5.2 Definition

An application is defined by creating an XML file with the extension `.app` in the `clientapps` directory. The root XML element in the `.app` file is the `application` element and the attributes allowed on this element are defined in Table 6.2, *Attributes of the application Element*. The application banner is configured using these attributes.

Attribute	Description
id	<p><i>Mandatory.</i></p> <p>The unique identifier for the application, which must match the name of the file. This id matches to an APPLICATION_CODE entry and is used to determine the application to display for a particular user.</p> <p>See Section 6.5.5, <i>Associate an Application with User</i> for more information.</p>
title	<p><i>Optional.</i></p> <p>The text for the title that will be displayed as part of the application banner. The attribute must reference an entry in the associated properties file.</p>

Attribute	Description
sub-title	<i>Optional.</i> The text for the subtitle that will be displayed as part of the application banner. The attribute must reference an entry in the associated properties file.
user-message	<i>Optional.</i> The text for the welcome message that will be displayed as part of the application banner. The attribute must reference an entry in the associated properties file. The text can contain a placeholder, <code>%user-full-name</code> , which will be replaced with the users full name. The full name is determined based on the <code>FirstName</code> and <code>Surname</code> fields on the <code>Users</code> database table.
hide-tab-container	<i>Optional.</i> When set to true, this indicates that there is only one section in the application and the section tab should not be displayed. The default is false.
header-type	<i>Optional.</i> This indicates that an additional header is to be used and what type of content will be provided. The values supported are static and dynamic. See Section 6.5.3, <i>Optional Header</i> for more information.
header-source	<i>Optional.</i> A reference to the source that will be used as an additional header. The value of this depends on the value of <code>header-type</code> . For static content, the attribute should reference a filename of a file in the resource store. For dynamic content, the attribute should reference a custom widget. See Section 6.5.3, <i>Optional Header</i> for more information.

Table 6.2 Attributes of the application Element

The application element supports the child elements detailed in Table 6.3, *Supported Child Elements of the application Element*.

Element	Description
section-ref	<i>1..n.</i> The application must contain a minimum of one <code>section-ref</code> element. Each <code>section-ref</code> element references a section to be included in the

Element	Description
	application. See Section 6.5.2.3, <i>section-ref</i> for more information.
application-menu	<i>Optional.</i> Allows for the optional addition of links to the application banner. The links supported include the user preferences editor, application logout and help. See Section 6.5.2.1, <i>application-menu</i> for more information.
application-search	<i>Optional.</i> Allows for the optional addition of a quick search facility on the application banner. See Section 6.5.2.2, <i>application-search</i> for more information.

Table 6.3 Supported Child Elements of the application Element

application-menu

The application menu forms part of the application banner, and allows for the optional addition of up to three links, specifically a link to the application help, a link to logout of the application and a link to open the user preferences dialog. Refer to User Interface Element 1.3 in Figure 6.1, *Application User Interface Overview* to see an example of an application menu configured in the Application Banner.

Each link is defined as a child element of `application-menu` element and the supported elements are detailed in Table 6.4, *Supported Child Elements of the application-menu Element*.

Element	Description
preferences	<i>Optional.</i> Defines a link to the user preferences dialog. This dialog allows a user to configure customizations for the application view. The title of the <code>preferences</code> link is defined using the supported <code>title</code> attribute. The value of the <code>title</code> attribute should be a reference to an entry in the associated properties file.
help	<i>Optional.</i> Defines a link to the general help for the Cúram application. The title of the <code>help</code> link is defined using the supported <code>title</code> attribute. The value of the <code>title</code> attribute should be a reference to an entry in the associated properties file.

Element	Description
logout	<p><i>Optional.</i> Defines a link to allow a user to end their session and logout of the application.</p> <p>The title of the <code>logout</code> link is defined using the supported <code>title</code> attribute. The value of the <code>title</code> attribute should be a reference to an entry in the associated properties file.</p>

Table 6.4 Supported Child Elements of the application-menu Element

application-search

Refer to User Interface Element 1.4 in Figure 6.1, *Application User Interface Overview* to see an example of a fully configured application search in the User Interface.

The application search, is defined using the `application-search` element. In its simplest form, the `application-search` element requires two attributes, which are used when there is only one type of search and no combo box is to be displayed:

Attribute	Description
default-search-page	<p><i>Optional.</i> A reference to the UIM page that will be displayed when the search button is clicked.</p> <p>When this attribute is used, it is assumed there is only one type of search and no search type combo box is displayed.</p>
initial-text	<p><i>Optional.</i> The text to be displayed in the text entry field as a prompt. This text should describe what type of information can be provided for the search, e.g. Enter a participant reference number.</p> <p>The attribute must reference an entry in the associated properties file.</p>

Table 6.5 Attributes of the application-search Element

The `application-search` element supports two child elements, detailed in Table 6.6, *Supported Child Elements of the application-search Element*, which are used for more complex style searches.

Element	Description
search-pages	<i>Optional.</i>

Element	Description
	Defines multiple types of search. See Section 6.5.2.2.1, <i>search-pages</i> for more information.
further-options-link	<i>Optional.</i> Defines a link to a more advanced search page. See Section 6.5.2.2.2, <i>further-options-link</i> for more information.

Table 6.6 Supported Child Elements of the application-search Element

search-pages

The `search-pages` element is used when multiple search types are required, e.g. Person, Case, or types of search, e.g. Person Surname, Person Reference Number. Each search type is listed in a combo box and a different prompt is displayed in the text entry field depending on the selected entry in the combo box.

The `search-pages` element supports the child elements detailed in Table 6.7, *Supported Child Elements of the search-pages Element*.

Element	Description
search-page	<i>1..n.</i> Defines a single search type. The attributes of the <code>search-page</code> element are defined in Table 6.8, <i>Attributes of the search-page Element</i> .

Table 6.7 Supported Child Elements of the search-pages Element



Note

Where the `search-pages` element is used to define multiple types of search, the `initial-text` and `default-search-page` must not be specified.

Attribute	Description
type	<i>Mandatory.</i> The unique identifier for the type of search. It will be passed as a parameter (<code>searchType</code>) to the UIM page invoked when the application search is performed.
description	<i>Mandatory.</i> The text to be displayed for the search option in the combo box. The attribute must reference an entry in the associated properties file.
page-id	<i>Mandatory.</i>

Attribute	Description
	A reference to a UIM page that will be displayed when the search button is clicked.
initial-text	<i>Mandatory.</i> The text to be displayed as a prompt in the text entry field when that business object is selected in the combo box. The attribute must reference an entry in the associated properties file.
default	<i>Optional.</i> A boolean indicating if this entry is the default entry to be selected in the combo box. One, and only one, entry should have the default specified as true.

Table 6.8 Attributes of the search-page Element

**Note**

Blank values are not allowed in the search type combo box, so if the user requires a generic search (i.e. across all business objects), they must provide configuration data for this. For example, a business object of "All" linked to a page that will carry out the search across all the business objects that have been defined.

Search pages are linked using a reference to the UIM page to be opened when the search button is clicked. The UIM pages defined for a search can expect a number of parameters to be passed to them and used as part of the search:

- **searchText.** The search text that has been entered in the text entry field.
- **searchType.** The selected search type. This is only applicable where multiple search types have been defined.

For more information on creation of UIM pages see Chapter 5, *UIM Reference*

further-options-link

In addition to multiple search types, the application search also supports a link to a more advanced search page. This is specified using the `further-options-link` element, which requires the following attributes:

Attribute	Description
description	<i>Mandatory.</i> The text of the link. The attribute must reference an entry in the associated properties file.
page-id	<i>Mandatory.</i> A reference to a UIM page that will be displayed when the link is clicked. This UIM page should re-

Attribute	Description
	quire no page parameters.

Table 6.9 Attributes of the further-options Element

section-ref

An application must reference a minimum of one, and up to a maximum of five sections, using the `section-ref` element. See Section 6.6, *Sections* for more information.

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The id of a section configuration file (<code>.sec</code>).

Table 6.10 Attributes of the section-ref Element

6.5.3 Optional Header

A custom header can be specified in addition to, or instead of, the application banner. The optional header is defined using the `header-type` and `header-source` attributes on the `application` element and can be defined as either a static HTML fragment or as a custom widget.

Where the header is required instead of the application banner, the optional attributes of the `applications` element, as listed in Table 6.2, *Attributes of the application Element*, should be omitted.

The `header-type` attribute is restricted to the values `static` or `dynamic`. Setting a static value indicates that a HTML fragment is to be placed within the header. In this instance, the `header-source` attribute should reference a file that is stored in the resource store. This file must be stored with a content type of `text/xml`.

If the `header-type` attribute is set to `dynamic`, the `header-source` attribute should reference the custom widget to be used to display the content. This reference will be the same as that specified with the relevant `styles-config.xml`. For more information on creating and referencing custom widgets please consult the *Cúram Custom Widget Development Guide*.

Whether a custom widget or HTML fragment is used it must always start with a `<div>` element.

6.5.4 Example

Example 6.1, *Simple.app* details an example application, which would be stored in a file called `SimpleApp.app`.


```

<?xml version="1.0" encoding="ISO-8859-1"?>
<ac:application
  id="SimpleApp"
  logo="SimpleApp.logo"
  title="SimpleApp.title"
  subtitle="SimpleApp.subtitle"
  user-message="SimpleApp.UserMessage">

  <ac:application-menu>
    <ac:preferences title="preferences.title"/>
    <ac:help title="help.title"/>
    <ac:logout title="logout.title"/>
  </ac:application-menu>

  <ac:application-search>
    <ac:search-pages>
      <ac:search-page type="SAS01"
        description="Search.Person.LastName.Description"
        page-id="Person_searchResolver"
        initial-text="Search.Person.LastName.InitialText"
        default="true" />
      <ac:search-page type="SAS02"
        description="Search.Person.Gender.Description"
        page-id="Person_listByGender"
        initial-text="Search.Person.Gender.InitialText" />
    </ac:search-pages>
    <ac:further-options-link
      description="Search.Further.Options.Link.Description"
      page-id="Person_search" />
  </ac:application-search>

  <ac:section-ref id="SimpleHomeSection"/>
  <ac:section-ref id="SimpleWorkspaceSection"/>
</ac:application>

```

Example 6.1 Simple.app



Note

In the above example a namespace, `ac` has been declared and all elements are prefixed with the namespace. This is recommended practice. Consult Section 3.12.12, *Application Configuration Files* for more information.

The `SimpleApp.app` should have a corresponding `SimpleApp.properties` file, which details the localizable content. For example:

```

SimpleApp.logo=CDEJ/themes/v6/images/application-logo.png
SimpleApp.title=C\u00F0Faram
SimpleApp.subtitle=Simple Application
SimpleApp.UserMessage=Welcome, %user-full-name

preferences.title=User Preference
help.title=Help
logout.title=Logout
Search.Person.LastName.Description=Surname
Search.Person.LastName.InitialText=Enter surname to search for
Search.Person.Gender.Description=Gender
Search.Person.Gender.InitialText=Enter gender to search for
Search.Further.Options.Link.Description=Advanced Search

```

In the above example, the Cúram logo image is referencing the default logo image shipped with the Cúram Client Development Environment (CDEJ). A custom logo can be added to the Images folder in the component and referenced directly as `Images/my-custom-logo.png`.



Note

In the properties file for the `SimpleApp.app` example, the `ú` in Cúram is added using the Unicode escape sequence. An alternative approach is to add the `ú` directly and ensure the file is saved in the UTF-8 format. Both approaches are supported for the application configuration files.

6.5.5 Associate an Application with User

A user must be mapped to the application and home page to display when they first login. The home page is the initial page, displayed in its associated tab. This is done using the following mapping:

- `APPLICATIONCODE` field on the `Users` database table

maps to

- an entry in the `APPLICATION_CODE` codetable

maps to

- the `id` attribute of an application

When a user logs in, the value of the `APPLICATIONCODE` field in the `Users` database table is used to determine both the application and home page to display.

The `value` field of the code table entry must match the name of the application (.app) file to use and the `description` field of the code table entry indicates the name of the UIM page to be displayed as the home page. The following example shows a subset of a code table definition:

```
<codetable java_identifier="APPLICATION_CODE"
  name="APPLICATION_CODE">
  <code default="false" java_identifier="SIMPLE_HOME"
    status="ENABLED" value="SimpleApp">
    <locale language="en" sort_order="0">
      <description>SimpleHome</description>
      <annotation></annotation>
    </locale>
  </code>
</codetable>
```

Example 6.2 CT_APPLICATIONCODE.ctx



Note

For more information on code tables see the *Cúram Server De-*

velopers Guide.

In this example, a code table entry SimpleApp has been defined, with a description of SimpleHome. The code SimpleApp, matches the id of the SimpleApp.app example. The description, SimpleHome, indicates the UIM page to be displayed as the home page. This page must be associated with the relevant application. For more details on how to associate pages with an application, see Section 6.11, *Opening Tabs and Sections*.

6.6 Sections

6.6.1 Introduction

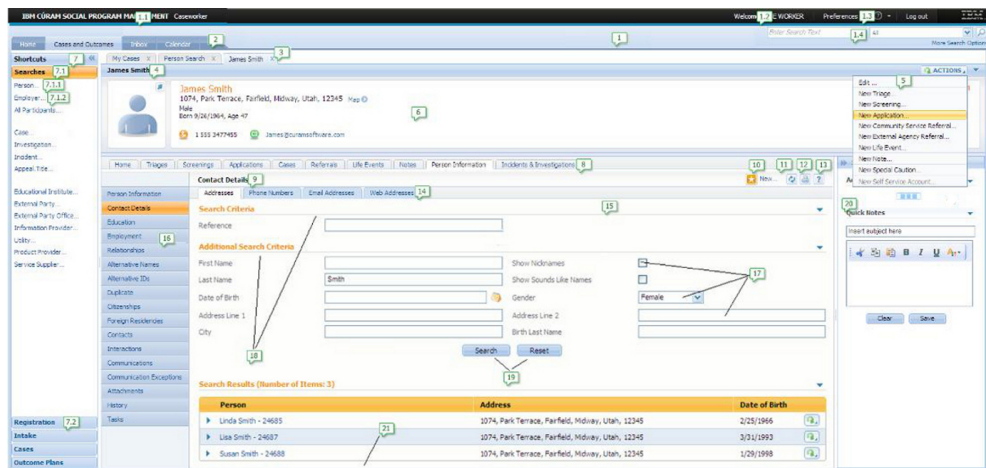


Figure 6.2 Application User Interface Overview

An application can contain one or more application sections, where a section is a collection of tabs and an optional section shortcut panel. A section shortcut panel supports quick links to open tabs and dialogs within a section.

It is recommended that a maximum of five sections be used, each representing a different set of activities that can be performed by a user. The five recommended types of sections are:

Refer to User Interface Element 2 in Figure 6.2, *Application User Interface Overview* to see sections configured in the User Interface. The section that is currently open is a lighter shade of color than the other sections.

- **Home.** The Home section is intended to contain only one tab, with a single page that acts as a home page for the user. The home page should provide a summary of significant information and quick links to common activities.
- **Workspace.** The Workspace section is where the majority of tasks relating to the user role will be performed.
- **Inbox.** The Inbox section represents the area of the application where

the user can access the work currently allocated to them.

- **Calendar.** The Calendar section contains a calendar of the users activities and schedules.
- **Reports.** The Reports section contains a number of reports relevant for the particular user.

6.6.2 Definition

A section is defined by creating an XML file with the extension `.sec` in the `clientapps` directory. The root XML element in the `.sec` file is the `section` element and the attributes allowed on this element are defined in Table 6.11, *Attributes of the section Element*.

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The unique identifier for the section, which must match the name of the file. This is used when referenced from an application (<code>.app</code>) configuration file.
<code>title</code>	<i>Mandatory.</i> The text for the title that will be displayed on the section tab. The attribute must reference an entry in the associated properties file.
<code>hide-tab-container</code>	<i>Optional.</i> When set to true, this indicates that there is only one tab in the section and the tab bar should not be displayed. The default is false.
<code>default-page-id</code>	<i>Optional.</i> A reference to a UIM page that should be opened by default when the section is opened. The UIM page referenced must be directly associated with a tab. For more information on associating pages with tabs, consult Section 6.8, <i>Tabs</i> . This attribute ensures that an anchored default tab is always open when the section is opened. An anchored tab does not contain an option to close it.

Table 6.11 Attributes of the section Element



Note

The `default-page-id` attribute must not be used on the "Home" or first section of an application. The user's home page, and its associated tab are opened automatically when a user logs into an application. See Section 6.5.5, *Associate an Application with User* for more information.

The `section` element supports the child elements detailed in Table 6.12, *Supported Child Elements of the section Element*.

Element	Description
<code>tab</code>	<i>1..n.</i> A reference to a tab to be included in this section. See Section 6.6.2.1, <i>tab</i> for more information.
<code>shortcut-panel-ref</code>	<i>Optional.</i> A reference to the section shortcut panel to be included in this section. See Section 6.6.2.2, <i>shortcut-panel-ref</i> for more information.

Table 6.12 Supported Child Elements of the section Element

`tab`

A section is a collection of tabs and to associate a tab with a section the `tab` element should be used. A `section` must define at least one `tab` element and tabs must only ever be referenced by one section in any application. This means that tabs can be reused in different sections, as long as the section is included in a separate application.

The attributes of the `tab` element are detailed in Table 6.13, *Attributes of the tab Element*

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The id of a tab configuration file (<code>.tab</code>). See Section 6.6.2.1, <i>tab</i> for more information.

Table 6.13 Attributes of the tab Element

`shortcut-panel-ref`

The `shortcut-panel-ref` element is used to define the section shortcut panel to add to the section. Only one `shortcut-panel-ref` should be specified per section. See Section 6.7, *Section Shortcut Panel* for more information.

The attributes of the `shortcut-panel-ref` element are detailed in Table 6.14, *Attributes of the shortcut-panel-ref Element*

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The id of a section shortcut panel (<code>.sec</code>). See Section 6.7, <i>Section Shortcut Panel</i> for more information.

Table 6.14 Attributes of the shortcut-panel-ref Element

6.6.3 Example

Example 6.3, *SimpleWorkspaceSection.sec* details an example section, which would be stored in a file called `SimpleWorkspaceSection.sec`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sc:section
  id="SimpleWorkspaceSection"
  title="SimpleWorkspaceSection.title">

  <sc:shortcut-panel-ref id="SimpleShortcutPanel"/>

  <sc:tab id="Person" />
  <sc:tab id="Employer" />
  <sc:tab id="Case" />
  ...
</sc:section>
```

Example 6.3 SimpleWorkspaceSection.sec

The `SimpleWorkspaceSection.sec` should have a corresponding `SimpleWorkspaceSection.properties` file, which details the localizable content. For example:

```
SimpleWorkspaceSection.title=Workspace
```

6.7 Section Shortcut Panel

6.7.1 Introduction

Each section can optionally contain a section shortcut panel which provides quick links to open content and perform actions within the section. The menu items in the shortcut panel can be divided into categories. Refer to User Interface Element 7 of Figure 6.2, *Application User Interface Overview* to see an example of a configured section shortcut panel.

When a section is first opened, the section shortcut panel is collapsed by default. The double arrow beside the title of the shortcut panel can be used to expanded, and subsequently collapse, the panel.

Menu items in a shortcut panel which open modal dialogs are identified by an ellipses (...), which indicates that further actions are required. Refer to User Interface Element 7.1.1 of Figure 6.2, *Application User Interface Overview* to see an example of a configured menu item in an expanded category of a shortcut panel.

6.7.2 Definition

A section shortcut panel is defined by creating an XML file with the extension `.ssp` in the `clientapps` directory. The root XML element in the `.ssp` file is the `section-shortcut-panel` element and the attributes allowed on this element are defined in Table 6.15, *Attributes of the section-shortcut-panel Element*.

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The unique identifier for the section shortcut panel, which must match the name of the file. This is used when referenced from a section (<code>.sec</code>) configuration file.
<code>title</code>	<i>Mandatory.</i> The text for the title that will be displayed for the sections shortcut panel, both when it is expanded and when it is collapsed. The attribute must reference an entry in the associated properties file.

Table 6.15 Attributes of the section-shortcut-panel Element

The `section-shortcut-panel` element supports the child elements detailed in Table 6.16, *Supported Child Elements of the section-shortcut-panel Element*.

Element	Description
<code>nodes</code>	<i>Mandatory.</i> Groups together multiple child <code>node</code> elements. See Section 6.7.2.1, <i>node</i> for more information.

Table 6.16 Supported Child Elements of the section-shortcut-panel Element

node

The `node` element is used to represent menu items and categories used within the shortcut panel. There are three supported types of `node` element and the `type` attribute is used to define this:

- **group.** A group node in a shortcut panel represents a category and is used to categorize a number of menu items as described in Section 6.7, *Section Shortcut Panel*. “Registration” are defined using `node` Each category is defined using `node` elements of type `group`. This type of `node` supports child `node` elements of type `leaf` and `separator`.
- **leaf.** A leaf in a shortcut panel is a menu item within a category, which

can open a page in an existing or new tab, or open a modal dialog ². Where a menu item opens a modal dialog, an ellipsis is appended to the text displayed to indicate more information is required.

- **separator.** A separator can be used to add extra space between menu items within a node of type group (i.e. a category).

The attributes supported by the node element are detailed in Table 6.17, *Attributes of the node Element*.

Attribute	Description
id	<i>Mandatory.</i> The identifier for the node. This must be unique within the .ssp file.
type	<i>Mandatory.</i> The type of node, where three types are supported: <ul style="list-style-type: none"> • group • leaf • separator
title	<i>Mandatory.</i> The text for the title of the node. The attribute must reference an entry in the associated properties file. Note: This is not required where the type is specified as separator.
page-id	<i>Optional.</i> A reference to the UIM page to be displayed when the menu item is selected. This is only applicable for node elements with a type of leaf.
open-as	<i>Optional.</i> Where set, this attribute indicates the UIM page to be displayed when the menu item is selected should be opened as a modal dialog. The only value supported is <i>modal</i> . This is only applicable for node elements with a type of leaf.
append-ellipsis	<i>Optional.</i> A boolean attribute which indicates if the ellipsis automatically appended to the menu item which opens in a modal dialog should be disabled. The default is true. The attribute is applicable only where the type attribute is leaf and the open-as attribute has been set. Note: Setting this attribute to true where the open-as attribute has not been set will not add the ellipsis

Attribute	Description
	to the menu item.

Table 6.17 Attributes of the node Element

6.7.3 Example

Example 6.4, *SimpleShortcutPanel.ssp* details an example section shortcut panel, which would be stored in a file called `SimpleShortcutPanel.ssp`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sc:section-shortcut-panel
  id="SimpleShortcutPanel"
  title="SimpleShortcutPanel.Title">

  <sc:nodes>
    <sc:node id="Searches" type="group"
      title="Searches.Title">
      <sc:node id="PersonSearch" type="leaf"
        page-id="Person_search"
        title="PersonSearch.Title" />
      ...
    </sc:node>
    <sc:node id="QuickLinks" type="group"
      title="QuickLinks.Title">
      ...
    </sc:node>
    <sc:node id="Registration" type="group"
      title="Registration.Title">
      <sc:node id="RegisterEmployer" type="leaf"
        page-id="Employer_register"
        title="RegisterEmployer.Title"
        open-as="modal" />
      ...
      <sc:node type="separator" id="separator" />
      ...
    </sc:node>
  </sc:nodes>
</section-shortcut-panel>
```

Example 6.4 SimpleShortcutPanel.ssp

The `SimpleShortcutPanel.ssp` should have a corresponding `SimpleShortcutPanel.properties` file, which details the localizable content. For example:

```
SimpleShortcutPanel.Title=Shortcuts Panel
Searches.Title=Searches
PersonSearch.Title=Person Search
QuickLinks.Title=Quick Links
Registration.Title=Registration
RegisterEmployer.Title=Register an Employer
```

6.8 Tabs

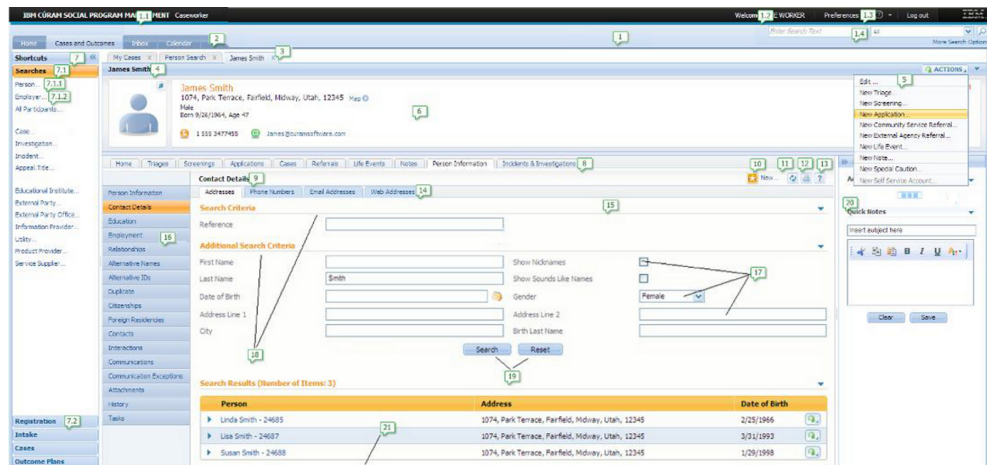


Figure 6.3 Application User Interface Overview

6.8.1 Introduction

A tab typically represents a business object, e.g. a Case or a Participant, though it can also be used to represent a logical grouping of information. Refer to User Interface Element 3 of Figure 6.3, *Application User Interface Overview* for an example of a configured tab in an application.

- **Tab Title Bar.** The title bar contains text to identify the current tab. Refer to User Interface Element 4 of Figure 6.3, *Application User Interface Overview* for an example of a tab title bar configured in an application.
- **Tab Actions Menu.** The actions menu provides actions associated with the business object represent by the tab. The actions can be a mix of menu items and other menus, each of which links to a page that will be displayed in the tab content area or a modal dialog. Refer to User Interface Element 5 of Figure 6.3, *Application User Interface Overview* for an example of a tab actions menu configured in an application.
- **Tab Context Panel.** The context panel is typically used to present summary information about the business object. This summary information is always available, no matter what page is displayed in the content area. Refer to User Interface Element 6 of Figure 6.3, *Application User Interface Overview* for an example of a tab context panel configured in an application.

The context panel can be collapsed and expanded to provide more space for the tab content area.

- **Tab Content Area.** A tab comprises of one or more pages of information. These pages are displayed in the content area and can be navigated using the navigation bar.
 - **Navigation Bar.** The navigation bar contains a number of naviga-

tion tabs, each of which link to a page or set of pages that are part of the tab. The navigation bar can be used to separate the business object information into logical groupings of pages. Refer to User Interface Element 8 of Figure 6.3, *Application User Interface Overview* for an example of a navigation bar configured in an application.

- **Page Group Navigation Bar.** Where a tab links to a set of pages, the pages are displayed as a page group navigation bar, with the first one selected by default. Refer to User Interface Element 16 of Figure 6.3, *Application User Interface Overview* for an example of a page group navigation bar configured in an application.
- **Page Content.** Selecting a navigation tab or page group entry will display the corresponding UIM page content within the content area. Refer to User Interface Element 15 of Figure 6.3, *Application User Interface Overview* for an example of a page content area configured in an application.

In addition to the above elements a Tab also supports an optional smart panel. A smart panel is an optional panel, displaying a UIM page, that is added to the right of the content area in a tab. It can be collapsed and expanded, and is collapsed by default. In addition, the size of the smart panel can be increased and decreased when it is expanded. Refer to User Interface Element 20 of Figure 6.3, *Application User Interface Overview* for an example of a smart panel configured in an application.

Finally, a tab supports the ability to dynamically enable/disable and hide/show entries in the tab actions menu, tab navigation bar and page group navigation bar. This dynamic content is updated based on configured refresh events.

A refresh event updates the specified part of the tab based on the submit of a modal dialog page or when a specific UIM page is loaded in the content area. For more information on configuring refresh events consult Section 6.8.2.6, *tab-refresh*.

6.8.2 Definition

A tab is defined by creating an XML file with the extension `.tab` in the `clientapps` directory. The root XML element in the `.tab` file is the `tab-config` element and the attributes required by this are defined in Table 6.18, *Attributes of the tab-config Element*.

Attribute	Description
<code>id</code>	<p><i>Mandatory.</i> The identifier for the tab, which must match the name of the file.</p> <p>The <code>id</code> attribute is used to reference the tab configuration from section configuration files (<code>.sec</code>). See Section 6.6.2.1, <i>tab</i> for more information.</p>

Table 6.18 Attributes of the tab-config Element

The `tab-config` element supports the child elements detailed in Table 6.19, *Supported Child Elements of the tab-config Element*.

Element	Description
<code>page-param</code>	<i>0..n.</i> Defines a parameter required when opening a tab. See Section 6.8.2.1, <i>page-param</i> for more information.
<code>menu</code>	<i>Optional.</i> A reference to the actions menu configuration. See Section 6.8.2.2, <i>menu</i> for more information.
<code>context</code>	<i>Mandatory.</i> A reference to the UIM page to be used as the tab context panel, or alternatively details of the tab name and title. See Section 6.8.2.3, <i>context</i> for more information.
<code>navigation</code>	<i>Mandatory.</i> A reference to the tab navigation configuration, or alternatively the name of the UIM page that will be opened in this tab. See Section 6.8.2.4, <i>navigation</i> for more information.
<code>smart-panel</code>	<i>Optional.</i> A reference to the UIM page to be used for the smart panel. See Section 6.8.2.5, <i>smart-panel</i> for more information.
<code>tab-refresh</code>	<i>Optional.</i> Defines what part of a tab should refresh under what circumstances. See Section 6.8.2.6, <i>tab-refresh</i> for more information.

Table 6.19 Supported Child Elements of the tab-config Element

page-param

The `page-param` element allows for multiple page parameters to be defined for a tab. Each page parameter defined maps to the name of a name-value pair that will be passed to all UIM pages that are opened from both the tab actions menu and the navigation bar.

Page parameters are also used to identify unique instances of a tab. For example, a tab is defined for a Person object. Two instances of this tab can be opened, one for James Smith and one for Linda Smith. The instances are uniquely identified by the page parameter, `id`, which has been defined for the tab. This `id` parameter maps to the unique id for the person and will be different for both James Smith and Linda Smith.

For more information on the behavior associated with opening tabs see Section 6.11, *Opening Tabs and Sections*.

Attribute	Description
name	<i>Mandatory.</i> A unique identifier for the page parameter.

Table 6.20 Attributes of the page-param Element

menu

The menu element contains a reference to the tab action menu configuration which is maintained in a separate configuration file, (.mnu). See Section 6.9, *Tab Actions Menu* for more information.

Attribute	Description
id	<i>Mandatory.</i> A reference to the id of a tab action menu configuration file (.mnu).

Table 6.21 Attributes of the menu Element

context

The context element defines a context panel by referencing a UIM page which forms the content of the context panel. The element is mandatory and if no context panel is to be defined, then a tab name and tab title must be specified.

The tab title bar and tab name can be populated with data using either the context panel UIM page or using the tab-name and tab-title attributes in the .tab file. Where the context panel UIM page is used only to add content to the tab name and tab title, the height attribute should be set to zero.

For more information on defining context panel UIM pages see Section 6.8.3, *Context Panel UIM*

Attribute	Description
page-id	<i>Optional.</i> A reference to the UIM page that will be used for the content of the context panel. If this is not specified, the tab-name and tab-title attributes <i>must</i> be specified.
tab-name	<i>Optional.</i> The text that will be displayed in the tab bar. The attribute must reference an entry in the associated properties file.
tab-title	<i>Optional.</i>

Attribute	Description
	The text that will be displayed in the tab title bar. The attribute must reference an entry in the associated properties file.
height	<i>Optional.</i> The pixel height of the context panel. This is only relevant if a <code>page-id</code> attribute has been specified to define a context panel. The default value if not specified is 150 pixels.

Table 6.22 Attributes of the context Element

navigation

The `navigation` element defines what pages will be opened within the tab. A single page can be defined using the `page-id` attribute, or multiple pages can be defined using a reference to the tab navigation configuration file (`.nav`). For more information on tab navigation configuration see Section 6.10, *Tab Navigation*.



Note

The `navigation` element is mandatory and one of either `page-id` or `id` must be specified.

Attribute	Description
page-id	<i>Optional.</i> A reference to the UIM page that will be opened in the tab. When a link to this UIM page is selected, it will automatically trigger the page to be opened in a new tab.
id	<i>Optional.</i> A reference to a tab navigation configuration file (<code>.nav</code>). See Section 6.10, <i>Tab Navigation</i> for more information.

Table 6.23 Attributes of the navigation Element

smart-panel

The content of the smart panel is defined by a UIM page, referenced by the `page-id` attribute. Like the context panel, the UIM elements that can be used are limited. See Section 6.8.3, *Context Panel UIM* for details of the limitations of the smart panel UIM. Refer to User Interface Element 20 of Figure 6.3, *Application User Interface Overview* for an example of a smart panel configured in an application.

Attribute	Description
page-id	<i>Mandatory.</i> A reference to the UIM page that will be displayed in the smart panel of the tab.
title	<i>Mandatory.</i> The text for the title that will be displayed for the smart panel, both when it is expanded and when it is collapsed. The attribute must reference an entry in the associated properties file
width	<i>Optional.</i> The initial width of the smart panel when it is expanded. The default value if this attribute is not set is 250 pixels.
collapsed	<i>Optional.</i> Boolean indicating if the smart panel should be expanded or collapsed by default. The default value if this attribute is not set is true.

Table 6.24 Attributes of the smart-panel Element

tab-refresh

By default, only the content area of a tab is refreshed when a modal dialog is submitted. When a modal dialog is closed/cancelled, i.e. no action is performed, the content area is not refreshed.

The `tab-refresh` element allows different aspects of a tab to be refreshed. The tab actions menu, tab navigation and context panel can all be refreshed based on two events. The first is when a specific UIM page is loaded in the content area and the second when a UIM page is submitted from a modal or the content area.

- **Tab Actions Menu.** Refreshing the tab actions menu results in updating the entries in the menu that can be dynamically disabled or hidden. For more information on dynamic support in the tab actions menu see Section 6.9.3, *Dynamic Support*.
- **Tab Navigation.** Refreshing the tab navigation results in updating the entries in the tab navigation bar and page group navigation bar that can be dynamically disabled or hidden. For more information on dynamic support in tab navigation see Section 6.10.3, *Dynamic Support*.
- **Context Panel.** Refreshing the context panel simply reloads the UIM page displayed in the context panel.
- **Content Area.** Refreshing the content area reloads the UIM page displayed in the content area. This refresh option is available for use only where a modal dialog has been opened from the list dropdown panel of a nested expandable list.

By default only the parent of list dropdown panel is updated when the modal dialog is submitted. Where the list dropdown panel exists in a nested expandable list, this will result in the parent list reloading and not the entire content area.

Under some circumstances, the entire content area may require updating and this option can be used to achieve this for this specific scenario.

The two different type of refresh events can be configured using the child elements detailed in Table 6.25, *Supported Child Elements of the tab-refresh Element*.

Element	Description
onload	<i>1..n.</i> Defines a refresh event, where when the specified page is loaded in the content area, the defined parts of the tab are updated.
onsubmit	<i>1..n.</i> Defines a refresh event, where when the specified page is submitted from a modal or in the content area, the defined parts of the tab are updated.

Table 6.25 Supported Child Elements of the tab-refresh Element

onsubmit/onload

The onsubmit and onload elements both require the same set of attributes, as described in Table 6.26, *Attributes of the onload/onsubmit Elements*.

Attribute	Description
page-id	<i>Mandatory.</i> A reference to the UIM page to associate with the refresh event.
context	<i>Optional.</i> Boolean indicating if the context panel should be update when the specified page is loaded or submitted.
menu-bar	<i>Optional.</i> Boolean indicating if the tab actions menu should be updated when the specified page is loaded or submitted. See Section 6.9.3, <i>Dynamic Support</i> for more information.
navigation	<i>Optional.</i> Boolean indicating if the tab navigation should be updated when the specified page is loaded or submitted. See Section 6.10.3, <i>Dynamic Support</i> for more information.

Attribute	Description
main-content	<p><i>Optional.</i></p> <p>Boolean indicating if the main content area should be updated when the specified page is loaded or submitted.</p> <p>This type of refresh event must only be used for modal dialogs that are opened from a list dropdown panel in a nested expandable list.</p>

Table 6.26 Attributes of the onload/onsubmit Elements

6.8.3 Context Panel UIM

A context panel is a specific type of UIM page identified by the `PAGE` element containing an attribute of `TYPE="DETAILS"`.

This type of UIM page can only use a subset of existing UIM elements. Specifically:

- `SERVER_INTERFACE` can only be used with a `DISPLAY` phase
- `ACTION_CONTROL` can only be used with an `ACTION` type
- The following elements are not supported:
 - `MENU`
 - `SHORTCUT_TITLE`
 - `JSP_SCRIPTLET`
 - `DESCRIPTION`
 - `INFORMATIONAL`
 - `SCRIPT`
 - `INCLUDE`
 - `VIEW`



Note

These same limitations apply to the smart panel UIM pages, but are not enforced.

A mandatory `TAB_NAME` element is required for context panel UIM pages, which allows for dynamic information to be added to the tab name. In addition the `PAGE_TITLE` element is used to add information to the tab title bar. For more information on these elements see Section 5.9.31, `TAB_NAME` and Section 5.9.27, `PAGE_TITLE`.

6.8.4 Example

Example 6.5, *SimpleTab.tab* details an example tab configuration file, which would be stored in a file called `SimpleTab.tab`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tc:tab-config
  id="SimpleTab">

  <tc:page-param name="concernroleid"/>

  <tc:menu id="SimpleMenu"/>

  <tc:context page-id="SimpleDetailsPanel"
    tab-name="simple.tab.name" />

  <tc:navigation id="SimpleNavigation"/>

  <tc:smart-panel page-id="SimpleSmartPanel"
    title="smart.panel.title"
    collapsed="true"
    width="300" />

  <tc:tab-refresh>
    <tc:onload page-id="SimpleHome" navigation="true"/>
    <tc:onsubmit page-id="ModifySomething"
      context="true" menu-bar="true"/>
  </tc:tab-refresh>
</tc:tab-config>
```

Example 6.5 SimpleTab.tab

The `SimpleTab.tab` should have a corresponding `SimpleTab.properties` file, which details the localizable content. For example:

```
simple.tab.name=Simple Tab
smart.panel.title=Smart Panel
```

6.9 Tab Actions Menu

6.9.1 Introduction

The tab actions menu is a dropdown menu in the tab title bar. The menu items listed in the menu allow actions specific to the tab to be performed.

The items support opening UIM pages in the content area of a tab, or alternatively opening a modal dialog to perform some action - these are identified by an ellipsis (...). Additionally, it is possible to download a file directly from a menu item.

The tab actions menu also supports the ability to dynamically hide/show and

enable/disable items in the menu. Refer to User Interface Element 5 of Figure 6.3, *Application User Interface Overview* for an example of a tab actions menu configured in an application. The menu items that are dynamically hidden are disabled in the menu.

6.9.2 Definition

A tab actions menu is defined by creating an XML file with the extension `.mnu` in the `clientapps` directory. The root XML element in the `.mnu` file is the `menu-bar` element and the attributes allowed on this element are defined in Table 6.27, *Attributes of the menu-bar Element*.

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The unique identifier for the menu, which must match the name of the file. The identifier is used when a menu is included in a tab configuration, using the <code>menu</code> element. See Section 6.8.2.2, <i>menu</i> for more information.

Table 6.27 Attributes of the menu-bar Element

A menu definition can be reused and referenced by multiple tab configurations. The menu itself comprises of menu items and submenus, which are used to group menu items. The child elements outlined in Table 6.28, *Supported Child Elements of the menu-bar Element* are used to define the structure of the menu.

Element	Description
<code>menu-item</code>	<i>0..n.</i> Defines a single entry in the menu, which links to a UIM page that can be opened in a modal dialog or in the content area of a tab. See Section 6.9.2.1, <i>menu-item</i> for more information.
<code>submenu</code>	<i>0..n.</i> Defines a grouping of menu items, which form a submenu. See Section 6.9.2.2, <i>submenu</i> for more information.
<code>menu-separator</code>	<i>0..n.</i> Defines a separator line between entries in the menu. See Section 6.9.2.3, <i>menu-separator</i> for more information.
<code>loader-registry</code>	<i>Optional.</i> Defines the server interfaces that can be called to dynamically change the state of the <code>menu-items</code> . See Section 6.9.2.4, <i>loader-registry</i> for more information.

Table 6.28 Supported Child Elements of the menu-bar Element

menu-item

An action entry in the tab actions menu is defined by the `menu-item` element. The attributes of this element are defined in Table 6.29, *Attributes of the menu-item Element*.

A `menu-item` can

- open a UIM page in the content area of a tab;
- open a UIM page in a modal dialog.
- download a file.

Menu items which open modal dialogs are identified by an ellipsis (...), which indicates that further actions are required.

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The unique identifier for the <code>menu-item</code> , which must be unique within the configuration file.
<code>page-id</code>	<i>Mandatory.</i> A reference to the UIM page to open when the <code>menu-item</code> is selected.
<code>title</code>	<i>Mandatory.</i> The text that will be displayed for the <code>menu-item</code> . The attribute must reference an entry in the associated properties file.
<code>open-as</code>	<i>Optional.</i> Where set, this attribute indicates that the UIM page to be displayed should be opened as a modal dialog. The only value supported is <i>modal</i> .
<code>append-ellipsis</code>	<i>Optional.</i> A boolean attribute which indicates if the ellipsis automatically appended to <code>menu-items</code> which open in a modal dialog should be displayed. The default is true. The attribute is applicable only where the <code>open-as</code> attribute has been set. Note: Setting this attribute to true where the <code>open-as</code> attribute has not been set will not add the ellipsis to the <code>menu-item</code> .
<code>window-options</code>	<i>Optional.</i> Defines the height and width of a modal dialog opened from the <code>menu-item</code> . This is only applicable where the <code>open-as</code> attribute is set to <i>modal</i> .

Attribute	Description
	<p>The format for the attribute is:</p> <pre>width=<pixel value>,height=<pixel value></pre> <p>For example:</p> <pre>window-options="width=500,height=300"</pre> <p>The <code>height</code> portion of the <code>window-options</code> is optional and if not specified, the height of the dialog will be automatically calculated.</p>
<code>dynamic</code>	<p><i>Optional.</i></p> <p>Boolean indicating that the <code>menu-item</code> can be dynamically disabled or hidden. See Section 6.9.3, <i>Dynamic Support</i> for more information.</p>
<code>visible</code>	<p><i>Optional.</i></p> <p>Boolean indicating if the <code>menu-item</code> is hidden or visible. The default is true.</p>
<code>type</code>	<p><i>Optional.</i></p> <p>Defines a <code>menu-item</code> that downloads a file when selected. The only value supported is <code>FILE_DOWNLOAD</code>.</p> <p>For more information see Section 6.9.4, <i>File Download Menu Item</i> for more information.</p>
<code>description</code>	<p><i>Optional.</i></p> <p>Defines text which forms a description for the <code>menu-item</code>. This is used for administration purposes only. The attribute must reference an entry in the associated properties file.</p>

Table 6.29 Attributes of the menu-item Element

submenu

A submenu is a group of menu items and is defined using the `submenu` element. The attributes of the `submenu` element are defined in Table 6.30, *Attributes of the submenu Element*.

Attribute	Description
<code>id</code>	<p><i>Mandatory.</i></p> <p>The unique identifier for the <code>submenu</code>, which must be unique within the configuration file.</p>
<code>title</code>	<p><i>Mandatory.</i></p> <p>The text that will be displayed for the <code>submenu</code>. The attribute must reference an entry in the associated properties file.</p>
<code>description</code>	<p><i>Optional.</i></p>

Attribute	Description
	Defines text which forms a description for the submenu. This is used for administration purposes only. The attribute must reference an entry in the associated properties file.

Table 6.30 Attributes of the submenu Element

The submenu element allows for further submenus to be defined, in addition to including menu items and menu separators. The supported child attributes (Table 6.31, *Supported Child Elements of the submenu Element*) can be used to achieve this.

Element	Description
menu-item	<i>0..n.</i> Defines a single entry in the submenu, which links to a UIM page that can be opened in a modal dialog or in the content area of a tab. See Section 6.9.2.1, <i>menu-item</i> for more information.
submenu	<i>0..n.</i> Defines a further sub grouping of menu items.
menu-separator	<i>0..n.</i> Defines a separator between entries in the submenu. See Section 6.9.2.3, <i>menu-separator</i> for more information.

Table 6.31 Supported Child Elements of the submenu Element

menu-separator

An actions menu, including submenus of this, can include a line separator to divide the entries in the menu. This is defined using a `menu-separator` element. The attributes of the `menu-separator` are outlined in Table 6.32, *Attributes of the menu-separator Element*.

Attribute	Description
id	<i>Mandatory.</i> The unique identifier for the <code>menu-separator</code> .

Table 6.32 Attributes of the menu-separator Element

loader-registry

The `loader-registry` element defines a list of loader implementations that will be used to dynamically enabled/disable and hide/show the menu items in the tab actions menu. For more information see Section 6.9.3, *Dy-*

dynamic Support.

Element	Description
loader	<i>1..n.</i> Defines one or more loader implementations that will be used to dynamically set the visibility and enabled state of the menu items. See Section 6.9.2.5, <i>loader</i> for more information.

Table 6.33 Supported Child Elements of the loader-registry Element

loader

The `loader` element defines a single loader implementation that will dynamically set the state of the menu items in a tab actions menu. For more information see Section 6.9.3, *Dynamic Support*.

Attribute	Description
class	<i>Mandatory.</i> The fully qualified class name of an implementation of the <code>curam.util.tab.impl.DynamicMenuStateLoader</code> interface.

Table 6.34 Attributes of the loader Element

6.9.3 Dynamic Support

The tab actions menu supports the ability to dynamically enable/disable and hide/show entries. This feature is supported using a combination of the `dynamic` attribute of the `menu-item` element, the `loader-registry` element and a Java loader implementation.

The Java loader implementation registered in the navigation configuration will be called when the tab is first loaded and based on the refresh options configured for a tab. The refresh options are configured in the tab configuration file (`.tab`). See Section 6.8.2.6, *tab-refresh* for more information.

A menu item can be specified as dynamic in the menu configuration file (`.mnu`) by adding `dynamic="true"` to the relevant `menu-item` element.

Where the `dynamic` attribute is set, a `loader-registry` is then required and should define the fully qualified classname which implements the `curam.util.tab.impl.DynamicMenuStateLoader` interface.

The `DynamicMenuStateLoader` interface requires one method, `loadMenuState`, to be implemented. The `loadMenuState` method is passed the following parameters:

- a list of menu item identifiers
- a set of name-value page parameters pairs

The loader implementation must decide which menu items to disable or hide. The method returns an object that represents the state of a given menu bar. A state must be set for all identifiers in the list. For more information on this interface, consult the Java Documentation.



Note

The list of menu item identifiers passed to the `loadMenuState` method are only those that have been identified as dynamic by the `dynamic` attribute on the `menu-item` element.

6.9.4 File Download Menu Item

A `menu-item` can reference a `FILE_DOWNLOAD` configuration using the `type="FILE_DOWNLOAD"` attribute. For example:

```
<mc: menu-item id="filedownloadItem" title="some.text.title"
              type="FILE_DOWNLOAD" page-id="FileDownload" />
```

The `page-id` attribute must match the `page-id` attribute specified for a `FILE_DOWNLOAD` element configured in the `curam-config.xml` file. For more information on the `FILE_DOWNLOAD` element in `curam-config.xml` see Section 5.9.3.1, *File Downloads*.

When configuring the `FILE_DOWNLOAD` element in `curam-config.xml`, only the parameters defined for the tab can be used as values for the `PAGE_PARAM` attribute of the `INPUT` element.

Example 6.6, *FILE_DOWNLOAD Configuration from curam-config.xml* shows a fragment of the `FILE_DOWNLOAD` configuration from the `curam-config.xml` file. In this example, the `fileID` page parameter must be specified as a `page-param` element in the tab configuration file (`.tab`).

Note also that the `PAGE_ID` attribute value of `FileDownload` matches the `page-id` attribute in the example above.

```
<FILE_DOWNLOAD CLASS="some.pkg.readFile"
              PAGE_ID="FileDownload">
  <INPUT PAGE_PARAM="fileID"
        PROPERTY="key$fileID" />
  <FILE_NAME PROPERTY="result$name" />
  <FILE_DATA PROPERTY="result$contents" />
  <CONTENT_TYPE PROPERTY="result$contentType" />
</FILE_DOWNLOAD>
```


Example 6.6 FILE_DOWNLOAD Configuration from curam-config.xml

6.9.5 Example

Example 6.7, *SimpleMenu.mnu* details an example actions menu configuration file, which would be stored in a file called *SimpleMenu.mnu*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mc:menu-bar
  id="SimpleMenu"

  <mc:loader-registry>
    <mc:loader class="some.pkg.SimpleMenuStateLoader" />
  </mc:loader-registry>

  <mc:submenu id="Person">

    <mc:menu-item id="dynamicLink"
      title="dynamicLink.title"
      page-id="SomeDynamicContent"
      dynamic="true" />

    <mc:menu-separator id="separator1" />

    <mc:menu-item id="simpleLink"
      title="simpleLink.title"
      page-id="SimplePage" />

  </mc:submenu>

  <mc:menu-item id="OpenModal"
    title="openmodal.title"
    page-id="DoSomethingInModal"
    open-as="modal"
    window-options="width=600" />

</mc:menu-bar>
```

Example 6.7 SimpleMenu.mnu

The *SimpleMenu.mnu* should have a corresponding *SimpleMenu.properties* file, which details the localizable content. For example:

```
dynamicLink.title=Some Dynamic Link
simpleLink.title=A Simple Link
openmodal.title=Open a Modal
```

6.10 Tab Navigation

6.10.1 Introduction

Tab navigation describes how the various UIM pages grouped as part of a

tab can be navigated to within a tab. There are two elements to tab navigation; the Content Area Navigation Bar, and the Page Group Navigation Bar.

- **Navigation Bar.** The navigation bar contains a number of tabs, each of which can map to a single UIM page or alternatively a set of UIM pages. The tabs in the navigation bar are referred to as navigation tabs. Refer to User Interface Element 8 of Figure 6.3, *Application User Interface Overview* for an example of a navigation bar configured in an application.
- **Page Group Navigation Bar.** Where a navigation tab maps to a set of UIM pages, these UIM pages are displayed as a page group navigation bar. Each link in the page group navigation bar is referred to as a navigation page. Refer to User Interface Element 16 of Figure 6.3, *Application User Interface Overview* for an example of a page group navigation bar configured in an application.

Selecting a navigation tab or navigation page will result in displaying the relevant UIM page in the content area of the tab. For navigation tabs that have a page group navigation bar, the first navigation page in the page group navigation bar is selected when the navigation tab is selected.

If a user selects a subsequent navigation page and then changes to a different navigation tab, the selected navigation page is remembered when the user returns to the original navigation tab and the page is reloaded.

The tab navigation configuration is key to when new tabs are opened. It is used to determine what UIM page is associated with what tab. For more information on this consult Section 6.11, *Opening Tabs and Sections*.

6.10.2 Definition

Tab navigation is defined by creating an XML file with the extension `.nav` in the `clientapps` directory. The root XML element in the `.nav` file is the `navigation` element and the attributes allowed on this element are defined in Table 6.35, *Attributes of the navigation Element*.

Attribute	Description
id	<i>Mandatory.</i> The unique identifier for the navigation configuration, which must match the name of the file. The identifier is used when a navigation configuration is included in a tab configuration, using the <code>navigation</code> element. See Section 6.8.2.4, <i>navigation</i> for more information.

Table 6.35 Attributes of the navigation Element

The child elements outlined in Table 6.36, *Supported Child Elements of the navigation Element* are used to define the structure of the navigation.

Element	Description
nodes	<i>Mandatory.</i> Groups navigation pages and navigation tabs together. See Section 6.10.2.1, <i>nodes</i> for more information.
loader-registry	<i>Optional.</i> Defines the server interfaces that can be called to dynamically change the state of the navigation tabs and navigation pages. See Section 6.10.2.4, <i>loader-registry</i> for more information.

Table 6.36 Supported Child Elements of the navigation Element

nodes

The `nodes` element groups together the elements that represent navigation tabs and navigation pages. These elements are outlined in Table 6.37, *Supported Child Elements of the nodes Element*.

Element	Description
navigation-page	<i>1..n.</i> Defines a navigation tab that has no page group navigation bar. See Section 6.10.2.3, <i>navigation-page</i> for more information.
navigation-group	<i>1..n.</i> Defines a navigation tab which contains a page group navigation bar. This element groups together <code>navigation-page</code> elements that form the page group navigation bar. See Section 6.10.2.2, <i>navigation-group</i> for more information.

Table 6.37 Supported Child Elements of the nodes Element

navigation-group

The `navigation-group` element defines a navigation tab that contains a page group navigation bar. The attributes of this element are outlined in Table 6.38, *Attributes of the navigation-group Element*.

Attribute	Description
id	<i>Mandatory.</i> The unique identifier for the <code>navigation-group</code> , which must be unique within the configuration file.
title	<i>Mandatory.</i> The text that will be displayed for the navigation tab

Attribute	Description
	in the navigation bar. The attribute must reference an entry in the associated properties file.
dynamic	<i>Optional.</i> Boolean indicating that the navigation tab can be dynamically disabled or hidden. See Section 6.10.3, <i>Dynamic Support</i> for more information.
visible	<i>Optional.</i> Boolean indicating if the navigation tab is hidden or visible. The default is true.
description	<i>Optional.</i> Defines text which forms a description for the navigation tab. This is used for administration purposes only. The attribute must reference an entry in the associated properties file.

Table 6.38 Attributes of the navigation-group Element

The `navigation-group` element groups together `navigation-page` elements to form the page group navigation bar. The first `navigation-page` element defined indicates the UIM page to display the first time a navigation tab is selected.

Subsequent selections of the navigation tab, for a given instance of a tab, will remember the previously selected navigation page.

Element	Description
navigation-page	<i>1..n.</i> Defines the set of navigation pages that are grouped together to form the page group navigation bar. See Section 6.10.2.3, <i>navigation-page</i> for more information.

Table 6.39 Supported Child Elements of the navigation-group Element

navigation-page

A `navigation-page` element can represent both a navigation tab and navigation page:

- Where the `navigation-page` element is defined a child element of the `nodes` element, it represent a navigation tab which is part of the navigation bar.
- Where the `navigation-page` element is defined a child element of the `navigation-group` element, it represent a navigation page which is part of the page group navigation bar.

The attributes of the `navigation-page` element are outlined in Table 6.40, *Attributes of the navigation-page Element*.

Attribute	Description
<code>id</code>	<i>Mandatory.</i> The unique identifier for the <code>navigation-page</code> , which must be unique within the configuration file.
<code>page-id</code>	<i>Mandatory.</i> A reference to the UIM page to open when the navigation tab or navigation page is selected.
<code>title</code>	<i>Mandatory.</i> The text that will be displayed for the navigation tab or navigation page. The attribute must reference an entry in the associated properties file.
<code>dynamic</code>	<i>Optional.</i> Boolean indicating that the navigation tab or navigation page can be dynamically disabled or hidden. See Section 6.10.3, <i>Dynamic Support</i> for more information.
<code>visible</code>	<i>Optional.</i> Boolean indicating if the navigation tab or navigation page is hidden or visible. The default is true.
<code>description</code>	<i>Optional.</i> Defines text which forms a description for the navigation tab or navigation page. This is used for administration purposes only. The attribute must reference an entry in the associated properties file.

Table 6.40 Attributes of the navigation-page Element

loader-registry

The `loader-registry` element defines a list of loader implementations that will be used to dynamically enable/disable and hide/show both the navigation pages and navigation tabs. For more information see Section 6.10.3, *Dynamic Support*.

Element	Description
<code>loader</code>	<i>1..n.</i> Defines one or more loader implementations that will be used to dynamically set the visibility and enabled state of the navigation pages and navigation tabs. See Section 6.10.2.5, <i>loader</i> for more information.

Table 6.41 Supported Child Elements of the loader-registry Element

loader

The `loader` element defines a single loader implementation that will dynamically set the state of the navigation pages and navigation tabs. For more information see Section 6.10.3, *Dynamic Support*.

Attribute	Description
<code>class</code>	<i>Mandatory.</i> The fully qualified class name of an implementation of the <code>curam.util.tab.impl.DynamicNavStateLoader</code> interface.

Table 6.42 Attributes of the loader Element

6.10.3 Dynamic Support

The tab navigation bar and page group navigation bar support the ability to dynamically enable/disable and hide/show navigation tabs and navigation pages. This feature is supported using a combination of the `dynamic` attribute of the `navigation-page` and `navigation-group` elements, the `loader-registry` element and a Java loader implementation.

The Java loader implementation registered in the menu configuration will be called when the tab is first loaded and based on the refresh options configured for a tab. The refresh options are configured in the tab configuration file (`.tab`). See Section 6.8.2.6, *tab-refresh* for more information.

A navigation tab and navigation page can be specified as dynamic in the navigation configuration file (`.nav`) by adding `dynamic="true"` to the relevant `navigation-page` or `navigation-group` elements.

Where a `dynamic` attribute is set, a `loader-registry` is then required and should define the fully qualified classname which implements the `curam.util.tab.impl.DynamicNavStateLoader` interface.

The `DynamicNavStateLoader` interface requires one method, `loadNavState`, to be implemented. The `loadMenuState` method is passed the following parameters:

- a list of `navigation-group` and `navigation-page` identifiers
- a set of name-value page parameters pairs

The loader implementation must decide which items to disable or hide. The method returns an object that represents the state of the navigation tabs and navigation pages. A state must be set for all identifiers in the list. For more information on this interface, consult the Java Documentation.



Note

The list of navigation identifiers passed to the `loadNavState` method are only those that have been identified as dynamic by the `dynamic` attribute on the `navigation-page` or `navigation-group` elements.

In addition, a `navigation-page` and `navigation-group` element cannot use the same identifier. The identifiers must be unique for all elements within the file.

6.10.4 Example

Example 6.8, *SimpleNavigation.nav* details an example tab navigation configuration file, which would be stored in a file called `SimpleNavigation.nav`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nc:navigation
  id="SimpleNavigation"

  <nc:loader-registry>
    <nc:loader class="some.pkg.SimpleNavStateLoader" />
  </nc:loader-registry>

  <nc:nodes>
    <nc:navigation-page id="Home"
      page-id="Home"
      title="Home.Title" />

    <nc:navigation-group id="Background"
      title="Background.Title">
      <nc:navigation-page id="Addresses"
        page-id="ParticipantAddressList"
        title="Addresses.Title" />
      <nc:navigation-page id="PhoneNumbers"
        page-id="ParticipantPhoneNumbers"
        title="Phone.Title" />
    </nc:navigation-group>

    <nc:navigation-page id="Identity"
      title="Identity.Title"
      page-id="ParticipantIdentity"
      dynamic="true" />
  </nc:nodes>
</nc:navigation>
```

Example 6.8 SimpleNavigation.nav

The `SimpleNavigation.nav` should have a corresponding `SimpleNavigation.properties` file, which details the localizable content. For example:

```
Home.Title=Home
Background.Title=Background
Addresses.Title=Addresses
Phone.Title=Phone Numbers
Identity.Title=Identity
```

6.11 Opening Tabs and Sections

6.11.1 Introduction

There are a number of ways to trigger opening a new section or tab.

- A section can be opened directly by selecting the relevant section tab control
- A tab can be opened directly by selecting the relevant tab control.
- Any link in the application has the potential to open a new tab.
- A section can be opened when a new tab is opened that is associated with a section other than the current section.

Opening a section or tab by selecting the relevant tab control is straightforward. To open a tab that is already open, but not in focus, the tab control is selected and focus is given to the tab.

Opening a section by selecting the relevant section tab control will give focus to that section. Any tabs already open in that section will then be accessible.

When a section is opened (directly) for the first time, it may contain no tabs or may result in the automatic opening of a default tab. This depends on the section configuration (see Section 6.6, *Sections*).

Opening a section or tab as a result of selecting a link is more complicated. When a link is selected, before the relevant UIM page is opened, the Cúram client will automatically determine if it should be opened in a new tab and if that tab should be opened in a new section. This is determined based a number of factors that will be detailed in the following sections.

6.11.2 Links

One of the actions that can trigger opening a new tab or new section is selecting a link to a UIM page. There are many different ways in the Cúram application to open a UIM page and many different contexts in which a UIM can be displayed.

A UIM page can be displayed in the following areas of an application:

- A content area
- A tab context panel
- A tab smart panel
- A modal dialog
- A list dropdown panel

A UIM page in any of these contexts can define links to another UIM page. There are different types of links:

- Page level actions menu (content area only)
- Modal button bar (modal dialog only)
- Buttons
- Hyperlinked text
- List actions menu

In addition to links on a UIM page, a UIM page can be opened via the following actions:

- Selecting an entry in the tab actions menu
- Selecting a link in the section shortcut panel
- Selecting a navigation bar tab
- Selecting a page group navigation bar entry

For more information on all the different types of action controls that can be defined in a UIM page, consult Chapter 5, *UIM Reference*. For the purposes of this section, selecting a link will apply to any action that can open a new UIM page.

6.11.3 Page to Tab Associations

A page is associated with tab based on the navigation configuration for the tab. The navigation for a tab is configured using the `navigation` element in the tab configuration file (`.tab`) and also, if defined, the navigation configuration file (`.nav`). See Section 6.10, *Tab Navigation* and Section 6.8.2.4, *navigation* for more information.

Where no tab navigation is defined for a tab, the `navigation` element defines a single UIM page (via the `page-id` attribute) that will result in opening the tab. A link to this page will open it in the relevant tab.

Where tab navigation is defined, any UIM page listed using a `page-id` attribute in the navigation configuration file (`.nav`) is considered to be associated with the tab. This means that a link to any of these referenced UIM pages will result in opening the relevant tab.

The page to tab association must be unique. This means that a page can be referenced only once by the navigation configuration for a tab. As a result, a navigation configuration cannot be re-used across multiple tabs.

There are a number of exceptions to this rule, but they are limited:

- The same UIM page can be referenced by more than one navigation configuration file (`.nav`), where the page is only ever linked-to from within the context of the tab.

This means that any links to the UIM page are always within the same tab. For example, a Notes UIM page is referenced by both the Person and Employer tabs. The only link to the Notes UIM page is from the page group navigation bar. The Notes UIM page is never referenced from a shortcut panel or linked by a UIM page that is not displayed within the context of the Employer or Person tabs.

- The same UIM page can be referenced by more than one navigation configuration for a tab, where the tabs are included in different application configurations (`.app`).
- A navigation configuration file (`.nav`) can be reused by two tabs, where the tabs are included in two different application configurations (`.app`).



Resolve Pages

It is recommended against using resolve pages³ in a navigation configuration. The reason for this is based on how the Cúram client application handles resolve pages and opening new tabs.

When a link to a resolve page is selected, the Cúram client recognises it is a resolve page and executes the content of the `JSP_SCRIPTLET`. The resulting UIM page that the `JSP_SCRIPTLET` redirects to is then used to determine what tab the page should be opened in.

6.11.4 Tab to Section Associations

A tab is associated with a section by listing it using the `tab` element in the section configuration file (`.sec`).

When a new tab is opened as a result of selecting a link, the tab is opened in the associated section and focus is given to that section and tab.

6.11.5 Page Parameters

The client determines if a new tab is opened based on the page to tab to section association. In addition, existing opens tabs and values of the parameters passed to a tab are taken into consideration.

Two instances of the same tab can be opened, where each instance is identified by the page parameters that have been provided. For example, James Smith and Linda Smith are uniquely identified by their concern role ID. The concern role ID is defined as a page parameter for the Person tab.

When a link to James Smith is selected, a new tab is opened showing the details for James Smith. A subsequent link to Linda Smith is selected and a new instance of the same tab configuration is opened, displaying Linda Smiths details.

When a link is selected, the Cúram client application automatically determ-

ines what tab, and section, it is associated with. It then compares this information, along with the page parameters to determine what action to take.

The rules for opening tabs are detailed in Table 6.43, *Tab Opening Rules*.



Note

The parameters passed when a link is selected must match the names of the page parameters defined in the tab configuration file.

Where not all required page parameters are provided, the behavior of those tabs within the application is not guaranteed. Any extra parameters provided will be ignored and not passed to the tab.

Page to Tab Association	Page Parameter Values	Action
Page maps to current tab	Match	Page opens in current tab
Page maps to current tab	Differ	Page opens in new instance of tab
Page maps to existing open tab	Differ	Page opens in a new instance of existing tab
Page maps to existing open tab	Match	Page opens in existing tab
Page maps to new, unopened tab	N/A	Page opens in new tab

Table 6.43 Tab Opening Rules



Limitations

There are a number of limitations and notes to be aware of when designing UIM pages to open in new tabs.

- Links in a modal dialog obey dialog rules first and only obey the rules for opening a tab when the dialog is closing.
- A link defined to open a modal dialog ignores the tab rules.
- Links in a tab navigation bar and page group navigation bar will always open within the context of the current tab.
- A submit link within the content area cannot open a new tab, even if the UIM page is configured to be associated with a different tab.
- If a UIM page is configured to be associated with a tab then the same page cannot be used as `INLINE_PAGE` in expandable lists.

Notes

¹Consult Chapter 5, *UIM Reference* for more information on User Interface Meta-data.

²A modal dialog is a UIM page opened in a new window, where the parent window cannot be accessed while it is open. Consult Section 5.9.22.3.1, *Using Modal Dialogs* for more information.

³ A resolve page is a specific type of UIM page that contains only a `JSP_SCRIPTLET` element. See Section 5.9.20, *JSP_SCRIPTLET* for more information.

Chapter 7

Session Management

7.1 Objective

This chapter provides detailed information on how browser sessions are handled in the Cúram application.

7.2 Prerequisites

You should be familiar with the basic concepts of Cúram CDEJ development (see Chapter 2, *Concepts*) and web application development.

7.3 Introduction

The current set of open tabs for a particular user is restored each time the user logs out of the application and logs back in. In addition, if the browser is refreshed (e.g. using the **F5** button), the currently open tabs are also restored.

The browser session plays an important role in the expected behavior when restoring tabs, and this chapter will detail how browser sessions interact with the restoration of tabs. In addition, a number of configuration options for the tab restoration feature are detailed.

7.4 Session Basics

A browser session can be defined as a continuous period of user activity in the web browser, where successive events are separated by no more than 30 minutes. The following listing shows the common examples of when a Cúram browser session is started or finished.

- A session starts when a user first logs into the application.

- As long as the user is actively using the browser, the session remains active.

If the browser is left inactive for a period of time, the session will timeout. In this case, the user will be required to log back in and a new session is started.

The default timeout is 30 minutes, but this can be configured using the application server's configuration settings. See the *Cúram Deployment Guides* for more information on application server configuration.

- The user can explicitly logout, using the logout link in the application banner. The session is terminated in this case and logging back in will start a new one.
- The browser is shutdown and a new browser instance is started. In this case, a new session is started and the user will be required to log in.

7.5 Tab Restoration

The list of currently open tabs is stored temporarily in the web tier, associated with the browser session, and more permanently on the database so that it can be restored after a user logs out of the application.

The data is persisted from the web tier to the database intermittently. As a result, there are cases where the last few changes to the open tabs may not be restored when the user logs in. This is most likely to happen where the session times out or the browser is restarted.

The behavior of tab restoration is different depending on whether it was the result of a browser refresh (**F5**) or the start of a new session (i.e. the user has logged in).

- `Browser Refresh`

If the browser is refreshed, tabs are restored to their current state from the web tier session data. No tab changes will be lost.

- The tab that was last selected in the selected section will remain the selected tab.
- The selected tab in other sections will revert to the first tab in those sections.
- The expanded or collapsed states of the shortcut panel, smart panel and page contents are not restored.

- `New Session`

When a new session starts, usually requiring the user to login, the tabs are restored to their current state using the session data stored on the database.

- The “Home” tab is restored as the selected tab.

- The selected tab in other sections will revert to the first tab in those sections.
- The expanded or collapsed states of the shortcut panel, smart panel and page contents are not restored.
- If no previous tab session data is available, only the “Home” tab is opened.



Note

See Section 2.10, *Direct Browsing* for a special case of tab restoration, where pages are directly accessed through the browser navigation bar.

7.6 Configuration

Each time a new tab is opened, a tab is closed or the content area of a tab is updated, the information is stored in the web tier. The tab session data is persisted from the web tier to the database intermittently. How often the data is persisted can be configured using the following options, which can be set in the `ApplicationConfiguration.properties` file.

- **tabSessionUpdateCountThreshold.** Specifies the number of tab session data updates that must be received before the data is persisted from the web tier to the database. Once the threshold is reached, the recent updates are written and counting starts again from zero until the threshold is reached. A value of one causes writes on every update. A value of zero (or a negative or invalid value) disables writing based on update counts. The default is every 10 updates.
- **tabSessionUpdatePeriodThreshold.** Specifies the number of seconds that must have elapsed since the last time session data was persisted from the web tier to the database before a new update will trigger another write. A value of zero (or a negative or invalid value) disables writing based on update periods. The default value is 120 seconds, or 2 minutes.

The properties work together based on which value is reached first. In other words, if the update count threshold (`tabSessionUpdateCountThreshold`) is not reached, but the update period threshold (`tabSessionUpdatePeriodThreshold`) has been reached, a write will occur, and vice versa.

If the update count threshold is set to one, the update period threshold is ignored. The reason for this is that writes will happen on every update, so there is no need to write based on a time period.



Note

Tab session data is persisted to the database when the user logs out, regardless of the value of the current update count and update peri-

od. The exception to this is if both the update count threshold and the update period threshold are set to zero.

Each user account has one persistent tab session database record for an application. The same user logging in to the application from different browser sessions will cause some interference and unpredictability in what data is persisted across sessions.

The interference and unpredictability of the persisted data, when multiple users are using the same login ID, is most likely encountered in a testing environment. It is recommended that the `tabSessionUpdatePeriodThreshold` and `tabSessionUpdateCountThreshold` properties are set to zero for testing environments to prevent this. Setting both properties to zero ensures that the tab session data is only persisted for the length of a browser session and not across sessions, i.e. login and logout.

It is also recommended that these settings are used where an "external" application is deployed and the external users all share the same generic user account.

7.7 Limitations

The tab session data records a limited number of tabs. The limit imposed relates to the total size of the tab session data and is approximately 70-80 tabs. Once this limit has been exceeded, tab session data is maintained only in the web tier and is no longer written to the database.

Restoration of the tab session when the browser is refreshed is not affected. However, if a user logs out with more tabs open than can be recorded for a session, only the state of the tabs at the time the limit was first exceeded will be restored.

Closing tabs will reduce the size of the tab session data and writing to the database will then resume as normal.

7.8 Browser Specific Session Management

The version of the browser used can have an effect on when new sessions are started and when they are shared. Two browser instances that share the same session will result in the same set of open tabs displayed in both instances. This can cause similar interference and unpredictability of the persisted data as with two users using the same login ID from different machines.



Example Session Issue

A user logs into the Cúram application in one browser instance. They then open a new browser tab, which is sharing the same session. From here, they directly access the Cúram login page and login as a different user.

In this situation, they are still logged in as the original user and will

see the tabs that were open in the original browser tab.

Within the same browser session, a user must always logout to end the session and be able to login as a new user.

The most common browsers supported are Internet Explorer 7 and Internet Explorer 8 and they share sessions across browser instances in different ways:

- *Internet Explorer 7*

If a new browser instance, or browser tab, is opened in Internet Explorer 7 using the *File→New Tab* or *File→New Window* options, from an existing browser instance, the session is shared across the instances. This means that if the user was already logged into the Cúram application in the original browser instance, they will also be logged into Cúram in the new tab or window.

If a new browser instance is started using the Internet Explorer link in the *Start* menu, the sessions are not shared and the user must login again to Cúram.

- *Internet Explorer 8*

Sessions are always shared in Internet Explorer 8, no matter where the browser instance or tab was started from. This is the default behavior.

To start a new instance of the browser that does not share the existing session, the *File→New Session* option should be used.

For further information on browser specific behavior, please consult the relevant online documentation.

Chapter 8

Domain Specific Controls

8.1 Objective

This chapter describes the domain specific controls that are provided by the *Cúram CDEJ*. These domain specific controls are employed to provide a more sophisticated interface for user information than the standard set of HTML controls.

8.2 Prerequisites

The reader should understand how to model their *Cúram* application, choosing appropriate domains for more complicated data. Knowledge of client development within the *Cúram* application is also necessary.

8.3 Introduction

Examples of domains requiring sophisticated controls include: dates, date-times, the meeting view and the rules decision tree. Any UIM page containing a server access bean with fields of this nature will have a web page generated containing a custom control appropriate to the type. For example, when a server bean contains the `CALENDAR_XML_STRING` domain, a calendar will be generated which expects server information in a particular XML format. Each of the following sections details the custom controls translated for particular domains.

8.4 Dates

Dates are mapped to the `SVR_DATE` domain. Any server access bean containing fields of this type will display a date selector to the user for data input. These selectors are HTML text fields with an adjacent pop-up icon which causes a pop-up menu to be displayed allowing the user to select a

date or date time with ease. Note that this functionality is based on JavaScript and it is important that the user have JavaScript enabled in their browser for this selector to work. The appearance of the date selector pop-up can be altered by overriding its dedicated cascading stylesheet. See Section 3.12.11, *Cascading Stylesheets* for more details. The out-of-the-box date date pop-up dialog has three input controls; a drop-down field for the month, a text input field for the year, and the days of the month are displayed so that a day can be selected. When the day of the month is selected, this will populate the date field.

The date format string associated with date format validations are customizable in the file `CDEJResources.properties` and defined by the property `curam.validation.calendar.dateFormat`:

```
curam.validation.calendar.dateFormat=M/dd/yyyy
```

Example 8.1 Customizing the Date Format

If this value is not set, the date format string will default to the date format setting specified in the `ApplicationConfiguration.properties` file.

8.5 Date-Times

Date-times are mapped to the `SVR_DATETIME` domain. Any server access bean containing fields of this type will display a date selector (see previous section) next to a time entry field.

Similar to the date selector, the pop-up here requires JavaScript to function correctly. It is important that the user have JavaScript enabled in their browser for these selectors to work.

There is an additional control for entering time as hours and minutes. It is displayed as two side-by-side drop down lists for selecting the hour and minute values.

When the `CURAM_TIME` domain (a descendant of the `SVR_DATETIME` domain) is used, the date input field will not be displayed.

The date time format string associated with date time format validations are customizable in the file `CDEJResources.properties` and defined by the property `curam.validation.calendar.dateTimeFormat`:

```
curam.validation.calendar.dateTimeFormat=HH:mm
```

Example 8.2 Customizing the Date Time Format

If this value is not set, the date time format string will default to `HH mm ss`.

8.5.1 Representing time-only values

As has been described above Curam has a base type for "date-only" and "date-time" values, however there is no specific base type for "time-only" values.

A `CURAM_TIME` domain is provided in out-of-the-box Curam and this is used by the client infrastructure to display a corresponding time only widget, in addition to performing certain processing when parsing and formatting values based on this domain. However, the underlying data representation is the same as for `SVR_DATETIME` and when working with time-only domains the corresponding server-side code must completely ignore the date part of the value.

Because time-only domains are based on the `SVR_DATETIME` domain, it should be noted that the default values will also be the same. The "zero date time" of 0001-01-01 00:00:00 is the value sent to the server if the field is left blank. If the field is set to 00:00, then 00:00 time value of today's date is sent.

The time input field rendered for `CURAM_TIME` domain is an editable combo box as the example below shows. The combo box contains selectable time values for every 30 minutes. The exact time value can also be entered directly in the field.

The values to be selected are in the application-wide format set in `ApplicationConfiguration.properties`, including AM/PM for the 12 hour display. A manually typed value should follow the same format.

8.5.2 Customizing the Time Format

The application-wide time format setting can be changed by setting or modifying the `timeformat` and `timeseparator` values in the `ApplicationConfiguration.properties` file as described in Section 3.11.2, *Configuring the Application*.

8.6 Frequency Pattern Selector

Frequency patterns are mapped to the `FREQUENCY_PATTERN` domain. Any server access bean containing fields of this type will display a frequency pattern selector to the user for data input. These selectors are non-editable HTML text fields with an adjacent pop-up icon which causes a pop-up menu to be displayed allowing the user to select a frequency pattern with ease. Note that this functionality is based on JavaScript and it is important that the user have JavaScript enabled in their browser for this selector to work. The appearance of the frequency pattern selector pop-up can be altered by overriding its dedicated cascading stylesheet. See Section 3.12.11, *Cascading Stylesheets* for more details. The figure below shows the frequency pattern selector.

<input type="radio"/> Daily	<input type="radio"/> Every <input type="text" value="1"/> day(s) <input type="radio"/> Every weekday
<input checked="" type="radio"/> Weekly	Recur every <input type="text" value="80"/> week(s) on: <input type="checkbox"/> Monday <input type="checkbox"/> Tuesday <input type="checkbox"/> Wednesday <input checked="" type="checkbox"/> Thursday <input type="checkbox"/> Friday <input type="checkbox"/> Saturday <input checked="" type="checkbox"/> Sunday
<input type="radio"/> Monthly	<input type="radio"/> day <input type="text" value="1"/> of every <input type="text" value="1"/> month(s) <input type="radio"/> the <input type="text" value="first"/> day of every <input type="text" value="1"/> month(s)
<input type="radio"/> Bi-monthly	<input type="radio"/> day(s) <input type="text" value="1"/> and <input type="text" value="1"/> of every month <input type="radio"/> the <input type="text" value="first"/> and <input type="text" value="second"/> <input type="text" value="Monday"/> of every month
<input type="radio"/> Yearly	<input type="radio"/> Every <input type="text" value="January"/> <input type="text" value="1"/> <input type="radio"/> the <input type="text" value="first"/> day of every <input type="text" value="January"/>
<input type="button" value="Ok"/> <input type="button" value="Cancel"/>	

Figure 8.1 Frequency Pattern Selector Pop-up

It is worth noting that the frequency pattern text selected varies in length, depending on the pattern selected. This makes the display of the selected pattern prone to re-sizing and wrapping, depending on the layout of the UIM page and the display space available.

8.7 Selection Lists

Within the Cúram application, the use of the standard HTML selection list i.e. the `select` element is supported. Selection lists will truncate long data strings in order to preserve the correct page layout. To combat this, the data's full value is available as a tooltip for each item in the list. The list can be populated with data in a number of ways as described in the following sections.

8.7.1 Populated from a Code-Table

If a `FIELD` has a target connection mapped to a property based on a code-table domain, a drop-down selection list will be displayed containing all code-table entries that are marked as “enabled”. The entries will be sorted alphabetically according to their code descriptions. This can be overridden by setting the “sort order” of each entry. Consult the *Cúram Server Developers Guide* for full details on creating code-tables in a Cúram application.

When the selection list is displayed the initially selected item is evaluated as follows:

1. The code value specified by the source connection of the field.
2. The default code of the code-table if the `FIELD` element's

USE_DEFAULT attribute is not set to false.

3. The first item in the selection list, if no default code is defined or the default code is marked as “disabled”.
4. Blank, if the FIELD element's USE_DEFAULT attribute is set to false.

A drop-down selection list can also be displayed as a scrollable selection list where a number of entries are initially displayed instead of just one. To do this simply set the HEIGHT attribute of the FIELD element to a value greater than 1.

8.7.2 Populated from Server Interface Properties

Data retrieved through server interface properties can also be used to populate a selection list. The INITIAL connection end-point is used in this case. The following are examples of a selection list on an insert and a modify page.

```
<FIELD LABEL="Field.Label">
  <CONNECT>
    <INITIAL NAME="DISPLAY" PROPERTY="personName"
      HIDDEN_PROPERTY="personID" />
  </CONNECT>
  <CONNECT>
    <TARGET NAME="ACTION" PROPERTY="personID" />
  </CONNECT>
</FIELD>
```

Example 8.3 Selection List on an Insert Page

In this example the field has an INITIAL connection end-point to populate the selection list and a TARGET connection end-point to specify what field the selected value should be mapped to. The PROPERTY attribute of the INITIAL connection end-point is the list of values you want the user to see in the selection list. When the list is displayed, the first item in the list will initially be selected. The HIDDEN_PROPERTY attribute specifies a list of corresponding values, when selected, will be mapped to the property specified in the TARGET connection end-point. The target property is a single field, *not* a list. In this example a list of people's names will be displayed but it is the selected person's unique ID that will be mapped to the target property. In certain circumstances the set of values visible to the user may also be what you want mapped to the target property. In this case do not use the HIDDEN_PROPERTY attribute.

The following example shows the same selection list, but used on a modify page. The only difference is a SOURCE connection end-point is used to specify what is selected in the list when the page is first displayed.

```
<FIELD LABEL="Field.Label">
  <CONNECT>
    <INITIAL NAME="DISPLAY" PROPERTY="personName"
      HIDDEN_PROPERTY="personID" />
  </CONNECT>
```

```

<CONNECT>
  <SOURCE NAME="DISPLAY" PROPERTY="sourcePersonID" />
</CONNECT>
<CONNECT>
  <TARGET NAME="ACTION" PROPERTY="personID" />
</CONNECT>
</FIELD>

```

Example 8.4 Selection List on a Modify Page

8.7.3 Drop-down, Scrollable and Checkboxed List types

Drop-down and Scrollable List

The selection list can be displayed as a drop-down list or as a scrollable selection list with a number of entries visible. A drop-down selection list is displayed by default. To change this to a scrollable selection list set the `HEIGHT` attribute of the `FIELD` element to a value greater than 1. The appearance of a selection list differs from a drop-down list in two noticeable ways. For a drop-down list only the default value is displayed and all the other selectable values are displayed only when the drop down arrow is selected. Additionally the drop-down list is not scrollable. However, a scrollable selection list does not have the drop-down arrow, a subset of the values are initially displayed - the size of the subset is dependant on the value of the `HEIGHT` that is set. This list has a scrollbar which can be used to scroll the list, and view and select the remainder of the selectable values.

Checkboxed List

Checkboxed selection list offers an alternative method of selecting individual entries, in this case using the check box control. This variation will be used if `CONTROL` attribute is set to `CHECKBOXED_LIST`. It is just an alternative way of representation, so everything else applicable to Scrollable List applies for Checkboxed List without change.

8.7.4 Adding an Empty Entry to a List for Non-Mandatory Fields

Browsers will select the first item in a selection list by default if no item is marked as selected. In certain cases you may not want to “suggest” a value to the user. A blank entry would be more suitable. Set the `USE_BLANK` attribute of the `FIELD` element to `true` to add a blank entry as the first item on the selection list.

8.7.5 Enabling Multiple Selection

Browsers allow multiple items to be selected in a selection list. To enable this first use a scrollable list as described above (you cannot select multiple items from a drop-down list). Then add the following to the `curam-`

config.xml file.

```
<MULTIPLE_SELECT>
  <DOMAIN_NAME="MY_DOMAIN" MULTIPLE="true" />
</MULTIPLE_SELECT>a
```

Example 8.5 Enabling multiple selection in curam-config.xml

For each domain which you want to enable multiple selection add a `DOMAIN` child element to the `MULTIPLE_SELECT` element. If a `FIELD` has a target connection which is based on a domain listed in the `MULTIPLE_SELECT` element, multiple selection will be enabled. When the form containing the selection list is submitted, the selected values will be packaged into a tab-delimited string. Therefore the target property must be based on a string domain. The same way, the source property in this case is also expected in the form of a tab-separated string of values to be selected initially (the values should match some of those specified via `HIDDEN_PROPERTY`).

8.7.6 Transfer List Widget

Overview

The Transfer List widget is a control used to facilitate multiple selections for a user (i.e. it is used as an alternative to a regular list which has multiple selection enabled). It consists of two HTML select controls placed side by side. The left control contains the items from which selections can be made (see See Section 8.7.3, *Drop-down, Scrollable and Checkboxed List types* for more details on selection lists.), the one to the right displays already selected items. Four buttons between the lists allow for selecting/de-selecting individual or all items (transferring them from one list to another and back as required).

Configuration

The Transfer List widget is displayed instead of a regular HTML multiple selection control when configured in one of the two ways described below. In order for all multiple selection controls in an application to be displayed as Transfer List widgets, `curam-config.xml` should contain the `TRANSFER_LISTS_MODE` element with its value is set to `true`. Alternatively, individual multiple select controls might be configured to be displayed that way by setting the `CONTROL` attribute on the appropriate UIM `FIELD` to be `TRANSFER_LIST`. This setting is applicable just for fields rendered as multiple selection controls on the resulting UIM page and will be ignored in any other case.

The Transfer List widget requires the same data and the same configuration for enabling multiple selection as a regular selection list.

8.8 User Preferences Editor

The User preferences editor allows a user to edit a user preference value for use anywhere within the application. For details on the definition of user preferences please consult the *Cúram Server Developers Guide*.

The editor may be accessed from the taskbar by clicking the preferences button. On clicking this button a popup window should be displayed with a list of all visible user preferences. Those preferences that are editable will appear as either a text field, radio buttons or a drop-down menu, depending on the type.

If the user wishes, they may edit the value of a preference and save the value using the `Submit Changes` link. When the user returns to the editor the updated values will appear. Any changes to user preferences using the editor will be applied immediately.

To return the values to those that were originally defined, the user should click the `Reset to Default` link. Selecting either of these buttons will close the popup window.

8.9 Rules Trees

8.9.1 Introduction

The `RESULT_TEXT` domain contains information about the success or failure of a particular claim against a set of rules. When the server supplies this information it is translated into a tree view displaying all rules. Figure 8.2, *Default Rules Tree View* below shows the default rules tree view.

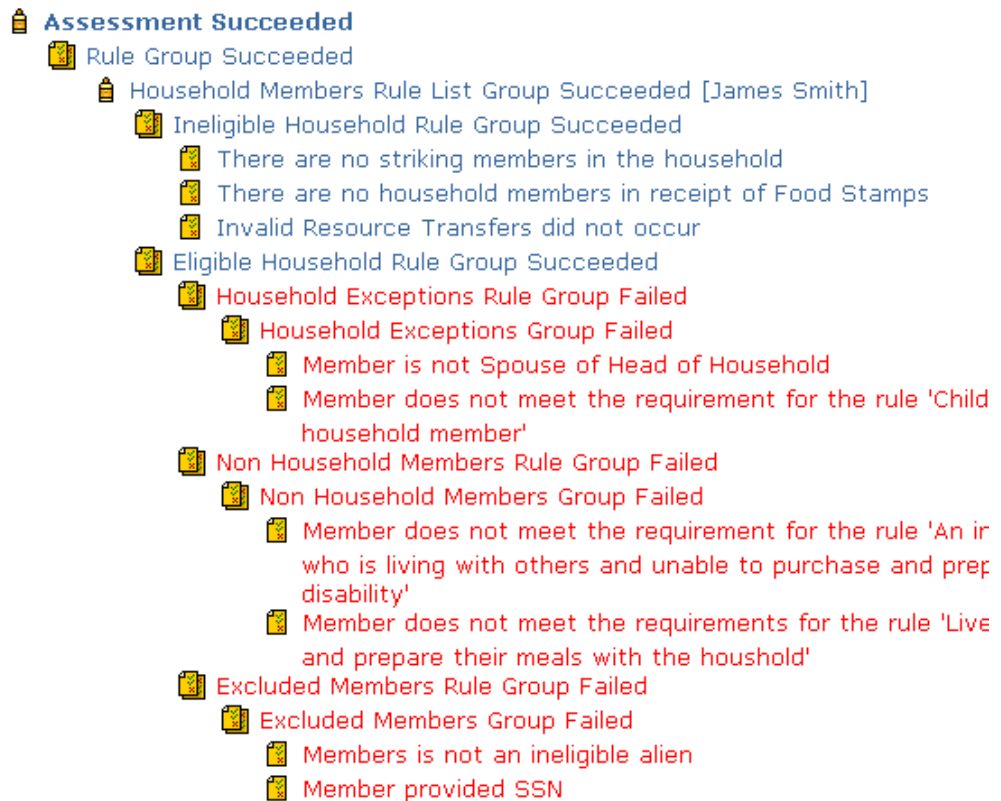


Figure 8.2 Default Rules Tree View

The `RULES_DEFINITION` domain also produces a rules tree, in this case displayed with the rules editor. For more details on the rules editor see Section 8.9.7, *Rules Editor*.

It is possible to use the `FIELD` element's `CONTROL` attribute to change the format of the rules display. The following sections will describe the various options for this attribute. Furthermore, the `FIELD` element's `CONFIG` attribute can be used to configure these rules trees.

Behavior of Summary and Highlight-On-Failure Rules Flags

The `summary-flag` has no effect in this view. All rules items are displayed.

The `highlight-on-failure` flag causes failed rules to be highlighted in a different color to those that have succeeded.

8.9.2 Default Rules View

The default rules view of the rules tree (Figure 8.2, *Default Rules Tree View*), specified by setting the `CONTROL` attribute of the `FIELD` element to `DEFAULT`, shows data in an expanded tree view using standard HTML. This view should be visible in most standard web browsers. However, as the rules result is often quite verbose, the resulting output can be confusing to the viewer of your web page.

8.9.3 Summary Rules View

To display a summary rules view, set the `CONTROL` attribute of the `FIELD` element to `SUMMARY`. The view of this tree is very similar to the default rules tree view which can be seen in Figure 8.2, *Default Rules Tree View* except that the details about why a rule failed or succeeded are not displayed in the tree.

Any rules, regardless of type, marked as summary items are displayed. The following section, Section 8.9.4, *Failed Rules View*, describes a similar view that only displays rules items whose type is explicitly set to `rule`. This view can be configured in the same manner as the dynamic rules view mentioned below. See Section 8.9.5, *Dynamic Rules View*.

8.9.4 Failed Rules View

To display a failed rules view, set the `CONTROL` attribute of the `FIELD` element to `FAILURE`. This view is similar in layout to the previously mentioned summary view. See Section 8.9.3, *Summary Rules View*

Any rules whose type is `rule` (and not `objective` or `rule_group` for example) and are marked as summary items are displayed. This view can be configured in the same manner as the dynamic rules view mentioned below. See Section 8.9.5, *Dynamic Rules View*

8.9.5 Dynamic Rules View

When the `CONTROL` attribute is set to `DYNAMIC`, this causes an expanding/contracting version of the decision to be displayed instead of a static tree. In this view the entire tree is not displayed. The view is “compressed” into multiple trees for each rules-item that has failed coupled with the “summary” flag on the item. See Section 8.9.5.1, *Behavior of Summary and Highlight-On-Failure Indicator* for more details on the summary flag. This is accomplished using scalable vector graphics (SVG) content displayed in the Adobe® *SVG Viewer* instead of HTML. Refer to the *Cúram v6 Supported Prerequisites* document to see the supported version of this Web Browser Plugin.

Although the dynamic view requires an extra browser plug-in, it provides the user with a much more comprehensive and interactive view of the rules data. The rules tree is more comprehensively organized with a supplementary conjunction text displayed next to the rules. The following image illustrates the dynamic rules view.

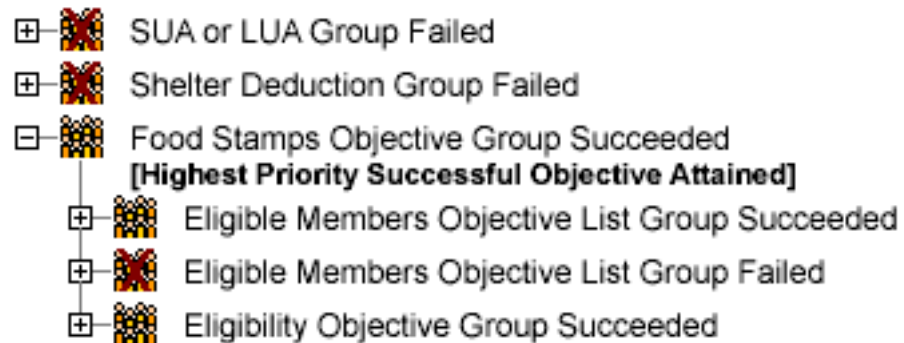


Figure 8.3 Dynamic Rules View

There is no need to set a HEIGHT or WIDTH as the rules window resizes itself automatically. The developer is limited to two dynamic rules windows per page.

Localization of the text to display within the viewer is accomplished through JavaScript property files as described in Section 4.8, *JavaScript Externalized Strings*. The name of these JavaScript property files should be SVGText. For example, SVGText.js_es.properties would be the name of the Spanish language version of SVGText.js.properties file.

All style information related to the dynamic rules widgets is held in a separate file called curam_svg.css. For further details see Section 3.12.11, *Cascading Stylesheets*.

The developer can configure the rules tree using an XML configuration file. For all rules widgets based on the RESULT_TEXT domain this configuration is read from RulesDecisionConfig.xml. A version of this file should be in your components directory. This XML configuration file is merged during the build process in a similar method to other XML configuration files.

The CONFIG attribute of the FIELD displaying rules is used to specify an ID matching a CONFIG element in the RulesDecisionConfig.xml file. The following is a sample of a RulesDecisionConfig.xml file:

```
<RULES-CONFIG DEFAULT="default-config">
  <CONFIG ID="default-config" HYPERLINK-TEXT="false">
    <TYPE NAME="PRODUCT"
      SUCCESS-ICON="Images/product-16x16.gif"
      FAILURE-ICON="Images/productFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="ASSESSMENT"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="SUBRULESET"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="OBJECTIVE_GROUP"
```

```

        SUCCESS-ICON="Images/default-16x16.gif"
        FAILURE-ICON="Images/defaultFail.gif"
        EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="OBJECTIVE_LIST_GROUP"
        SUCCESS-ICON="Images/default-16x16.gif"
        FAILURE-ICON="Images/defaultFail.gif"
        EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="OBJECTIVE"
        SUCCESS-ICON="Images/default-16x16.gif"
        FAILURE-ICON="Images/defaultFail.gif"
        EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="RULE_GROUP"
        SUCCESS-ICON="Images/default-16x16.gif"
        FAILURE-ICON="Images/defaultFail.gif"
        EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="RULE_LIST_GROUP"
        SUCCESS-ICON="Images/rule-group-16x16.gif"
        FAILURE-ICON="Images/ruleGroupFail.gif"
        EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="RULE"
        SUCCESS-ICON="Images/rule-16x16.gif"
        FAILURE-ICON="Images/ruleFail.gif" />
</CONFIG>
<CONFIG ID="Rules.Config.Core"
    HYPERLINK-TEXT="true"
    OPEN-NODE-PARAM="openNode"
    DECISION-ID-SOURCE="source-Decision-ID"
    DECISION-ID-TARGET="decision-ID">
    <TYPE NAME="PRODUCT" EDIT-PAGE="RulesResult" />
    <TYPE NAME="ASSESSMENT" EDIT-PAGE="RulesResult" />
    <TYPE NAME="SUBRULESET" EDIT-PAGE="RulesResult" />
    <TYPE NAME="OBJECTIVE_GROUP" EDIT-PAGE="RulesResult" />
    <TYPE NAME="OBJECTIVE_LIST_GROUP" EDIT-PAGE="RulesResult" />
    <TYPE NAME="OBJECTIVE" EDIT-PAGE="RulesResult" />
    <TYPE NAME="RULE_GROUP" />
    <TYPE NAME="RULE_LIST_GROUP" EDIT-PAGE="RulesResult" />
    <TYPE NAME="RULE" EDIT-PAGE="RulesResult" />
</CONFIG>
</RULES-CONFIG>

```

Example 8.6 Sample `RulesDecisionConfig.xml` File

Note that the `RULES-CONFIG` root element only contains the `DEFAULT` attribute. This attribute is mandatory and should match an `ID` attribute value on a `CONFIG` element in this document. The default configuration contains the icon information as well as the default nodes to link to if no configuration is required for a widget. These are covered by the `SUCCESS-ICON`, `FAILURE-ICON`, and `EDIT-PAGE` attributes respectively.

Each `CONFIG` element has a `HYPERLINK-TEXT` attribute which is used to specify whether the text next to a rules node in the widget is also to be used as a hyperlink to the link page set by the `EDIT-PAGE` for the `TYPE` in question.

Note that the `CONFIG` with the `ID` of value of `Rules.Config.Core` has the optional attribute `OPEN-NODE-PARAM`. This attribute is the name of a page parameter whose value is the `ID` of a node to open when the page is loaded. This configuration file is also used for configuration of the dynamic full tree rules view described in the next section.

The `CONFIG` attributes `DECISION-ID-SOURCE` and `DECISION-ID-TARGET` are used to identify a page parameter whose value will be the source for a new parameter (named by the `DECISION-ID-TARGET`) ap-

pended to each link on the widget. The above example will look for a page parameter called source-Decision-ID and pass on its value as a parameter to any links on the widget. This new value will be identified by a parameter named decision-ID.

The decision ID parameter may also be sourced from a field on a server bean instead of from a page parameter. This is achieved by adding DECISION-ID-SOURCE-BEAN and DECISION-ID-SOURCE-FIELD attributes to the CONFIG element instead of a DECISION-ID-SOURCE attribute. A validation error is thrown if all three are present. The DECISION-ID-SOURCE attribute should be the name of a bean on the page and the DECISION-ID-SOURCE-FIELD attribute should be the full name of a field providing the decision ID value. The following is an example of this configuration:

```
<CONFIG ID="Decision.ID.Bean.Source"
  HYPERLINK-TEXT="true"
  OPEN-NODE-PARAM="openNode"
  DECISION-ID-TARGET="decision-ID"
  DECISION-ID-SOURCE-BEAN="DISPLAY"
  DECISION-ID-SOURCE-FIELD="dtls$decision-ID">
  <TYPE NAME="PRODUCT" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="ASSESSMENT" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="SUBRULESET" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="OBJECTIVE_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="OBJECTIVE_LIST_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="OBJECTIVE" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="RULE_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="RULE_LIST_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="RULE" EDIT-PAGE="RulesResult"/>
</CONFIG>
```

Example 8.7 Example of Decision ID Sourced from a Bean

Behavior of Summary and Highlight-On-Failure Indicator

The highlight-on-failure indicator on a rules item does not have any effect in this view.

If an item fails and is marked as a summary item, this item should only be displayed as a separate tree if no item along its parent path (i.e. any group that contains it) has failed and is marked as a summary item. Consider the following tree of rule groups and rules and note the result and summary attributes on each item. Note that this is purely for illustrative purposes and does not represent the data-format created by the Rules Engine.

```
<decision>
  <rules-item id="B" type="rule-group"
    result="success" summary="true">
    <rules-item id="C" type="rule"
      result="success" summary="false" />
    <rules-item id="D" type="rule"
      result="fail" summary="true" />
  </rules-item>
  <rules-item id="E" type="rule-group"
    result="fail" summary="true">
    <rules-item id="F" type="rule"
      result="fail" summary="false" />
    <rules-item id="G" type="rule"
      result="success" summary="false" />
  </rules-item>
</decision>
```

```

</rules-item>
<rules-item id="H" type="rule-group"
  result="success" summary="true">
  <rules-item id="I" type="rule"
    result="success" summary="true" />
  <rules-item id="J" type="rule"
    result="fail" summary="false" />
</rules-item>
</decision>

```

Example 8.8 Example of Rules Tree Items with Summary Flag

A rule that fails and is marked as "not a summary item" may still display as long as it is contained within another node that fails and has summary set to "true". A rule that fails and is marked as "not a summary item" will never display as the root of a tree in the dynamic rules view. So, the data above will result in separate “trees” as follows.

```

- D
- E
-- F
-- G

```

From the first rule-group “B”, only the item “D” is displayed because it has failed and is marked as a summary item. It appears as a single-node tree.

The rule-group “E” is marked as a summary item and it has failed, therefore it and all its child nodes are displayed no matter what the success\failure status or summary flag on the child nodes is.

The entire rule-group “H” is filtered out. “H” itself, and “I” have succeeded and will not be displayed. Although “J” has failed it is not marked as a summary item and therefore is not displayed.

8.9.6 Dynamic Full Tree Rules View

When the CONTROL attribute is set to DYNAMIC_FULL_TREE a view, similar in functionality to the dynamic rules view described in the previous section, is displayed. The main difference is that the entire rule set is displayed, similar to the default rules view, although the tree is interactive thus requiring the SVG viewer. There is no filtering of the display of rule groups in this view, potentially making it difficult to understand for someone who is not familiar with the rules engine. Configuration of this view is through the RulesDecisionConfig.xml file described in the previous section.

8.9.7 Rules Editor

The RULES_DEFINITION domain produces the rules editor. This control has a default HTML-only view or, if the FIELD's CONTROL attribute is set to DYNAMIC, an SVG view. See Section 8.9.2, *Default Rules View* and Section 8.9.5, *Dynamic Rules View* for more information.

This widget uses the CONFIG attribute to specify an ID attribute value matching the ID attribute value of a CONFIG element in the RulesEditorConfig.xml file. This XML configuration file is merged during the

build process in a similar method to other XML configuration files. The following is a sample of RulesEditorConfig.xml:

```
<RULES-CONFIG DEFAULT="DefaultConfig">
  <CONFIG ID="DefaultConfig" HYPERLINK-TEXT="true">
    <TYPE NAME="Product"
      SUCCESS-ICON="Images/product-16x16.gif"
      FAILURE-ICON="Images/productFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="Assessment"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="SubRuleSet"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="ObjectiveGroup"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="ObjectiveListGroup"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="Objective"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="SubRuleSetLink"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="RuleGroup"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="RuleListGroup"
      SUCCESS-ICON="Images/rule-group-16x16.gif"
      FAILURE-ICON="Images/ruleGroupFail.gif"
      EDIT-PAGE="RatesNewColumn" />
    <TYPE NAME="Rule"
      SUCCESS-ICON="Images/rule-16x16.gif"
      FAILURE-ICON="Images/ruleFail.gif" />
    <TYPE NAME="DataItemAssignment"
      SUCCESS-ICON="Images/default-16x16.gif"
      FAILURE-ICON="Images/defaultFail.gif"
      EDIT-PAGE="RatesNewColumn" />
  </CONFIG>
  <CONFIG ID="Editor.Config"
    HYPERLINK-TEXT="true"
    OPEN-NODE-PARAM="openNode"
    DECISION-ID-SOURCE="source-Decision-ID"
    DECISION-ID-TARGET="decision-ID">
    <TYPE NAME="Product" EDIT-PAGE="RulesResult" />
    <TYPE NAME="Assessment" EDIT-PAGE="RulesResult" />
    <TYPE NAME="SubRuleSet" EDIT-PAGE="RulesResult" />
    <TYPE NAME="ObjectiveGroup" EDIT-PAGE="RulesResult" />
    <TYPE NAME="ObjectiveListGroup" EDIT-PAGE="RulesResult" />
    <TYPE NAME="Objective" EDIT-PAGE="RulesResult" />
    <TYPE NAME="SubRuleSetLink" EDIT-PAGE="RulesResult" />
    <TYPE NAME="RuleGroup" EDIT-PAGE="RulesResult" />
    <TYPE NAME="RuleListGroup" EDIT-PAGE="RulesResult" />
    <TYPE NAME="Rule" />
    <TYPE NAME="DataItemAssignment" EDIT-PAGE="RulesResult" />
  </CONFIG>
</RULES-CONFIG>
```

Example 8.9 Sample RulesEditorConfig.xml File

Note that the RULES-CONFIG root element only contains the DEFAULT

attribute. This attribute is mandatory and should match an ID on a CONFIG element in this document. The default configuration contains the icon information as well as the default nodes to link to if no configuration is present for a widget. These are covered by the SUCCESS-ICON, FAILURE-ICON, and EDIT-PAGE attributes respectively.

Each CONFIG element has a HYPERLINK-TEXT attribute which is used to specify whether the text next to a rules node in the widget is also to be used as a hyperlink to the link page set by the EDIT-PAGE for the TYPE in question.

Note that the CONFIG with the ID of value of `Editor.Config` has the optional attribute OPEN-NODE-PARAM. This attribute is the name of a page parameter whose value is the ID of a node to open to when the page is opened.

The CONFIG attributes DECISION-ID-SOURCE and DECISION-ID-TARGET are used to identify a page parameter whose value will be the source for a new parameter (named by the DECISION-ID-TARGET) appended to each link on the widget. The above example will look for a page parameter called `source-Decision-ID` and pass on its value as a parameter to any links on the widget. This new value will be identified by a parameter named `decision-ID`.

The decision ID parameter may also be sourced from a field on a server bean instead of from a page parameter. This is achieved by adding DECISION-ID-SOURCE-BEAN and DECISION-ID-SOURCE-FIELD attributes to the CONFIG element instead of a DECISION-ID-SOURCE attribute. A validation error is thrown if all three are present. The DECISION-ID-SOURCE attribute should be the name of a bean on the page and the DECISION-ID-SOURCE-FIELD attribute should be the full name of a field providing the decision ID value. The following is an example of this configuration:

```
<CONFIG ID="Decision.ID.Bean.Source"
  HYPERLINK-TEXT="true"
  OPEN-NODE-PARAM="openNode"
  DECISION-ID-TARGET="decision-ID"
  DECISION-ID-SOURCE-BEAN="DISPLAY"
  DECISION-ID-SOURCE-FIELD="dtls$decision-ID">
  <TYPE NAME="PRODUCT" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="ASSESSMENT" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="SUBRULESET" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="OBJECTIVE_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="OBJECTIVE_LIST_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="OBJECTIVE" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="RULE_GROUP" EDIT-PAGE="RulesResult" />
  <TYPE NAME="RULE_LIST_GROUP" EDIT-PAGE="RulesResult"/>
  <TYPE NAME="RULE" EDIT-PAGE="RulesResult"/>
</CONFIG>
```

Example 8.10 Example of Decision ID Sourced from a Bean

8.10 Meeting View

8.10.1 Overview

The meeting view is a control that displays scheduling information in a chart format. It is associated with the `USER_DAILY_SCHEDULE` domain. The data to display in the meeting view is in XML format. Configuration settings for the meeting view must be in a file called `MeetingViewConfig.xml` in a component. The format for the XML data and configuration settings are described below. Finally, the control has two modes of operation: single and multiple selection.

8.10.2 Single Selection Mode

The first column gives a list of users. The second column indicates the duration of the event to be scheduled. The third column displays the times during the day that the user is available or busy. The available times are hyperlinks that can be clicked to indicate the schedule the start time for the meeting. Note that any parameters passed to a page containing the meeting view will be included in any links within the view. Only start times that can accommodate the relevant meeting duration will be hyperlinks. For example, in Figure 8.4, *Single Selection Mode Example* below, John Smith is busy from 10:30 until 12:30, so it would not be possible to select 10:00 as the start time for a meeting with a duration of one hour and the 10:00 time slot will not be a hyperlink.

User	Duration	Schedule
John Smith	1:00	
Joe Davis	0:30	
Jack Murphy	1:30	
Jim McGuinness	1:55	

Figure 8.4 Single Selection Mode Example

Note that any parameters passed to a page containing the meeting view will be included in any links within the view.

8.10.3 Multiple Selection Mode

This view returns a tab-delimited list of the user IDs of selected rows. The meeting view widget in this mode is the same as that described above for the single selection mode except that it has an extra column which is inserted as the first column in the list and has a selectable checkbox for each list item. The users in this mode of widget are chosen by selecting their associated check boxes. Time slots are not hyperlinked and are for display only.

8.10.4 XML Formats

The meeting view control expects information in a specific XML format. Below is an example of this:

```
<SCHEDULE MODE="Single|Multiple" TYPE="User"
  READ_ONLY="False" DATE="2003-30-10">
  <USER NAME="John Smith" ID="12345" DURATION="90">
    <BUSY START="2003-30-10 10:30:00" END="2003-30-10 12:30:00"/>
    <BUSY START="2003-30-10 15:45:00" END="2003-30-10 16:15:00"/>
  </USER>
  <USER NAME="James Smith" ID="12346" DURATION="90">
    <BUSY START="2003-30-10 12:30:00" END="2003-30-10 13:30:00"/>
    <BUSY START="2003-30-10 15:00:00" END="2003-30-10 18:15:00"/>
  </USER>
</SCHEDULE>
```

Note that in the format above: the MODE attribute is either Single or Multiple; the DURATION attribute is in minutes; START and END attributes are date-times in the format “yyyy-MM-dd HH:mm:ss”. The READ_ONLY attribute, if set to false, indicates that no time slot will be selectable as a hyperlink. The DATE attribute contains the date of the current scheduling and must be supplied. It should be in the format “yyyy-MM-dd”. Finally, the TYPE attribute associates the schedule information with configuration settings which are also specified in an XML format as below:

```
<SCHEDULE_CONFIG>
  <CONFIG TYPE="User" INTERVAL="15" START="08:00" END="16:00">
    <USER_HOME PAGE="PersonHome"
      ID_PARAM="UserID" NEW_WINDOW="True" />
    <NEW_EVENT PAGE="AddNewEvent" ID_PARAM="UserID"
      START_PARAM="start" END_PARAM="end" />
    <MULTI_SELECT PAGE="SelectedUsers"
      TAB_STRING_PARAM="selectedUsers"
      DATE_PARAM="eventDate" />
  </CONFIG>
</SCHEDULE_CONFIG>
```

Where INTERVAL is the duration in minutes of each segment of the time line. This can be 15, 30, or 60. Only these values are acceptable. The START and END attributes detail the beginning and end times of the time line. They are in the form “HH:mm”. Each CONFIG element can have the following sub-elements:

USER_HOME

The PAGE attribute details which page to link to when clicking on the user's name. The ID_PARAM attribute is the name of the parameter to supply with the user's ID as a value. NEW_WINDOW attribute, true by default, specifies if the link opens in a new window or not.

NEW_EVENT

The PAGE attribute details which page to link to when clicking on a time slot. The ID_PARAM attribute is the name of the parameter to supply with the user's ID as a value. The START_PARAM attribute is the name of the parameter to supply with the start time of the new event. Similarly, the END_PARAM describes the name of the end time parameter. Both of these attributes will be in the current application's date-time format.

MULTI_SELECT

The `PAGE` attribute details which page to link to when the submit button on the multi-select view is pressed. `TAB_STRING_PARAM` is the name of the link parameter to supply containing the tab-delimited string of selected users. `DATE_PARAM` is the name of another link parameter containing the date of the event in question. The date value is taken from the value of the `DATE` attribute on the `SCHEDULE` element.

8.11 Charts

8.11.1 Overview

Charts are displayed when one of the domains of `CHART_XML`, `LINE_CHART_XML`, `PIE_CHART_XML` or `BARCHART_XML` domains (or any derivation of them) is used as the source of a field.



Note

Charts are rendered in the browser using *Adobe®Flex* technology, which requires *Adobe®Flash Player*. Refer to the *Cúram Third-Party Tools Installation Guide for Windows* document to see the supported version of Adobe Flash Player.

8.11.2 Chart appearance

The figure below shows a bar chart. Each row represents a unit of information comprised of a caption and a stack of differently colored bars of variable length. Their length represents the quantity of the unit in question and can be ascertained using the numbered marks on the horizontal axis, or a data tip which is available when you hover over the unit, as described below. The chart scale is chosen to fit the biggest stack of bars (this might be overridden by a configuration setting). Each bar is a hyperlink to a page containing further information. The vertical axis of this chart displays captions, describing each bar stack category. Captions might be dates as in example below, date ranges or textual values. Captions are optionally rendered as hyperlinks leading to pages with additional information. Both bar links and caption links are configurable, as described in Section 8.11.3, *Chart configuration*.

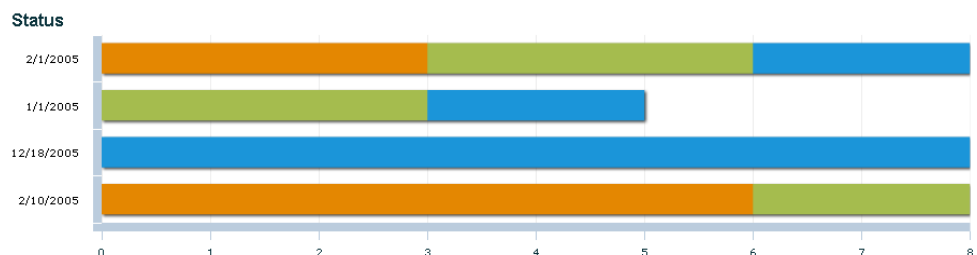


Figure 8.5 Bar Chart Example

**Note**

Colors are not customizable, they are automatically calculated by Adobe Flex technology.

Captions might be dates as in example below, date ranges or textual values. They are optionally rendered as hyperlinks leading to pages with additional information, in which case captions are additionally visually indicated when hovered over.

Textual captions, as shown above, might get longer than one line. In such a case long captions are wrapped within the category segment. If a caption text exceeds two lines, though, it is truncated at that point and an additional tool tip with the full label text is displayed when such a label is hovered over.

Both bar links and caption links are configurable, as described in Section 8.11.3, *Chart configuration*.

A column chart example is shown below. This chart is similar to the bar chart and configurable the same way, except that units of information are displayed in column stacks rather than bars, and axes are interchanged accordingly. It is also possible to configure a column chart so that it has a legend that describes what each of the possible shaded areas in a column represents. The user can hover over a shaded area in a column, which displays what it represents when mapped to an entry in the legend.

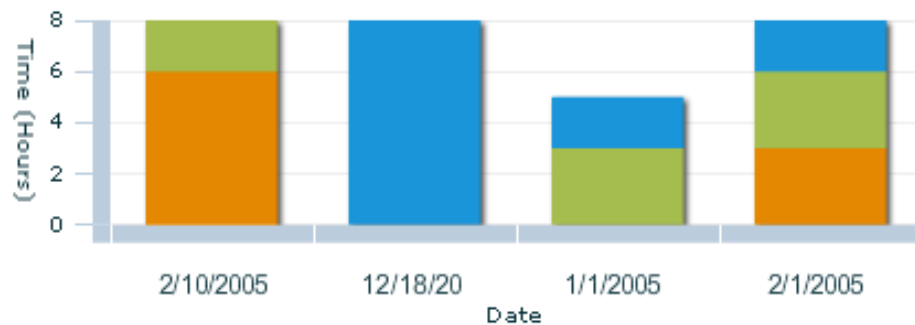


Figure 8.6 Column Chart Example

Another way of presenting chart information is to use a line chart. In this chart, information is rendered as points in each category group, with points of the same type joined by straight lines (e.g. to represent data changes over time). Line charts differ from bar and column charts in that neither the points nor lines are currently hyperlinks. The same applies to line chart captions.

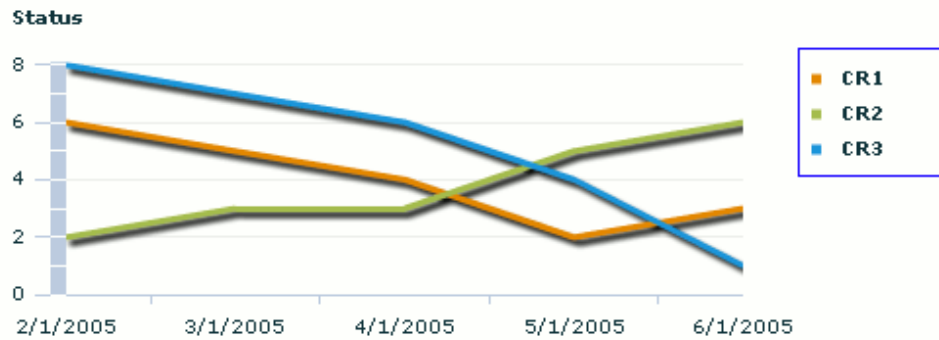


Figure 8.7 Line Chart Example

The last available chart type is a pie chart, an example of which is shown below. Charts of this type are typically used to illustrate relative magnitudes, frequencies or percentages. The arc length of each sector is proportional to the quantity it represents. Together, the sectors create a full disk. Pie charts use callout-like labels, which provide details of the item represented by a sector and its percentage in the pie. Sectors are rendered as hyperlinks, leading to pages with additional information; however, chart labels are not currently available as hyperlinks.

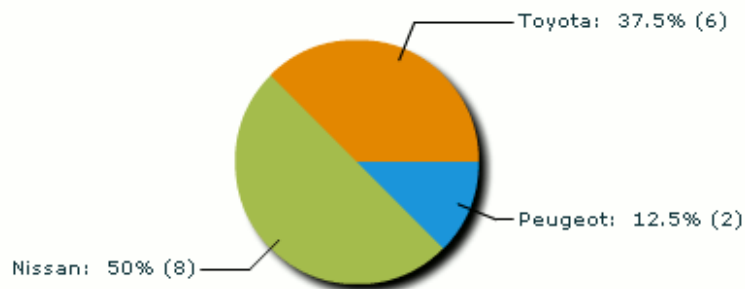


Figure 8.8 Pie Chart Example

By default, charts are displayed without a legend so that all the available space can be dedicated to the chart itself. However, charts can be configured to include a legend which shows extra information on what is represented by the elements of the chart. An example of a chart with a legend is shown below.

The example also shows data tips, which are displayed on a chart when you hover the mouse over a particular chart data element. Data tips are shown regardless of whether a legend is included or not.

- The data tip for bar and column charts shows absolute and relative quantitative information attributed to the element and the element stack, the category (group) to which that element belongs and the type of the element (corresponding to an entry in the legend, if present).

- As line charts are not stacked, relative quantity information is not shown in their data tips; line chart data tips are also displayed only when the mouse is over a data point and not over a line.
- For a pie chart, a data tip displays absolute quantitative information for the particular sector and the percentage of the sector within the disk.



Note

Line charts always display a legend and this is currently not configurable. A legend is currently not displayed for pie charts.

8.11.3 Chart configuration

Various aspects of charts can be configured. This is accomplished by setting the CONFIG attribute on the UIM field in question. The appropriate XML configuration file must contain a configuration section with a unique identifier matching the text in the CONFIG attribute.

All the necessary chart configuration files are to be in your component directory.

Different types of charts are currently configured in separate configuration files:

- Bar charts and column charts both use ChartConfig.xml and are also backward compatible with the previous configuration file version, BarChartConfig.xml (data is taken from whichever of those two files contains a configuration with the required ID; if configurations with the same ID exist in both files, the one found in ChartConfig.xml takes precedence).
- LineChartConfig.xml configuration file is used to look for line chart configuration data.
- Pie chart configuration data is to be placed into file PieChartConfig.xml

The following is a sample of a chart configuration file:

```
<CHART-CONFIG>
  <CONFIG ID="Column.Chart.Config" ORIENTATION="VERTICAL"
    X_AXIS_LABEL="Vert.BarChart.X-Axis"
    Y_AXIS_LABEL="Vert.BarChart.Y-Axis">
    <LEGEND CODETABLE="Attendance">
      <ITEM CODE="CR1"/>
      <ITEM CODE="CR2"/>
      <ITEM CODE="CR3"/>
    </LEGEND>
    <LINK LOCATION="ComponentRedirect">
      <PARAMETER NAME="vertID" VALUE="ID" USE_PAGE_PARAM="false"/>
      <PARAMETER NAME="dueDate" VALUE="START_DATE"
        USE_PAGE_PARAM="false"/>
      <PARAMETER NAME="transID" VALUE="ID" USE_PAGE_PARAM="true"/>
    </LINK>
    <CAPTION_LINK LOCATION="AnotherPage">
      <PARAMETER NAME="vertID" VALUE="ID" USE_PAGE_PARAM="false"/>
      <PARAMETER NAME="dueDate" VALUE="START_DATE">

```

```

        USE_PAGE_PARAM="false"/>
        <PARAMETER NAME="transID" VALUE="ID" USE_PAGE_PARAM="true"/>
    </LINK>
</CONFIG>

<CONFIG ID="BarChart.Config" ORIENTATION="HORIZONTAL"
    CAPTION="Status.Caption"
    CAPTION_TEXT_CODETABLE="Cars"
    MIN_HEIGHT="200" MAX_HEIGHT="500">
    <LEGEND VISIBLE="true" CODETABLE="OldCars">
        <ITEM CODE="CR1"/>
        ...
    </LEGEND>
    <LINK LOCATION="TransferPage">
        <PARAMETER NAME="horID" VALUE="ID" USE_PAGE_PARAM="false"/>
        ...
    </LINK>
</CONFIG>
<CONFIG ID="BarChart.Config" TYPE="line"
    CAPTION="Line.Chart.Caption">
    <LEGEND>
        <ITEM CODE="CR1"/>
        ...
    </LEGEND>
    <LINK LOCATION="ComponentRedirect">
        <PARAMETER NAME="horID" VALUE="ID" USE_PAGE_PARAM="false"/>
        ...
    </LINK>
</CONFIG>
</CHART-CONFIG>

```

The CHART-CONFIG root element contains only CONFIG elements. The CONFIG element contains all configuration for a particular field, identified by the ID attribute. The following table describes all attributes of the CONFIG element. BarChart.properties referred to in this table is a properties file in the client application's <CLIENT_DIR>\components\core folder, used to look up values required.

Attribute	Description
ID	Unique identifier for this CONFIG element.
TYPE	Can be either line or pie, depending on required type of chart. If not present, ORIENTATION attribute will be used to define if bar or column chart is to be displayed.
ORIENTATION	Can be either HORIZONTAL or VERTICAL, depending on required type of chart, HORIZONTAL meaning bar chart and VERTICAL - column chart.
CAPTION_TEXT_CODETABLE	Code table currently used for label captions throughout a chart. If not specified, literal values from chart data will be used.
MAX_VALUE	Maximum value for a numeric axis of column or bar chart. Automatically calculated to fit the maximum element, if not specified.
MAX_INCREMENT	Maximum increment value for a numeric axis of a chart. Numbered ticks are drawn on a chart at the

Attribute	Description
	specified intervals. If not specified, numbered ticks are placed at uniform intervals along the numeric axis, taking into account it's maximum value.
X_AXIS_LABEL	Key to a text property in <code>BarChart.properties</code> . This text is used as the label for the x-axis in the column or line chart, or y-axis in the bar chart. Not used on pie chart.
Y_AXIS_LABEL	Key to a text property in <code>BarChart.properties</code> . This text is used as the label for the y-axis in the column or line chart, or x-axis in the bar chart. Not used on pie chart.
MIN_HEIGHT	This setting is used to define minimum chart object height and is to be specified in pixels. Where a chart contains a small number of items and would be short based on that content size, minimum height introduced by this setting is used. The setting is optional, so 250px default minimum height is used if <code>MIN_HEIGHT</code> is not specified.
MAX_HEIGHT	This setting is used to define the maximum chart object height on screen and should be specified in pixels. Where a chart contains numerous items and its contents exceeds the <code>MAX_HEIGHT</code> specified, this setting is used for the chart object height and a vertical scrollbar appears to allow for access to the rest of the items in the chart. The setting is optional and a default of 250px is used if the attribute is not specified. A value of -1 for <code>MAX_HEIGHT</code> means that the chart takes whichever height its content needs to be displayed in full. It is worth noting that the minimum height setting, either default or explicit, is still taken into account in this case. As a result, charts with little content will not be shorter than minimum or default height implies. Finally, a chart with <code>MAX_HEIGHT</code> set to -1 will not display its vertical scrollbar and the browser scrollbar will appear once the chart is too big to fit into the screen area available.
CAPTION	Key to a text property in <code>BarChart.properties</code> . This text is used as the label for the whole chart.

Table 8.1 Attributes of the CONFIG element



Note

The example lists sample `ChartConfig.xml` contents. The older

format in `BarChartConfig.xml` is almost the same except that the root element is called `BARCHART-CONFIG`.

The older versions of `BarChartConfig.xml` do not contain configuration for label links. This element might be added, if required to this file directly; it is preferable, though, to create appropriate full configuration with the same ID in the `ChartConfig.xml` which will override the older version.

`MIN_HEIGHT` and `MAX_HEIGHT` settings currently do not apply to line or pie charts and will be ignored for these types.

The `CONFIG` element has three child elements: `LEGEND`, `LINK` and optional `CAPTION_LINK`.

- The `LEGEND` element defines the items available for use in the `TYPE` attribute of a `BLOCK` element in chart data returned from the server. The element has an optional `CODETABLE` attribute, specifying the code table used for legend item translation, and an optional `VISIBLE` attribute which indicates if the legend should be seen on screen or not. This attribute has a default value of `false`, so it must be explicitly set to `true` in order for the legend to be displayed.

The `ITEM` child element of specifies each legend entry. Its `CODE` attribute is the text or code table code used to identify a legend item. The code table containing the `CODE` value will be ascertained first from the `CAPTION_TEXT_CODETABLE` value of the `CONFIG` element, then the `CODETABLE` attribute on the `LEGEND` element value, or, in case neither of these attributes are present or do not apply to a particular `CODE`, the literal value will be used as a caption. The same caption is used for a context data tip displayed when mouse pointer is over a corresponding chart element.

- The `LINK` child element is used to configure hyperlinks on bar chart bars and column chart columns or pie chart segments. Its `LOCATION` attribute is the ID of the UIM page to link to. A `LINK` element can have any number of `PARAMETER` child elements. The `NAME` attribute of a `PARAMETER` is the name to give the parameter when transferred as part of hyperlink. The `VALUE` attribute is the name of the attribute on the `BLOCK` element or the `CAPTION` element in the chart input data returned from the server (see below) to use as a parameter value unless `USE_PAGE_PARAM` is `true`, in which case `VALUE` is the name of a page parameter.
- Finally, the `CAPTION_LINK` element is used whenever chart captions are intended to be rendered as links and contains separate settings for such links. The `CAPTION_LINK` element contents are similar to those of the `LINK` element. When this element is skipped, captions are displayed as static text. Also, captions as links are currently supported on bar and column charts only.

Texts for chart caption and axes labels can be customized and localized by creating a properties file called `BarChart.properties` in the client ap-

plication's <CLIENT_DIR>\components\core folder and placing there values under keys, corresponding to the ones specified among CONFIG element parameters as described above.

In addition, the text displayed for the word `total` displayed in the bar tooltips is customizable using the key `total.tooltip.text` in the `BarChart.properties` file.



Note

Bar colors are not customizable in charts and are automatically calculated by Adobe FLEX.



Collapsible Cluster Support

Collapsible clusters are not supported for any cluster containing this widget.

8.11.4 Chart Data Formats

The data to be displayed in a chart comes from the server in XML format.

Below is example of the XML used to create a chart:

```
<CHART>
  <UNIT>
    <CAPTION TEXT="TR1" START_DATE="2004-12-31"
      END_DATE="2005-03-06" />
    <BLOCK ID="1" TYPE="CR1" DUE_DATE="2005-01-01" LENGTH="33" />
    <BLOCK ID="2" TYPE="CR3" DUE_DATE="2005-02-01" LENGTH="14" />
  </UNIT>
  <UNIT>
    <CAPTION TEXT="TR2" START_DATE="2004-12-31" />
    <BLOCK ID="3" TYPE="CR3" DUE_DATE="2005-01-02" LENGTH="11" />
  </UNIT>
  <UNIT>
    <CAPTION TEXT="TR3" END_DATE="2005-03-08" />
    <BLOCK ID="4" TYPE="CR1" DUE_DATE="2005-01-03" LENGTH="22" />
    <BLOCK ID="5" TYPE="CR2" DUE_DATE="2005-01-09" LENGTH="15" />
    <BLOCK ID="6" TYPE="CR3" DUE_DATE="2005-01-01" LENGTH="8" />
  </UNIT>
</CHART>
```

Example 8.11 Sample Horizontal Bar Chart XML

The root element, CHART, can contain any number of UNIT elements. These elements are used to group related information into groups (clusters) and contain one CAPTION element and one or more BLOCK child elements.

The CAPTION element displays an appropriate caption depending on what attributes are set:

- If either the `START_DATE` or both `START_DATE` and `END_DATE` attributes are set, then the caption will be either a single start date or a range of dates.
- If the `TEXT` attribute is set, then the caption text is first looked for in the code table specified in the `CAPTION_TEXT_CODETABLE` attribute of the CONFIG element (see above), then looked for as a property in

`BarChart.properties` using the `TEXT` value as a key, or, if neither attempt is a match, the literal `TEXT` value is rendered as a caption.

Each `BLOCK` element represents a block to be drawn on a chart as a bar, column, line chart point or pie chart sector. This element must have an associated `TYPE` attribute to match it with a particular item. The `LENGTH` attribute is necessary to define the measurement of the block. In the bar or column chart this is the length/height of a bar/column; in a line chart it's the position of an edge point; in a pie chart it's the relative sector arc length. The `ID` attribute is a unique identifier for a block and can be used as a parameter for any hyperlinks. The optional `DUE_DATE` attribute can also be used as an `ID` parameter for hyperlinks on a particular block. It represents the due date for a given block.



Note

- There are no restrictions on the number or names of the attributes of `BLOCK` element. This facilitates passing an arbitrary set of attributes in the links from a chart (provided the configuration is updated appropriately). However, one should keep in mind, that the names of the attributes provided in this section are reserved and bound to the particular elements, i.e. even though `START_DATE` attribute could be added to a `BLOCK` element, in this case it will be interpreted as a literal value and not a date as it would be in the context of `CAPTION` element.
- Due to the nature of pie chart, no more than one `BLOCK` element will be processed and displayed in this type of chart.

8.12 Heatmap Widget

8.12.1 Overview

The Heatmap widget is a control which displays a grid of items of different importance. Items in the widget are presented by color shades varying from red to blue, indicating their importance level from highest to lowest.

The widget is inserted into the page when the `XML_HEATMAP` domain is associated with `UIM` source property of a `FIELD`.

The Heatmap widget expects XML data from the server in the following format:

```
<HEATMAP>
  <REGION REGION_ID="R1" LABEL="highest importance" />
  <REGION REGION_ID="R2" LABEL="middle importance">
    <ITEM ITEM_ID="id9" LABEL="0009" />
    <ITEM ITEM_ID="id10" LABEL="0010" />
    <ITEM ITEM_ID="id21" LABEL="0021" />
  </REGION>
  <REGION REGION_ID="R3" LABEL="lowest importance">
    <ITEM ITEM_ID="id22" LABEL="0022" />
  </REGION>
</HEATMAP>
```

```
</REGION>
...
</HEATMAP>
```

Here, the REGION elements specify the importance level ("heat") of their contained ITEMS. There should be at least two regions in a heatmap widget. The color will always start from red, so if no items of that importance are there, empty REGION elements should be inserted for the widget to render properly.

The following image shows an example of the Heatmap widget.

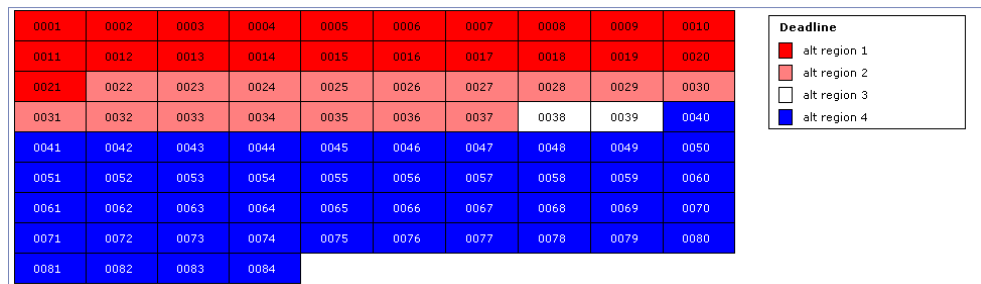


Figure 8.9 Heatmap Example

8.12.2 Configuration

Different types of heatmap can be configured by creating entries in the HeatmapConfig.xml file in your components directory, using the following format:

```
<HEATMAP_CONFIG>
  <CONFIG ID="Map1" NUM_COLS="10" NUM_ROWS="4"
    LEGEND_POSITION="LEFT"
    LEGEND_TITLE="Deadline"
    LEGEND_TITLE_PROPERTY="Localised.Legend.Title">
    <ITEM_LINK PAGE_ID="Sample_page">
      <PARAM NAME="configParameter" VALUE="ITEM_ID" />
    </ITEM_LINK>
  </CONFIG>
  <CONFIG ID="Map2" NUM_COLS="6">
    ...
  </CONFIG>
</HEATMAP_CONFIG>
```

The attributes of a CONFIG element are summarized in the following table:

Attribute	Description
NUM_COLS	This attribute allows you to set the number of items displayed in each row of the Heatmap
NUM_ROWS	This attribute allows you to specify the number of visible rows in the Heatmap. If this attribute is set to less rows than are required to display the data, a vertical scrollbar will be provided. If this attribute is not present, the widget will expand to display as many rows as are required.
LE-	By default, the Heatmap legend is drawn to the right

Attribute	Description
GEND_POSITION	of the widget. This attribute can be used to draw the legend to the left instead, by setting it's value to LEFT.
LEGEND_TITLE	The default title for a legend is Legend. This attribute can be used to specify a more logical title to use.
LEGEND_TITLE_PROPERTY	Optional attribute used to customize/localize the displayed title. The value here is the key in the CDEJResources.properties file or its localized version (see Chapter 4, <i>Localization</i> for more details on localization).

Table 8.2 Attributes for **CONFIG** element

The ITEM_LINK element can be used to specify the page to which to link when a user clicks on an item in the Heatmap, by setting it's PAGE_ID attribute. The PARAM child element can be used to specify what page parameters to pass (the NAME attribute) and what data items to use as their value (the VALUE attribute). Values which don't match any attributes in the ITEM elements in the Heatmap XML are assumed to be literal values.

To specify which configuration to use for a given instance of the Heatmap widget, the CONFIG attribute of the field containing the widget should be set to the ID of the desired configuration.

8.13 Workflow

8.13.1 Overview

A workflow depicts a series of steps that routinely take place in order for a unit of work to be completed. The WORKFLOW_GRAPH_XML domain, or any derivation of it, causes a workflow to be displayed. The data to be displayed in a workflow comes from the server in XML format. Configuration settings for the Workflow must be in a file called WorkflowConfig.xml, of which there can be only one per component. The format for the XML data and configuration settings are described below. Any static text for this view can be customized and localized by creating a properties file called Workflow.properties in the client application's <CLIENT_DIR>\components\core folder.

8.13.2 Workflow Details

Figure 8.10, *Workflow* shows a sample workflow view. A box, along with a representative icon, represents a discrete unit of work and is called an *activity*. Any line connecting nodes is called a *transition* and is intended to illustrate the flow of work. For this reason, the start and end activities are repres-

ented by icons only. Workflow proceeds from the left and ends at the right-most activity. An activity is a hyperlink to a tab containing further details on that activity. An activity can have a second, smaller icon indicating that there is a notification on this activity. Clicking on the notification icon (a small envelope in the image below) will open a separate tab with details of the notification.

An activity has an entry point and an exit point for a transition, on the right and the left respectively. When two or more transitions leave an exit point this is called a split. The transitions in a split can be given a number to indicate their relative progression. When two or more transitions meet at an activity's entry point this is called a join. If either a join or a split is an “and” type, also called a “conjunction”, then it is represented as a small square. This implies that a series of transitions have to take place together in order for the workflow to proceed. If a join or a split is an “xor” type, an either-or situation, then a small circle is used. There are examples of both in the figure below. Finally, a transition can have an associated transition condition. This means that certain criteria have to be met in order for a transition to proceed. This is represented by an asterisk on the transition and the full condition information is displayed in a pop-up if the user hovers the mouse over the symbol.

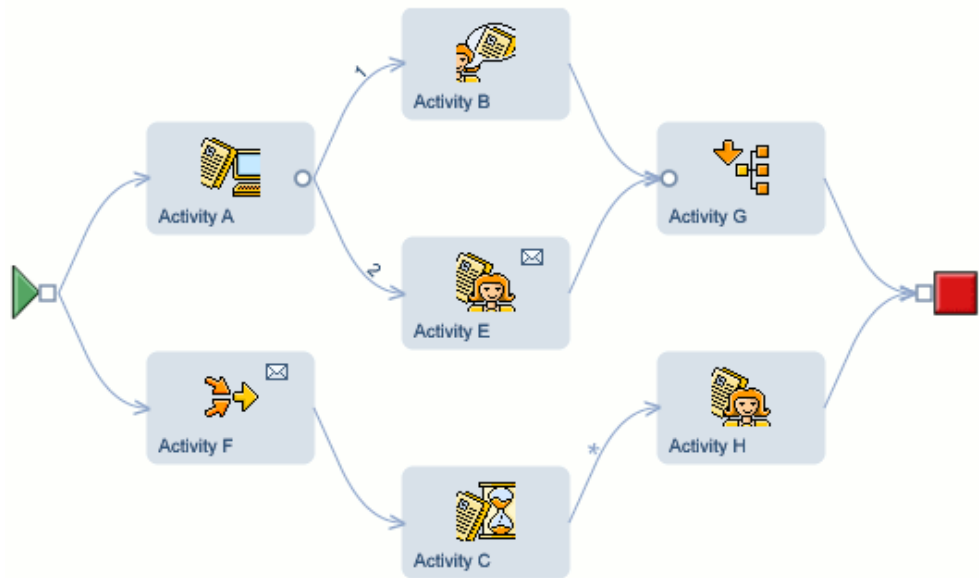


Figure 8.10 Workflow

8.13.3 Workflow XML Formats

The workflow widgets require XML data that conforms to the workflow schema defined in the workflow.xsd file located in the lib\curam\xml\schema folder of your CDEJ installation folder. Below is an example of workflow XML data:

```
<WORKFLOW ID="4791830003522207744" PROCESS-VERSION="1" >
```

```

<NODE ID="6953557824660045824" X="2.0" Y="1.0"
TEXT="Loop Activity [End]" HIDDEN="false"
ACTIVITY-TYPE-CODE="AT9" HAS-NOTIFICATION="true"
IS-EXECUTED="false" SPLIT-TYPE="AND" JOIN-TYPE="AND"
TASK-ID="1" />
<NODE ID="-3566850904877432832" X="3.0" Y="1.0"
TEXT="EndProcessActivity" HIDDEN="false"
ACTIVITY-TYPE-CODE="AT7" IS-EXECUTED="false"
JOIN-TYPE="AND" TASK-ID="2" />
<NODE ID="2702159776422297600" X="1.0" Y="2.0"
TEXT="Activity 1" HIDDEN="false"
ACTIVITY-TYPE-CODE="AT5" IS-EXECUTED="false"
SPLIT-TYPE="AND" JOIN-TYPE="AND" TASK-ID="3" />
<EDGE FROM="6953557824660045824" TO="-3566850904877432832"
HIDDEN="false" TRANSITION-ID="1621295865853378560"
IS-EXECUTED="false" REVERSE-ARROW="false" />
<EDGE FROM="3566850904877432832" TO="2702159776422297600"
HIDDEN="false" TRANSITION-ID="0" IS-EXECUTED="false"
REVERSE-ARROW="true" />
</WORKFLOW>

```

The root element, WORKFLOW, can have any number of NODE (activity) and EDGE (transition) elements. The ID attribute on WORKFLOW identifies this particular workflow as does the PROCESS-VERSION attribute.

The NODE element represents a single activity in the workflow. All attributes of a node are defined in the following table:

Attribute	Description
ID	Unique identifier for this element, supplied as a parameter in the row header hyperlink.
X	An x-coordinate for an element on the workflow graph.
Y	A y-coordinate for an element on the workflow graph.
TEXT	The text of an activity.
ACTIVITY-TYPE-CODE	Code for an activity type. Used as a parameter in a hyperlink.
HIDDEN	Boolean property to indicate if an edge or node is to be hidden. If true the node will not be displayed.
IS-EXECUTED	Boolean property to indicate if an activity has already been executed for a particular process instance. If set to true then the activity has been executed.
SPLIT-TYPE	The split type associated with an activity.
JOIN-TYPE	The join type associated with an activity.
ACTIVITY-INSTANCE-ID	The unique identifier of an activity instance for a particular process instance.
START-DATE-TIME	The start date time of an activity instance or transition instance for an executed or currently executing process.
END-DATE-TIME	The end date time of an activity instance or trans-

Attribute	Description
	ition instance for an executed or currently executing process.
STATUS	The current status of an activity instance.
TASK-STATUS	Code for the status of a task.
TASK-RE-SERVED-BY	The name of the user reserving the task.
TASK-TOTAL-TIME-WORKED	The total time worked on a task in seconds.
NUMBER-ITERATIONS	The number of times the activity contained in a node has been executed.
TASK-ID	The unique identifier for the task.

Table 8.3 Attributes of a Node

The EDGE element represents a single transition in the workflow. All attributes of an edge are defined in the following table:

Attribute	Description
FROM	The ID of the node this edge is from.
TO	The ID of the node this edge is to.
TRANSITION-ID	The unique identifier of a transition.
IS-FOLLOWED	Boolean property to indicate if a particular transition has already been followed for a process instance.
TRANSITION-INSTANCE-ID	The unique identifier of a transition instance for a particular process instance.
REVERSE-ARROW	Boolean property to indicate if an arrow on an edge should be reversed. In this case, the arrow will be going into the FROM node instead of the TO node.
IS-EXECUTED	Boolean property to indicate if an activity has already been executed for a particular process instance. If set to <code>true</code> then the activity has been executed.
TRANSITION-CONDITION	The condition associated with a transition in an edge.
REAL_FROM	ID of a node that this edge is actually from as opposed to an intermediate hidden node identified by the FROM attribute.
REAL_TO	ID of a node that this edge is actually to as opposed to an intermediate hidden node identified by the TO attribute.
ENABLED	Boolean property to indicate if an edge is to be en-

Attribute	Description
	abled as a hyperlink. This attribute is <code>false</code> by default.
ORDER	Indicates the order of an edge relative to other edges.

Table 8.4 Attributes of an Edge

As mentioned above, workflow charts are configurable. This is accomplished by setting the `CONFIG` attribute on the UIM field in question. The `WorkflowConfig.xml` XML configuration file must contain a configuration section with a unique identifier matching the text in the `CONFIG` attribute. The XML schema format for this file is defined in the `workflow-config.xsd` file located in the `lib\curam\xml\schema` folder of your CDEJ installation folder. The following is a sample of this file:

```
<WORKFLOW_CONFIG>
  <ICON CODE="AT1" PATH="Images/manual.gif"/>
  <ICON CODE="AT2" PATH="Images/automatic.gif"/>
  <ICON CODE="AT4" PATH="Images/subflow.gif"/>
  <ICON CODE="AT5" PATH="Images/route.gif"/>
  <ICON CODE="AT6" PATH="Images/eventwait.gif"/>
  <ICON CODE="AT7" PATH="Images/endprocess.gif"/>
  <ICON CODE="AT8" PATH="Images/loopbegin.gif"/>
  <ICON CODE="AT9" PATH="Images/loopend.gif"/>
  <ICON CODE="AT10" PATH="Images/decision.gif"/>
  <ICON CODE="AT11" PATH="Images/startprocess.gif"/>
  <ICON NOTIFICATION="true"
    PATH="CDEJ/cdej-images/notification.gif"/>
  <CONFIG ID="WorkFlow.Config"
    NOTIFICATION_PAGE="viewActivityNotification"
    DETAILS_PAGE="componentRedirect"
    START_PROCESS_TYPE="AT11" END_PROCESS_TYPE="AT7"/>
</WORKFLOW_CONFIG>
```

The `WORKFLOW_CONFIG` root element contains `CONFIG` elements and `ICON` elements. The `CONFIG` element contains all configuration for a particular field, identified by the `ID` attribute. The following table describes all attributes of the `CONFIG` element:

Attribute	Description
ID	Unique identifier for this configuration.
DETAILS_PAGE	ID of a UIM page to use as a destination for a hyperlink on a node.
HEIGHT	Height in pixels of a workflow chart. If height is not specified, a height will be chosen that attempts to maximize the use of available space.
ACTIVITY_CODETABLE	Codetable name for resolving <code>ACTIVITY-TYPE-CODE</code> attribute values.
TASKSTATUS_CODETABLE	Codetable name for resolving <code>TASK-STATUS</code> attribute values.

Attribute	Description
PRO- CESSSTATUS_CODETABL E	Codetable name for resolving the status of a process instance (e.g. In Progress, Completed, Suspended or Aborted).
SHOW_INSTANCE_DATA	Determines if the chart should display a text area containing all instance data information. Valid settings are <code>true</code> and <code>false</code> .
START_PROCESS_TYPE	Code identifying the ACTIVITY-TYPE-CODE set as the start process type. This activity will be drawn without a box.
END_PROCESS_TYPE	Code identifying the ACTIVITY-TYPE-CODE set as the end process type. This activity will be drawn without a box.
NOTIFICATION_PAGE	ID of a UIM page to use as a destination for a hyperlink on a notification icon.
READONLY_VIEW	Determines if the links on a workflow graph should be disabled.
HIGH- LIGHT_ACTIVITY_PARA M	Represents the parameter used to determine the current activity in a workflow. The value of the parameter is matched with a corresponding attribute in the XML data returned from the server to indicate which node has to be highlighted.

Table 8.5 Attributes of Workflow CONFIG element

The `ICON` child element of the `WORKFLOW_CONFIG` root element defines all icons for the workflow chart. Either the `CODE` attribute or the `NOTIFICATION` attribute defines what kind of icon this is. If `CODE` is set then the `ACTIVITY-TYPE-CODE` on a `NODE` is used to match an icon to a particular activity type. If the `NOTIFICATION` attribute is set to `true` then this icon is used as a graphic depicting a notification present on an activity. The `PATH` attribute on `ICON` is used to point to an image file, relative to your project's `WebContent` directory.

8.14 Evidence View

This view has two modes for displaying and comparing evidence data.

8.14.1 Evidence Display Mode

The `EVIDENCE_XML` domain results in a table displaying evidence items. There are three columns in the table. The first displays the evidence item name, the second shows the group to which evidence item belongs and the value of the item is displayed in the third column. The value of the item will

be formatted based on it's domain.

8.14.2 Evidence Comparison Mode

The EVIDENCE_XML_COMPARE domain results in three tables displaying evidence comparison results. The comparison results consist of three tables to display items which were modified, added or deleted. All three tables follow the same format: the first column displays the evidence item name; the second column displays the group which the evidence item belongs to and corresponding values are displayed in the third (the modified evidence table will have a fourth fourth column to show previous values against current values) column.

8.14.3 Configuration

The evidence view is configurable by changing settings in appropriate properties files. For Evidence Display mode this is the DisplayEvidence.properties file and for Evidence Comparison mode configuration, ComparedEvidence.properties file is used. These properties files should be created in the <CLIENT_DIR>\components\core folder.

Configuration files contain table headers and captions for all the columns as well as visibility settings for each column. There is also a links section for specifying links to pages for each evidence item and item group column if needed. If a link is not required, leave the value empty rather than deleting the property itself. Also there are properties containing textual substitution for an empty value case and textual insert used in evidence item name.



Note

The properties specifying visibility settings are not localizable strings and should contain either “true” or “false” depending on desired visibility of the corresponding column.

Below is an example of the configuration settings for display evidence mode:

```
#Textual descriptions for comparison sections.
Table.Summary.Single=This table contains evidence items.

# Comparison section labels
Evidence.Table.Label=Evidence Items

#Column headers
Description.Column.Header=Rule
Group.Column.Header=Group
Value.Column.Header=Value

#Visibility
Description.Column.Visible=true
Group.Column.Visible=true
Value.Column.Visible=true

# Localizable messages
Message.No.Value=This item is not set
Message.Item.Joint=referenced by rule item
```

```
#Links (Values should be UIM PAGE_IDS)
Description.Column.Link=Home
Group.Column.Link=GroupHome
```

The following is an example of the configuration settings for the evidence comparison mode:

```
#Textual descriptions for comparison sections.
Table.Summary.MODIFIED=This table contains modified evidence
Table.Summary.NEW=This table contains newly added evidence items.
Table.Summary.REMOVED=This table contains removed evidence.

# Comparison section labels
Evidence.Label.MODIFIED=Modified evidence
Evidence.Label.NEW=Newly added evidence items
Evidence.Label.REMOVED=Removed evidence items

#Column headers
Description.Column.Header=Rule
Group.Column.Header=Group
Oldval.Column.Header=Previous Value
Value.Column.Header=New Value

#Visibility
Description.Column.Visible=true
Group.Column.Visible=true
Oldval.Column.Visible=true
Value.Column.Visible=true

#Links (Values should be UIM PAGE_IDS)
Description.Column.Link=Home
Group.Column.Link=GroupHome
```

8.14.4 Data Format

The Evidence view expects the following XML format. Below is an example for Evidence Comparison mode:

```
<EVIDENCE_COMPARE>
  <EVIDENCE TYPE="MODIFIED">
    <GROUP ID="mod1ID"
      DESCRIPTION="en|EvidenceGroup1">
      <EVIDENCE_ITEM ID="modItem1ID"
        DESCRIPTION="en|Number of Children"
        OLDVAL="11" VALUE="13"
        DOMAIN="SVR_INT32"/>
    </GROUP>
    <GROUP ID="mod2ID"
      DESCRIPTION="en|EvidenceGroup2">
      <EVIDENCE_ITEM ID="modItem3ID"
        DESCRIPTION="en|Are you married"
        OLDVAL="false" VALUE="true"
        DOMAIN="SVR_BOOLEAN"/>
    </GROUP>
  </EVIDENCE>
  <EVIDENCE TYPE="NEW">
    <GROUP ID="new1ID"
      DESCRIPTION="en|EvidenceGroup1">
      <EVIDENCE_ITEM ID="newItem1ID"
        DESCRIPTION="en|Number of cars"
        VALUE="6"
        DOMAIN="SVR_INT32"/>
    </GROUP>
  </EVIDENCE>
  <EVIDENCE TYPE="REMOVED">
```

```

<GROUP ID="del1ID"
  DESCRIPTION="en|Deletion">
  <EVIDENCE_ITEM ID="delItem1ID"
    DESCRIPTION="en|Number of houses"
    OLDVAL="1"
    DOMAIN="SVR_INT32"/>
</GROUP>
</EVIDENCE>
</EVIDENCE_COMPARE>

```

The following is an example of the Evidence Display mode:

```

<evidence>
  <group id="group1" display-name="EvidenceGroup1">
    <item name="item11"
      display-name="Number of Children"
      initial-value="13" no-value="false"
      type="SVR_INT32"/>
    <item name="item12"
      display-name="item with no value"
      initial-value="" no-value="true"
      type="SVR_STRING"/>
  </group>
  <group id="group2" display-name="EvidenceGroup2">
    <item name="item21"
      display-name="Are you married"
      initial-value="true" no-value="false"
      type="SVR_BOOLEAN"/>
    <item name="item22"
      display-name="Some important dates"
      initial-value="" no-value="false"
      type="SVR_DATE">
      <value position="10" description="Important date 1"
        value="20050401T000000">
      <value position="18" description="Important date 2"
        value="20050601T000000">
      <value position="5" description="Important date 3"
        value="20051231T000000">
    </item>
  </group>
</evidence>

```

The `display-name` attribute represents a description for every item or group, the `description` does the same for the `value` element. Group ids, evidence item names and value descriptions are supplied by the evidence text returned from the rules engine. The `type` attribute is used to select particular representation for different data types from the server. The name attribute of `item` and the `id` attribute of `group` are used as link parameters if a link is specified for the first or second column.

8.15 Calendar

The calendar is used by any UIM page which displays a field from a server access bean containing a `CALENDAR_XML_STRING` domain. This view allows for scheduling of events from different time-frames; monthly, weekly and daily. The following image shows a section of the calendar week view as it would be displayed in a web page.

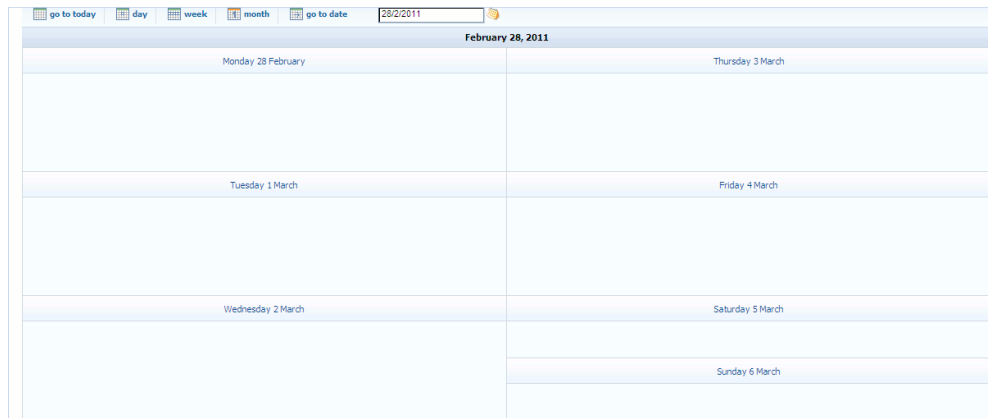


Figure 8.11 Calendar Week View

Programmatically, the calendar expects to be populated with information about events in an XML format.

The following is an example of what the XML received from the server might look like:

```
<CURAM_CALENDAR_DATA TYPE="UserCalendar" >
  <EVENT>
    <ID>1</ID>
    <DATE>2002-10-10</DATE>
    <STARTTIME>10:10:10</STARTTIME>
    <ENDTIME>10:10:10</ENDTIME>
    <DURATION>0</DURATION>
    <DESCRIPTION>Hello World!</DESCRIPTION>
    <STATUS>ATS1</STATUS>
    <PRIORITY>AP1</PRIORITY>
    <LEVEL>AL1</LEVEL>
    <RECURRING>>false</RECURRING>
    <READ_ONLY>>false</READ_ONLY>
    <ALL_DAY>>false</ALL_DAY>
    <ATTENDEE>>true</ATTENDEE>
    <ACCEPTANCE>>true</ACCEPTANCE>
  </EVENT>
  <SINGLE_DAY_EVENT>
    <ID>2</ID>
    <DATE>2003-04-01</DATE>
    <TYPE>ET1</TYPE>
    <DESCRIPTION>April Fool's Day</DESCRIPTION>
  </SINGLE_DAY_EVENT>
</CURAM_CALENDAR_DATA>
```

Example 8.12 Calendar XML Stream

Notice that there can be two kinds of event elements contained within the CURAM_CALENDAR_DATA XML data: EVENT and SINGLE_DAY_EVENT. In the schema of the XML data expected the root element, CURAM_CALENDAR_DATA, can hold any number (zero to many) of EVENT and SINGLE_DAY_EVENT elements; CURAM_CALENDAR_DATA can optionally have a TYPE attribute which associates this sequence of events with a particular calendar configuration (see example below).

The following tables describe the schema definitions for each of the attrib-

utes allowed on the EVENT and the SINGLE_DAY_EVENT elements respectively.

Attribute Name	Description	Required
ID	A string to uniquely identify this event.	
DATE	The date of the event in xs:date format: (CCYY-MM-DD) I.e. 21- Aug-2002 is represented as 2002-08-21.	No
STARTTIME	The start time in xs:time format: (hh:mm:ss). I.E. 1:34 pm and 56 seconds is represented as 13:34:56.	
ENDTIME	The start time in xs:time format: (hh:mm:ss).	No
DURATION	The duration of the event in minutes. This should be an integer.	No
DESCRIPTION	A Description of the event.	No
STATUS	The status of the event. This node is limited to values stored in the Activity-TimeStatus code table in the reference application.	No
PRIORITY	The priority of the event. This node is limited to values stored in the ActivityPriority code table in the reference application.	No
LEVEL	Code that shows the level of the activity. This node is limited to the values stored in the ActivityLevel code table in the reference application.	No
RECURRING	Recurring indicator: true if this event is a recurring event. Otherwise false.	No
READ_ONLY	Read-only indicator: true if this event is a read-only event. Otherwise false.	No
ALL_DAY	All-day indicator: True if this is an all-day event. Otherwise false.	No
ATTENDEE	Attendee indicator: true if the user is attending a meeting. Otherwise false.	No
ACCEPTANCE	Acceptance indicator: True if the user has accepted an invitation to a meeting. Otherwise false.	
POSITION	For a spanning event, indicates first or last component of the event.	No

Table 8.6 EVENT attributes in schema

Attribute Name	Description	Required
ID	A string to uniquely identify this event.	No
DATE	The date of the event in xs:date format.	No
TYPE	The type of a single day event.	No
DESCRIPTION	A Description of the event.	No

Table 8.7 SINGLE_DAY_EVENT attributes in schema

Once a field based on the CALENDAR_XML_STRING domain returns XML information formatted according to the aforementioned schema, it will be displayed in the appropriate time position by the calendar. Any web page containing a calendar can be set to open on different dates and views by specifying the *startDate* and *calendarViewType* parameters in the page's URL. The *startDate* parameter should be formatted according to the date format expected by the application and the *calendarViewType* parameter should be one of the following codes.

Code	Value
CVT1	Day view
CVT2	Week view
CVT3	Month view

Table 8.8 Calendar View Type Values

You can configure the display of calendar information using the CalendarConfig.xml file. There should be at least one copy of this in the components folder. This file should contain configuration information for each type of calendar, the TYPE attribute of the CURAM_CALENDAR_DATA element mentioned above associates a calendar data stream with a particular type. The following is an example of the structure of the CalendarConfig.xml

```
<CONFIGURATION MONTH_CELL_HEIGHT="4"
    SHOW_REPEAT_EVENT_TEXT="true">
  <CALENDAR TYPE="UserCalendar">
    <DESCRIPTION_LOCATION>DetailsPage.do</DESCRIPTION_LOCATION>
    <DAY_VIEW_TIME_FORMAT>24</DAY_VIEW_TIME_FORMAT>
  </CALENDAR>
  <EVENT_TYPES>
    <TYPE NAME="ET1" ICON="Images/mandatory.gif"/>
    <TYPE NAME="ET2" ICON="Images/case.gif"/>
    <TYPE NAME="ET3" ICON="Images/concern.gif"/>
  </EVENT_TYPES>
</CONFIGURATION>
```

Example 8.13 CalendarConfig.xml Example

The overall schema for this configuration file specifies the CONFIGURATION element as the root element. The CONFIGURATION has an optional MONTH_CELL_HEIGHT attribute which sets the maximum number of rows

to display in a single cell in the month view. The default value is three. The `SHOW_REPEAT_EVENT_TEXT` optional attribute, if set to `true`, will display the event description in each month cell if an event spans multiple days. This attribute is `false` by default.

The `CONFIGURATION` root element can hold any number of `CALENDAR` elements and a single `EVENT_TYPES` element. The `TYPE` attribute of `CALENDAR` associates this configuration information with an XML stream returned from the server. The `DESCRIPTION_LOCATION` element of `CALENDAR` is for constructing a link to a page containing more information on any event in the calendar. The following table lists the parameters passed with this hyperlink.

Parameter Name	Description
ID	the event ID
RE	Recurrence indicator
AT	Attendee indicator
RO	Read-only indicator
LV_	Activity level
AC	Acceptance indicator

Table 8.9 Parameters Passed to Event Description Pages

The `CALENDAR` element should also contain an element called `DAY_VIEW_TIME_FORMAT`. The valid values for this element are 12 and 24. They specify whether the time in the day view is displayed using a 12 or 24 hour format.

The `EVENT_TYPES` element is used for mapping images to display as icons next to single day events. The `NAME` attribute of the `TYPE` element must match a `TYPE` element on a `SINGLE_DAY_EVENT` supplied by the server for the image specified by the `ICON` attribute to be displayed.

The schema for the calendar configuration file (`CalendarConfiguration.xsd`) and the schema for the `CALENDAR_XML_STRING` domain (`CuramCalendar.xsd`) are located in your project's `WebContent/WEB-INF/CDEJ/schema` folder.

8.16 Payment Statement View

The payment statement view is used for displaying under or over payment within the *Cúram* application framework. This following image demonstrates this view:




Action	Period	Description	Actual	Reassessed	Difference
	01/01/2000 to 10/01/2000	Gross Payment	100.00	90.00	10.00
		Another Payment	100.00	90.00	10.00
		Net Payment	100.00	90.00	10.00
	10/01/2000 to 20/01/2000	Gross Payment	100.00	90.00	10.00
		Another Payment	100.00	90.00	10.00
		Net Payment	100.00	90.00	10.00
	20/01/2000 to 30/01/2000	Gross Payment	100.00	90.00	10.00
		Another Payment	100.00	90.00	10.00
		Net Payment	100.00	90.00	10.00
Total Gross Over Payment					35.00
Total Tax Deduction					7.50
Total Utility Deduction					7.00
Total Liability Deduction					0.00
Total Net Under or Over Payment					89.30

Figure 8.12 Payment Statement View

The payment statement view supports the display of benefits as well as liabilities. The domain `BENEFIT_REASSESSMENT_RESULT_TEXT` should be used for a benefit payment statement view. The domain `LIABILITY_REASSESSMENT_RESULT_TEXT` should be used for a liability payment statement view. It is expected that all string data returned for this field follows a specific tab-delimited format. Examples of using these domains can be found in the *Cúram* reference application.

There is also a properties file associated with this view: `PaymentStatement.properties` in the `<CLIENT_DIR>\components\core` folder. The link to a page providing further details on a statement can be defined using a set of four parameters:

```
PaymentStatement.RowLink.Benefit.PageID
PaymentStatement.RowLink.Benefit.ParameterName
PaymentStatement.RowLink.Benefit.Label
PaymentStatement.RowLink.Benefit.Image
```

There is one set of parameters for Benefit pages and one for Liability pages. `PageID` is the name of the page to link to. `ParameterName` is the name of the parameter to be passed to this page to identify the id of the payment in question. `Label` supplies the text of the link, if `Image` is not used. Otherwise it supplies the tool-tip for the image-based link.

The remaining properties are simply externalized strings for the widget.

```
PaymentStatement.RowLink.Benefit.PageID=FromBenefit
PaymentStatement.RowLink.Liability.PageID=FromLiability

PaymentStatement.RowLink.Benefit.ParameterName=param1
PaymentStatement.RowLink.Liability.ParameterName=param2

PaymentStatement.RowLink.Benefit.Label=Link Text 1
PaymentStatement.RowLink.Liability.Label=Link Text 2

#PaymentStatement.RowLink.Benefit.Image=Images/icon.gif
PaymentStatement.RowLink.Liability.Image=Images/icon.gif

PaymentStatement.Text.fromToDateSeparator=\ to
PaymentStatement.Text.Action=Action
PaymentStatement.Text.Period=Period
```

```

PaymentStatement.Text.Desc=Description
PaymentStatement.Text.Actual=Actual
PaymentStatement.Text.Reassessed=Reassessed
PaymentStatement.Text.Liability.Received=Received
PaymentStatement.Text.Diff=Difference
PaymentStatement.Text.GrossTotal=Total Gross Over Payment
PaymentStatement.Text.TaxTotal=Total Tax Deduction
PaymentStatement.Text.UtilityTotal=Total Utility Deduction
PaymentStatement.Text.LiabilityTotal=Total Liability Deduction
PaymentStatement.Text.NetTotal=Net Under or Over Payment

```

Example 8.14 A Sample PaymentStatement.properties File

8.17 Batch Function View

The batch function view is generated from the PARAM_TAB_LIST domain. It allows you to enter parameters to submit a batch program for execution. The labels of each field are provided to the view by a single tab-delimited string.

8.18 Addresses

The ADDRESS_DATA domain type maps to a tag for entering and displaying addresses. Although the user sees several fields, addresses are stored as a single string field. Each of the fields displayed as part of the out-of-the-box address are text input fields except for the state field which is drop-down field.

To parse the address and display it, the elements that make up the address have to be defined in the curam-config.xml file. Different address configurations for different locales in the Cúram application can be defined. Example 8.15, *Address Configuration in curam-config.xml* demonstrates how to set this configuration using the ADDRESS_CONFIG element.

```

<ADDRESS_CONFIG>
  <LOCALE_MAPPING LOCALE="en_US"
    ADDRESS_FORMAT_NAME="US" />
  <LOCALE_MAPPING LOCALE="en_GB"
    ADDRESS_FORMAT_NAME="UK" />
  <ADDRESS_FORMAT NAME="US" COUNTRY_CODE="US">
    <ADDRESS_ELEMENT LABEL="Address.Label.AptSuite"
      NAME="ADD1" />
    <ADDRESS_ELEMENT LABEL="Address.Label.Street.1"
      NAME="ADD2" />
    <ADDRESS_ELEMENT LABEL="Address.Label.Street.2"
      NAME="ADD3" />
    <ADDRESS_ELEMENT LABEL="Address.Label.City"
      NAME="CITY" />
    <ADDRESS_ELEMENT CODETABLE="AddressState"
      LABEL="Address.Label.State"
      NAME="STATE" />
    <ADDRESS_ELEMENT LABEL="Address.Label.Zip"
      NAME="ZIP" />
  </ADDRESS_FORMAT>

  <ADDRESS_FORMAT NAME="UK" COUNTRY_CODE="GBR">
    <ADDRESS_ELEMENT LABEL="Address.Label.Address.1"
      MANDATORY="true" NAME="ADD1" />
    <ADDRESS_ELEMENT LABEL="Address.Label.Address.2"

```

```

        NAME="ADD2" />
    <ADDRESS_ELEMENT LABEL="Address.Label.Address.3"
        NAME="ADD3" />
    <ADDRESS_ELEMENT LABEL="Address.Label.Address.4"
        NAME="ADD4" />
    <ADDRESS_ELEMENT LABEL="Address.Label.County"
        NAME="ADD5" />
    <ADDRESS_ELEMENT LABEL="Address.Label.City"
        NAME="CITY" />
    <ADDRESS_ELEMENT LABEL="Address.Label.PostCode"
        NAME="POSTCODE" />
    <ADDRESS_ELEMENT CODETABLE="Country"
        LABEL="Address.Label.Country"
        NAME="COUNTRY" />
</ADDRESS_FORMAT>
</ADDRESS_CONFIG>

```

Example 8.15 Address Configuration in curam-config.xml

The `ADDRESS_CONFIG` element is built using multiple, `LOCALE_MAPPING` elements and `ADDRESS_FORMAT` elements. In a situation where the Cúram application is deployed with multiple locales a developer may wish to use different `ADDRESS_FORMAT` tags for different locales. To do this we use the `LOCALE_MAPPING` element. This element contains a `LOCALE` attribute which defines the locale and an `ADDRESS_FORMAT_NAME` attribute which defines the `ADDRESS_FORMAT` element to map to. By default, the OOTB Cúram application has two `ADDRESS_FORMAT` elements defined. A US format which is automatically mapped to the `en_US` locale and a UK format which is automatically mapped to the `en_GB` locale. As these locales are automatically mapped we are not required to define `LOCALE_MAPPING` elements for them, but they are shown in the example above to illustrate how the `LOCALE_MAPPING` element is used.

The `ADDRESS_FORMAT` has an optional `COUNTRY_CODE` attribute which is used in the address header when an address is first created. If it is not set, the `COUNTRY_CODE` defaults to `GBR` when the address format specified is `UK` and to `US` for everything else. The `COUNTRY_CODE` is not used by the infrastructure. It is one of the fields in the address string used by the application, but infrastructure provides an initial value for it.

The `ADDRESS_FORMAT` elements contain `ADDRESS_ELEMENT` elements which defines the fields in the address tag. The `ADDRESS_ELEMENT` element has a `LABEL` attribute which refers to properties contained in the `CDEJResources.properties` file. This file is located in your `<client-dir>/JavaSource/curam/omega3/i18n` folder. The address is then built using `ADDRESS_ELEMENT` tags which must be given a name and label. Note that a code table can also be specified for each `ADDRESS_ELEMENT`. When a code table is specified, a drop-down list will display the code table entries and the default code will be pre-selected.

The optional `MANDATORY` attribute specifies if an address element is required to be filled in. The Mandatory indicator is an asterisk beside the field label as shown in the example above. Please note, that in order for `MANDATORY` settings in `curam-config.xml` to work, the field supplying the

address data should be marked mandatory in application model.

8.19 Schedule View

The schedule view is used for any domain of the type SCHEDULE_DATA. This view displays a grid of time-line information for the hours between 8 am and 8 pm. Each row in this grid represents a person whose full name is displayed in the row header. Each cell in the person's row represents a half hour period containing an indicator for whether they are available or not. If a user clicks on a free cell, they should be linked to a page allowing them to enter further schedule events.

The information and setup of this particular view involves a particular setup in a page's UIM file. Example 8.16, *UIM Example of Schedule View* is an example of the UIM for a schedule field.

```
<FIELD>
  <CONNECT>
    <SOURCE NAME="ACTION" PROPERTY="schedule"/>
  </CONNECT>
  <CONNECT>
    <LINK PAGE_ID="IncomeScreening_confirmAppointment">
      <CONNECT>
        <SOURCE NAME="ACTION" PROPERTY="appointmentDate"/>
        <TARGET NAME="PAGE" PROPERTY="date"/>
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="ACTION" PROPERTY="userFullName"/>
        <TARGET NAME="PAGE" PROPERTY="fullUserName"/>
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="ACTION" PROPERTY="userName"/>
        <TARGET NAME="PAGE" PROPERTY="userName"/>
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="PAGE" PROPERTY="caseID"/>
        <TARGET NAME="PAGE" PROPERTY="caseID"/>
      </CONNECT>
      <CONNECT>
        <SOURCE NAME="PAGE" PROPERTY="pageDescription"/>
        <TARGET NAME="PAGE" PROPERTY="pageDescription"/>
      </CONNECT>
    </LINK>
  </FIELD>
```

Example 8.16 UIM Example of Schedule View

The Cúram page generator expects any schedule FIELD element to be followed by a LINK element which details the PAGE_ID of the page to go to when a free cell is clicked on. The following three CONNECT elements should be fields which provide the following attributes to the link: the date of the day in question (the time is appended to this date); the full name of the user; and the user's unique identifier. The order of these CONNECT elements is important or the schedule view will not contain the correct links.

The SCHEDULE_DATA domain is expected to be a list of user names and 32 bit schedule fields separated by a tab. An example of one such element of this list would be:

John Smith<tab>16777212

Please note that 16777212 is the integer value which translates to the bit field 000000001111111111111111111100. A one represents a half hour when Mr. Smith is busy and a zero stands for free time. The bit field is read from the least significant bit first, i.e. from right to left, with 8 am represented by the right-most bit. As we are dealing with a twelve hour period and each bit stands for a half hour, only the first 24 bits are important. The last byte is disregarded.

The rendered widget is displayed as series of horizontal rectangular blocks (per user), with each block representing half an hour. Half hour blocks of free time are displayed differently than the other blocks (busy) in terms of color and size.

8.20 Radio Button Group

An alternative way to present a set of code table values is as a radio button group, each radio button representing a code table item. To display in the form of radio buttons, a field representing a code table value should be mapped to the `SHORT_CODETABLE_CODE` domain or to a domain directly inheriting from `SHORT_CODETABLE_CODE`.

8.21 Pop-up Pages

This section describes how to set up a pop-up page. The Cúram application has a number of built-in pop-up pages such as the Date Selector pop-up described earlier which are “helpers” used to enter data. Developers are also allowed to specify their own pop-up pages. For example, when scheduling a meeting for a person you don't want the user to have to know or fill in that persons unique ID. Instead the user should be provided with a search facility or a pre-populated list of valid options they can select from. This is achieved in Cúram with pop-up pages.

The out-of-the-box pop-up widget has a input field (grey in color) with a search - in the form of a magnifying glass - and a clear icon beside it. When the user clicks on the search icon this will activate a pop-up page. The user can select an item from the pop-page which will populate the text input field on the pop-up widget.

The following sections describe the steps involved in creating a pop-up.

8.21.1 Configure the Pop-up Page

The first step is to configure the pop-up page. This is performed by the `POPUP_PAGES` element in `curam-config.xml`.

```
<POPUP_PAGES DISPLAY_IMAGES="true|false">
  <CLEAR_TEXT_IMAGE>Images/minus.gif<CLEAR_TEXT_IMAGE>
  <POPUP_PAGE PAGE_ID="PersonSearch"
    CREATE_PAGE_ID="RegisterPerson"
    CONTROL_TYPE="textunderline|textinput"
    CONTROL_EDITABLE="true|false">
```



```

CONTROL_INSERT_MODE="overwrite|insert|append">
<DOMAIN>PERSON_ID</DOMAIN>
<WIDTH>800</WIDTH>
<HEIGHT>600</HEIGHT>
<SCROLLBARS>>true</SCROLLBARS>
<IMAGE>Images/search.gif</IMAGE>
<LABEL>Search</LABEL>
<CREATE_IMAGE>Images/new.gif</CREATE_IMAGE>
<CREATE_LABEL>New</CREATE_LABEL>
</POPUP_PAGE>
</POPUP_PAGES>

```

Example 8.17 Pop-up Configuration Example

On the root element the `DISPLAY_IMAGES` attribute can be used to configure whether images or text is used for the actions which open a pop-up or clear the currently selected value.

The nested elements are:

`CLEAR_TEXT_IMAGE` : The location of the image to use as a “clear this text” button. Note that this is an application wide setting.

`POPUP_PAGE` : For each domain definition which requires a pop-up there must be instance of this element. Up to two pop-ups can be associated with a single domain; one to search for an existing item, another to create a new item. The following attributes and child elements control various aspects of how the pop-up is presented to the user.

Name	Description
<code>PAGE_ID</code>	Specifies the UIM page id of the pop-up page to open to search for an existing item.
<code>CREATE_PAGE_ID</code>	Specifies the UIM page id of the the pop-up page to open to create a new item.
<code>CONTROL_TYPE</code>	Specifies the type of control where the value returned from the pop-up will be written to. The default value is <code>textunderline</code> which displays static text with an underline. To display a text input field set the value to <code>textinput</code> . When a a text input control is configured, on the UIM <code>FIELD</code> which uses a pop-up, the <code>HEIGHT</code> attribute can be used to change from a single line text input to a multi-line text area.
<code>CONTROL_EDITABLE</code>	This attribute is only valid when <code>CONTROL_TYPE</code> is set to <code>textinput</code> . It controls whether the text input field is editable or not. Set to <code>true</code> to create a editable field and <code>false</code> to create a non-editable field. Note that Internet Explorer does not give any visual indication that the text input field is not editable.
<code>CONTROL_INSERT_MO</code>	This attribute is only valid when <code>CONTROL_TYPE</code> is set to <code>textinput</code> . It allows you to configure

Name	Description
DE	how a value selected from a pop-up is inserted into the associated input control. The default is <code>overwrite</code> which means the selected value will overwrite the previous contents. Setting the attribute to <code>insert</code> means the selected value will be inserted at the current cursor position. Setting the attribute to <code>append</code> means the selected value will be appended to the previous contents of the input control.

Table 8.10 Attributes of the **POPUP_PAGE** element.

Name	Description
DOMAIN	Domain used to identify this pop-up page. If a FIELD element with a TARGET connection is based on this domain, a pop-up will be used instead of a standard text entry box.
CT_CODE	This is a second way to identify a pop-up page. The attribute contains a code table code value and is used when associating multiple pop-up pages with a single field and is described in further detail below.
WIDTH	Width in pixels of pop-up dialog. This element is optional. If not included, the default width of 600 pixels will be used.
HEIGHT	Height in pixels of pop-up dialog. This element is optional. If not included, the height will be automatically calculated based on the page contents.
IMAGE	Location of image which when clicked launches the pop-up defined by the POPUP_PAGE element's PAGE_ID attribute.
IMAGE_PROPERTY	Optional key in the <code>CDEJRe-sources.properties</code> file under which the locale-specific location of the pop-up launcher image otherwise specified by IMAGE attribute is stored. If the IMAGE is also specified for the same configuration, it will take precedence over the IMAGE_PROPERTY and this attribute will be ignored.
HIGH_CONTRAST_IMAGE	Location of the high contrast image which when clicked launches the pop-up defined by the POPUP_PAGE element's PAGE_ID attribute.
HIGH_CONTRAST_IMAGE_PROPERTY	Optional key in the <code>CDEJRe-sources.properties</code> file under which the locale-specific location of the pop-up launcher image otherwise specified by HIGH_CONTRAST_IMAGE attribute is stored. If the

Name	Description
	HIGH_CONTRAST_IMAGE is also specified for the same configuration, it will take precedence over the HIGH_CONTRAST_IMAGE_PROPERTY and this attribute will be ignored.
LABEL	Alternate text for the image defined by the IMAGE element. If the POPUP_PAGE element's DISPLAY_IMAGES attribute is set to false, this text will be displayed instead of the image.
LABEL_PROPERTY	Optional key in the CDEJRe-sources.properties file under which the locale-specific value of the label attribute otherwise specified by the LABEL attribute is stored. If LABEL is also specified for the same configuration, it will take precedence over the LABEL_PROPERTY and this attribute will be ignored.
CREATE_IMAGE	Location of image which when clicked launches the pop-up defined by the POPUP_PAGE element's CREATE_PAGE_ID attribute.
CREATE_IMAGE_PROPERTY	Optional key in the CDEJRe-sources.properties file under which the locale-specific location of the pop-up launcher image otherwise specified by CREATE_IMAGE attribute is stored. If the CREATE_IMAGE is also specified for the same configuration, it will take precedence over the CREATE_IMAGE_PROPERTY and this attribute will be ignored.
CREATE_LABEL	Alternate text for the image defined by the CREATE_IMAGE element. If the POPUP_PAGE element's DISPLAY_IMAGES attribute is set to false, this text will be displayed instead of the image.
CREATE_LABEL_PROPERTY	Optional key in the CDEJRe-sources.properties file under which the locale-specific value otherwise specified by the CREATE_LABEL attribute is stored. If the CREATE_LABEL is also specified for the configuration, it will take precedence over the CREATE_LABEL_PROPERTY and this attribute will be ignored.

Table 8.11 Child elements of the **POPUP_PAGE** element.

8.21.2 Create the Pop-up Page

A Cúram pop-up page is written in UIM. It can be written to display a set of

existing items for the user to select from or to register a completely new item.

A pop-up which returns existing items

The following is an example of a pop-up page which accepts user input, displays a list of search results, one of which can be selected and its unique identifier returned to the parent page.

```
<PAGE PAGE_ID="Person_search" POPUP_PAGE="true">
  <PAGE_TITLE ICON="PersonSearchPageIcon">
    <CONNECT>
      <SOURCE NAME="TEXT"
        PROPERTY="PageTitle.StaticText1"/>
    </CONNECT>
  </PAGE_TITLE>
  <SERVER_INTERFACE NAME="ACTION"
    CLASS="Person"
    OPERATION="search"
    PHASE="ACTION"
  />
  <CLUSTER NUM_COLS="2" TITLE="Cluster.Title.SearchCriteria">

    <ACTION_SET ALIGNMENT="CENTER" TOP="false">
      <ACTION_CONTROL LABEL="ActionControl.Label.Search"
        TYPE="SUBMIT" DEFAULT="true">
        <LINK PAGE_ID="THIS"/>
      </ACTION_CONTROL>
      <ACTION_CONTROL LABEL="ActionControl.Label.Cancel"
        IMAGE="CancelButton" TYPE="DISMISS"/>
    </ACTION_SET>

    <FIELD LABEL="Field.Label.ReferenceNumber">
      <CONNECT>
        <TARGET NAME="ACTION"
          PROPERTY="personSearchKey$referenceNumber"/>
      </CONNECT>
    </FIELD>
  </CLUSTER>

  <LIST TITLE="List.Title.SearchResults">
    <CONTAINER LABEL="Container.Label.Action">
      <ACTION_CONTROL LABEL="ActionControl.Label.Select"
        TYPE="DISMISS" >
        <LINK>
          <CONNECT>
            <SOURCE NAME="ACTION" PROPERTY="dtls$personID" />
            <TARGET NAME="PAGE" PROPERTY="value" />
          </CONNECT>
          <CONNECT>
            <SOURCE NAME="ACTION"
              PROPERTY="dtls$personFullName" />
            <TARGET NAME="PAGE" PROPERTY="description" />
          </CONNECT>
        </LINK>
      </ACTION_CONTROL>
    </CONTAINER>
    <FIELD LABEL="Field.Title.ReferenceNumber">
      <CONNECT>
        <SOURCE NAME="ACTION" PROPERTY="dtls$referenceNumber"/>
      </CONNECT>
    </FIELD>
    <FIELD LABEL="Field.Title.FirstName">
      <CONNECT>
        <SOURCE NAME="ACTION" PROPERTY="dtls$personName"/>
      </CONNECT>
    </FIELD>
  </LIST>
</PAGE>
```

The points to note about this example are:

- The `PAGE_ID` attributes of the `UIM PAGE` element and the `POPUP_PAGE` element in `curam-config.xml` must match.
- The `POPUP_PAGE` attribute of the `UIM PAGE` element must be set to `true`.
- The submit action is linked to `THIS`. This means the page will be redisplayed after the submit button is pressed.
- To cancel the pop-up an action control of type `DISMISS` is used. If the action control does not have a child `LINK` element, the pop-up will be closed without returning any values to the parent page which opened it.
- The search results list in this example is made up of three columns. The first contains a link which will close the pop-up and return the selected values, the remaining columns display further information about the person.
- To close the pop-up and return values, an action control of type `DISMISS` is used. This is placed in a `CONTAINER` so it is the first column in the search results list. The user can click this link to select one of the search results.
- To specify what values should be returned a child `LINK` element is added to the action control. When used in an action control to close a pop-up all standard attributes of the `LINK` element (e.g. `PAGE_ID`) have no meaning and will be ignored.
- For Cúram pop-up pages two values must always be returned. These are specified using `CONNECT` elements. Both connections must use a target of `PAGE` and have the `PROPERTY` set to `value` and `description`. The `value` connection specifies the value required on the page that opened the pop-up, in this example the person's unique record ID. The `description` connection specifies descriptive text to be shown to the user, in this example the person's name. So, on the page which opened the pop-up, the person's name will be displayed to the user, but it is their unique ID which will be submitted to the server.

It is not necessary for pop-up pages to accept input. For example, the `LIST` can be populated from a display phase server interface if necessary.

A pop-up which creates a new item

A pop-up may also create a new item and have the newly generated unique identifier for that item returned to the parent page. To do this create a page which a `ACTION_CONTROL` of type `SUBMIT_AND_DISMISS` must be used. For example;

```
<ACTION_CONTROL TYPE="SUBMIT_AND_DISMISS" LABEL="Button.Submit">
  <CONNECT>
    <SOURCE NAME="ACTION" PROPERTY="dtls$personID" />
    <TARGET NAME="PAGE" PROPERTY="value" />
  </CONNECT>
</ACTION_CONTROL>
```

```

</CONNECT>
<CONNECT>
  <SOURCE NAME="ACTION"
    PROPERTY="dtls$personFullName" />
  <TARGET NAME="PAGE" PROPERTY="description" />
</CONNECT>
</ACTION_CONTROL>

```

Once the type attribute is set to `SUBMIT_AND_DISMISS` the rules for the child `LINK` and `CONNECT` element is the same as described in the previous section for a `DISMISS` action control. After the form is successfully submitted the pop-up will be dismissed and the new values returned to the parent page.

8.21.3 Using the Pop-up Page

Pop-up pages are opened using standard UIM `FIELD` elements. If the field has a target connection which is based on a domain as configured in `curam-config.xml` a link to open the pop-up will be generated rather than a standard text entry field. This is illustrated in the screen shot above with the “Preferred Office” input field.

The following is the most basic example of a `FIELD` opening a pop-up. It is from an insert page so only a target connection is specified. Using the current example, the person's unique ID will be assigned to the field specified in the target connection and the person's name will only be used for visual purpose to display to the user.

```

<FIELD LABEL="Field.Label.person">
  <CONNECT>
    <TARGET NAME="ACTION" PROPERTY="personID"/>
  </CONNECT>
</FIELD>

```

Example 8.18 Opening a Pop-up from an Insert Page

The following example is from a modify page which means the field will have a source value which must be displayed to the user. It is slightly more complex than standard fields on a modify page because there are actually two source values to be handled. The person's unique ID and the person's name. In this case the `INITIAL` connection is used to specify the person's name. This will only be used to display to the user and note that it is not submitted to the server. Following that the field is just like any other on a modify page. The source connection specifies the existing value of the field, the target connection specifies where the value should be submitted to.

```

<FIELD LABEL="Field.Label.person">
  <CONNECT>
    <INITIAL NAME="DISPLAY" PROPERTY="personName" />
  </CONNECT>
  <CONNECT>
    <SOURCE NAME="DISPLAY" PROPERTY="personID" />
  </CONNECT>
  <CONNECT>
    <TARGET NAME="ACTION" PROPERTY="personID"/>
  </CONNECT>
</FIELD>

```

Example 8.19 Opening a Pop-up from a Modify Page

When invoking a pop-up it is also possible to supply page parameters to the pop-up. This is a slight variation on the two examples above and involves the use of the `LINK` element. The following is an example of two parameters passed to a pop-up page, one sourced from an existing page parameter, the other from a server interface property. When a `LINK` element is used in this context no attributes such as `PAGE_ID` should be specified. Also a `TEXT` source connection cannot be used to supply a parameter to a pop-up page.

```
<FIELD LABEL="Field.Label.person">
  <CONNECT>
    <TARGET NAME="ACTION" PROPERTY="personID"/>
  </CONNECT>
  <LINK>
    <CONNECT>
      <SOURCE NAME="PAGE" PROPERTY="personID"/>
      <TARGET NAME="PAGE" PROPERTY="param1"/>
    </CONNECT>
    <CONNECT>
      <SOURCE NAME="DISPLAY" PROPERTY="personName"/>
      <TARGET NAME="PAGE" PROPERTY="param2"/>
    </CONNECT>
  </LINK>
</FIELD>
```

Example 8.20 Supplying Parameters to a Pop-up Page

8.21.4 Using Multiple Pop-up Search Pages for a Single Field

In some cases we need to search for different types of Cúram entities but that search is associated with a single field. For example you may have a requirement to search for a Cúram client which has a generic domain of `CURAM_CLIENT_ID`. This could be a person, an employer, a product provider etc. Individual search pages may already exist for these types so you should be able to reuse them. Assuming the pop-up search pages already exist, this involves two extra steps which are described in the following sections and. The resulting pop-up widget is as described in Section 8.21, *Pop-up Pages* except that there is an additional drop-down field rendered to the left of the text input field. In order to activate the pop-up page for this widget, the user first selects the type of search to be performed from the drop down list and then clicks on the search icon.

8.21.5 Configure the Multiple Pop-up Page

This can be configured through the `MULTIPLE_POPUP_DOMAINS` element in `curam-config.xml`. The following is an example:

```
<MULTIPLE_POPUP_DOMAINS>
  <CLEAR_TEXT_IMAGE>Images/clear.gif</CLEAR_TEXT_IMAGE>
  <MULTIPLE_POPUP_DOMAIN>
    <DOMAIN>CURAM_CLIENT_ID</DOMAIN>
    <LABEL>Search</LABEL>
    <IMAGE>Images/search.gif</IMAGE>
```

```
</MULTIPLE_POPUP_DOMAIN>
</MULTIPLE_POPUP_DOMAINS>
```

Example 8.21 Multiple Pop-up Domains

The nested elements are:

CLEAR_TEXT_IMAGE : The location of the image to use as a “ clear this text” button. This is an application wide setting.

MULTIPLE_POPUP_DOMAIN : For each domain which you wish to associate multiple pop-up windows create an instance of this element.

DOMAIN : The name of the domain which is associated with multiple pop-up windows

IMAGE : Location of image to be used for pop-up icon.

LABEL : Alternate text to be used for pop-up icon.

As shown above, when using multiple pop-up pages a drop-down list is required to select the pop-up type. This drop-down list is populated as normal from a code-table. The code-table codes are the link between the drop-down list and pop-up that is opened. This requires the **CT_CODE** child element of the **POPUP_PAGE** element to be set to the code-table code value.

8.21.6 Using the Multiple Pop-up Page

Once the configuration is done the final step is the write the UIM necessary to display the pop-up search.

```
<CONTAINER LABEL="Label.person">
  <FIELD LABEL="Field.Label">
    <CONNECT>
      <TARGET PROPERTY="popupType" NAME="ACTION"/>
    </CONNECT>
  </FIELD>
  <FIELD LABEL="Field.Label">
    <CONNECT>
      <TARGET PROPERTY="clientID" NAME="ACTION"/>
    </CONNECT>
  </FIELD>
</CONTAINER>
```

Example 8.22 UIM to Use Multiple Pop-up Windows

The main points to note are:

- A **CONTAINER** and two **FIELD** elements are required, one for the drop-down list, the other for the value which will be returned from the pop-up. The container must not include any other **FIELD** elements.
- The first field should be based on a code-table domain which contains a list of codes which corresponds to the **CT_CODE** element described earlier.
- The second field should have a target connection which is based on a domain using the **MULTIPLE_POPUP_DOMAIN** element.

8.22 Agenda Player

The Agenda Player (or player for short) is a wizard-like control which provides guided navigation through a specified set of screens. As the name implies the screens in the Agenda Player are supposed to be part of a certain agenda or scenario, most typically involving step-by-step collecting of information.

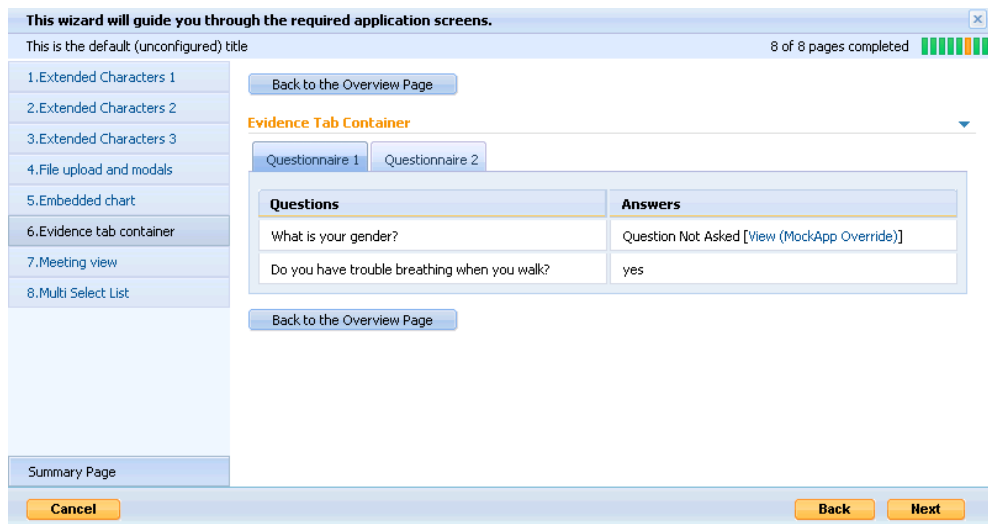


Figure 8.13 Agenda Player Example



Note

Agenda Player widget is not supported outside the modal dialog context, an attempt to open it in the tab content panel or elsewhere (e.g., as the inline page of an expandable list) will lead to an explicit error message stating this.

8.22.1 Agenda Player screen structure

Depending on how the Agenda Player player is configured, the screen is divided into either three or four parts:

- Along the top is the Agenda Player header. It contains a customizable Agenda Player title on the left and, where appropriate, a progress bar on the top right, which shows the user's progress through the agenda. The steps completed in the progress bar will be shaded in color whereas the steps that have yet to be completed will not. See Figure 8.13, *Agenda Player Example* for more details.
- On the left of an Agenda Player, a navigation panel (optional) shows the list of pages in the current agenda. The user's progress through the sequence is continuously displayed there (in addition to progress bar) by highlighting of the current page. The appearance and behavior of the

other pages in the agenda depends on the mode used (see below). The pages in an agenda can be grouped into sections and the player provides the ability to collapse and expand visited sections.

At the bottom of the navigation panel is the summary link, which allows users to jump directly to the player summary page (they would also get there by navigating through all the pages in the agenda). The summary link is only displayed if there is an appropriate element specified in the agenda XML. The navigation panel is not displayed in the navigator-less (claimant) view of the Agenda Player.

- Along the bottom, a set of buttons is displayed to allow the user to step forward and back through the Agenda Player. There are also buttons to jump to the summary page (displayed optionally) and to quit the Player. See Figure 8.13, *Agenda Player Example* for more details.



Note

The text used for these buttons can be customized (see below). However, for the remainder of this section they are further referred as the *Back*, *Next*, *Finish* and *Cancel* buttons, which are their default captions.

- The main area of the screen is the content area. This area displays normal client pages which might also be used outside of the Agenda Player.

8.22.2 Navigation modes

In addition to using the back and next buttons to navigate through an agenda, the player can provide additional options in the navigation panel, depending on the mode used.

The Agenda Player can be configured to operate in one of three navigation modes: `basic`, `incremental` or `full`, with `incremental` mode being the default.

- The `basic` mode is used for strictly sequential navigation through the agenda pages. In this mode the navigation panel is just used for additional information, indicating which page the user is currently on. The only navigation means are the standard player buttons.
- The `incremental` mode expands on the `basic` mode by providing links in the navigation panel to any pages which have already been visited. A user can use these links to skip back and forward between previously visited pages, but will still need to use the next button to progress any further.
- The `full` mode is actually a non-sequential mode as all the navigation panel elements are initially rendered as links. Sequential advancing is possible here as well, as the player buttons are fully functional, but there are no restrictions placed on the order in which you navigate through the agenda. This, however, means that things related to the sequential pro-

gress might be unavailable, or not work properly in this mode (for example, the progress bar is not displayed for this mode at all; dynamic parameters might not be available if a screen which expects these parameters is visited before the one where these parameters are initialized, etc.). Because of this the `full` navigation mode should be used where specifically required and the agenda should be designed/configured keeping in mind the possible consequences.

Agenda Player mode configuration is described in Section 8.22.4, *Agenda Player Configuration*



Note

Within the Player screens there might be hyperlinks leading to other pages, which open in the client area, yet do not belong to the specified Player screen set. In this case all the navigation means on the Player, including buttons and links rendered for `incremental` or `full` mode are disabled until the flow returns back to an Agenda Player screen. This means in particular that such a 'side' page should provide means of returning to the AgendaPlayer page flow (by linking to the appropriate page or closing the modal opened from the Player).

8.22.3 Navigator-less View

By default, an Agenda Player is displayed with all the screen parts present. However, in some situations, you may like to simplify the view and behavior of the player using the view without the navigation panel (also called Claimant view after the expected usage - i.e. online claimants). In this view Agenda Player is displayed without the navigation panel. Only the standard player buttons can be used for navigation, so the mode setting is effectively ignored.

The fourth player button, *Finish*, is automatically available on the button bar at the bottom of the page (see Figure 8.13, *Agenda Player Example* for more details) for the Claimant view. The button makes it possible to jump directly to the summary page rather than having to advance to it through all the pages. It is shown only when there is a summary page present in the agenda XML returned from the server.

Player configuration to allow for Claimant view is described in the section below.

8.22.4 Agenda Player Configuration

The Agenda Player can be configured by adding/modifying entries in `AgendaConfig.xml`. A version of this file should be in your `components` directory.

The following is an example of the Agenda Player configuration file contents:

```
<AGENDA>
  <PLAYER ID="DefaultConfig" TITLE="Default.Title"
    MODE="incremental" CONFIRM-QUIT="false" />
  ...
  <PLAYER ID="Claimant.Config" TITLE="Claimant.Title"
    NAVIGATOR-HIDDEN="true" MODE="incremental"
    CONFIRM-QUIT="true" />
</AGENDA>
```

The attributes that can be used for particular configuration (PLAYER element) are as follows.

Attribute	Description
ID	The ID of this particular configuration (referred to by CONFIG attribute of FIELD element in UIM which contains Agenda Player).
TITLE	Title key for Agenda Player title, displayed on its header. This key is used to look up customized/localized title from appropriate properties file as described in Section 8.22.5, <i>Agenda Player Customization</i> .
MODE	This attribute allows for specifying Agenda Player navigation mode. It might have values of <code>basic</code> , <code>incremental</code> or <code>full</code> , <code>incremental</code> being the default one, used if the attribute is skipped in an configuration.
NAVIGATOR-HIDDEN	When this attribute is specified and set to <code>true</code> , Agenda Player will be displayed in Claimant View (see above).
CONFIRM-QUIT	This attribute can be used to display a confirmation dialog when a user clicks on the <i>Cancel</i> button. When present and set to <code>true</code> , a confirmation dialog will be displayed to confirm the user's intention to quit the Agenda Player or to cancel and return to the player.

Table 8.12 Attributes of the **PLAYER** element

8.22.5 Agenda Player Customization

The Agenda Player comes with support for customization/localization of certain elements. The elements which can be customized are the player title, Progress Bar text, the player button texts, the quit confirm dialog text and descriptions for each of the frames in the player.

Player related properties are kept in the files `<client-dir>/JavaSource/curam/omega3/i18n/CDEJResources.properties` and

`<client-dir>/components/<component_name>/AgendaPlayer.properties`. where `<component_name>` represents the name of the component where the customizations are being applied.

Player title is customized by specifying custom value under the key used for it in `AgendaConfig.xml` (see above). The value under the key is to be placed into `AgendaPlayer.properties`.

The Progress Bar text is customized within an Agenda Player header by modifying the `AgendaPlayer.properties` file to include values for the keys: `Progress.Bar.Prefix`, `Progress.Bar.Middle`, `Progress.Bar.Suffix`. Please note that all three keys must be present with blank values for unused ones in order to ensure clean rendering of the customized Progress Bar text. If this is not the case then a situation may occur where a non-blank default value is used instead of one undefined.

The text strings associated with Agenda Player control buttons are customizable in the file `CDEJResources.properties` and defined by properties `wizard.button.back.title`, `wizard.button.forward.title`, `wizard.button.finish.title`, and `wizard.button.quit.title`.

The frame descriptions are useful for users of screen readers but don't appear visually on the screen. The entries for frame description customizations in `CDEJResources.properties` are `wizard.frameset.title`, `wizard.header.frame.title`, `wizard.navigation.frame.title`, `wizard.content.frame.title`, `wizard.button.frame.title`.



Note

The Agenda Player was formerly known as the Wizard widget, so several attribute and property names still refer to wizard.

In order to change the default question in the quit confirmation dialog, the property `Quit.Dialog.Question` should be added/changed in `AgendaPlayer.properties`.

8.22.6 Player data

There are some specific UIM pages related with Agenda Player:

- **Navigation page:** Each Player requires a navigation page that will become the navigation panel of the Agenda Player. This page has two required characteristics. First, the root `PAGE` element has a `TYPE` of `SPLIT_WINDOW`. This indicates that the page will form part of a frame-set. Second, the page contains a field with a single source connection and domain type `AGENDA_XML`. This field supplies the Agenda Player with the list of pages, parameters and other information that drives the Agenda Player.
- **Summary page:** This page is optional and might just be a regular UIM

page. However, summary page, specifically displaying summary of visited and unvisited pages is also available. If this information is to be displayed in a summary page, a WIDGET element with TYPE attribute set to WIZARD_SUMMARY should be present among page elements.

- Exit page: This is a regular UIM page to which the user is forwarded after quitting the player.

The following is an example of the UIM used to specify the navigation page. It contains a single field which supplies the agenda XML data.

```
<PAGE PAGE_ID="WizardTest" TYPE="SPLIT_WINDOW">

  <PAGE_TITLE>
    <CONNECT>
      <SOURCE NAME="TEXT" PROPERTY="page.title"/>
    </CONNECT>
  </PAGE_TITLE>

  <SERVER_INTERFACE NAME="DISPLAY" CLASS="Agenda"
    OPERATION="getAgenda"/>

  <PAGE_PARAMETER NAME="agendaRef"/>

  <CONNECT>
    <SOURCE NAME="PAGE" PROPERTY="agendaRef"/>
    <TARGET NAME="DISPLAY" PROPERTY="key$agendaRef"/>
  </CONNECT>

  <CLUSTER SHOW_LABELS="false">
    <FIELD>
      <CONNECT>
        <SOURCE NAME="DISPLAY" PROPERTY="agendaXML"/>
      </CONNECT>
    </FIELD>
  </CLUSTER>

</PAGE>
```

The following is an example of a specific summary page:

```
<PAGE PAGE_ID="WizardSummary">

  <PAGE_TITLE>
    <CONNECT>
      <SOURCE NAME="TEXT" PROPERTY="Page.Title"/>
    </CONNECT>
  </PAGE_TITLE>

  <CLUSTER SHOW_LABELS="false" TITLE="Cluster.Title">
    <WIDGET TYPE="WIZARD_SUMMARY"/>
  </CLUSTER>

</PAGE>
```

The agenda data that drives the Player looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<agenda>
  <page-flow>
    <section description="First section"
      status="SCT1">
      <page id="Person_homePage" description="Home"
        status="SC1" initial="true"
        submitonnext="true"/>
    </section>
    <section description="Second section"
      status="SCT2">
      <page id="Person_listAddress" status="SC2"
        description="Addresses"/>
    </section>
  </page-flow>
</agenda>
```

```

<page id="Person_listBankAccount" status="SC1"
description="Bank Accounts"
submitonnext="true"/>
<page id="Person_listCommunication" status="SC3"
description="Communications"/>
<page id="Person_listTask" status="SC2"
description="Tasks"/>
<page id="Person_listCitizenship" status="SC2"
description="Citizenships"/>
<page id="Person_listFinancial" status="SC2"
description="Financial"/>
<page id="Person_listNote" status="SC4"
description="Notes"/>
</section>
<summary id="WizardSummary"
description="Summary Page"
close-on-submit="true"
status="SCT3"/>
</page-flow>
<parameters>
<parameter name="concernRoleID" value="101"/>
<parameter name="dynamicParam" value="0"/>
</parameters>
<exit-page id="Person_homePage">
<parameters>
<parameter name="concernRoleID" value="101"/>
</parameters>
</exit-page>
</agenda>

```

There is one page element per screen to be displayed in the Agenda Player. The attributes that can be used in this element are as follows.

Attribute	Description
id	The page id for the page (as set in the PAGE_ID of the PAGE element in the page's UIM definition).
description	The description of the page that will be displayed in the Navigation Panel.
status	A status code that is mapped to an image.
initial	Set to true if this is the page that should be displayed when the Agenda Player is first opened.
disableback	Set to true if the <i>Back</i> button should be disabled on this page.
disableforward	Set to true if the <i>Forward</i> button should be disabled on this page.
submitonnext	Set to true if the <i>Forward</i> button should submit the form on this page.
close-on-submit	This attribute applies to summary element only and allows for alternative way of quitting the player, as described below.

Table 8.13 Attributes of the **page** element

The important features to note are:

- The sequence of screens in the Agenda Player is exactly as listed in the agenda data.
- One of the pages in the Agenda Player can be marked as the start page by setting the `initial` attribute to `true`. When the Agenda Player is first displayed, this page will be loaded but it will still be possible to navigate back to previous pages. If the Player is configured to use `incremental` mode, pages prior to the initial pages on the navigation panel will be rendered as hyperlinks; for a `full` navigation mode all the page items except current one will be hyperlinks.
- In the XML sent back by the application server, the `page` elements might be contained within `section` elements or there might be no `section` element at all. The optional `summary` element, however, is to be always placed directly within `page-flow`.
- All pages in the Agenda Player take the same set of parameters or a subset thereof. These parameters are specified in the agenda data.
- Page parameters can also be dynamic. These parameters initially carry special value of 0 (note `dynamicParam` in the Agenda Player sample data above) and are intended to be initialized during user interaction with Agenda Player (e.g., user ID is only available after a user registers herself).
- The `exit-page` denotes the page which the user will be taken to when the *Cancel* button is clicked. This page will completely replace the Agenda Player and can be any page in the application with any parameters (matching those specified by `exit-page` parameter sub-elements in agenda XML from the server).
- When `submitonnext` is set for a page, the submit button on that page (there should only be one) will be hidden when it is displayed within the player. The player's *Next* button can be used to submit the form instead and will proceed to the next page if no validation error occurs. If there are validation errors, the page will return to itself displaying the validation errors on the top, as it would for any other application page.

To allow for pages where the record itself is optional (i.e. you could move on to the next screen without creating one), but some of the fields are mandatory, if you do try to create a record, the infrastructure will not perform mandatory field validations if no value has been entered/chosen for any field on the page. The appropriate server interface will still be called, so it is up to the application logic to work out what was intended (e.g. don't create a record, delete an existing record, etc.). This behavior only applies when using the `submitonnext` feature.

- The summary page can provide an alternative way to quit the Player. In order to do this, the summary page should contain a submit button, and the summary element in the agenda XML from the server should have `close-on-submit` specified and set to be `true`. If the user clicks on the submit button on such a summary page and the submit succeeds, the

player closes down and the user is forwarded to whatever page is specified by the link associated with the submit button.

- Each page can be assigned a status code using the `status`. These status codes can be anything at all as long as they are mapped in the `ImageMapConfig.xml` file under the domain `AGENDA_XML`. When the list of pages is displayed in the left column, each will have an icon attached corresponding to its status code.

The following is an example of mapping status codes to images the `ImageMapConfig.xml` file.

```
<domain name="AGENDA_XML">
  <locale name="en">
    <mapping value="SC1" image="Images/Wizard/status1.gif"
      alt="English text..."/>
    ...
    <mapping value="SC4" image="Images/Wizard/status4.gif"
      alt="English text..."/>
  </locale>
  <locale name="fr">
    <mapping value="SC1" image="Images/Wizard/status1.gif"
      alt="French text..."/>
    ...
  </locale>
</domain>
```

The appearance of the Agenda Player control buttons, the summary screen and the navigation is defined in CSS. For details, please see Section 3.12.11, *Cascading Stylesheets*.

The `UIM CONDITION` element allows for the conditional display of action controls, clusters or lists on a page that is displayed within an Agenda Player (see See Section 5.9.6, *CONDITION* for more details on the condition element). To hide/display elements based on whether the page is in an Agenda Player or not, the `NAME` and `PROPERTY` attributes can only have the values `CONTEXT` and `inWizard` respectively.

```
<ACTION_SET ...>
  <CONDITION>
    <IS_TRUE NAME="CONTEXT" PROPERTY="inWizard"/>
  </CONDITION>
  ...
</ACTION_SET>
```

Example 8.23 Condition example:

This indicates that the action set should be displayed only when that Action Set is on a page that is displaying within a Agenda Player.

8.23 LOCALIZED_MESSAGE Domain

The `LOCALIZED_MESSAGE` domain allows entries in a server message catalog to be displayed on a client screen. The domain is string based but expects the string to be formatted in specific way. The Cúram Server Development Environment (SDEJ) provides support for formatting a message catalog entry in this way so it can be returned to the client. See the *Cúram*

Server Developers Guide for full details on working with message catalogs.

Once the message catalog entry has been formatted on the server side it should be assigned to a field which is based on the `LOCALIZED_MESSAGE` domain and returned to the client. The message entry will be displayed according to the current locale and values will be assigned to the message placeholders.

8.24 Decision Assist: Decision Matrix Widget

8.24.1 Overview

The Decision Matrix widget is a control that is used to construct questionnaires. Refer to the *Decision Assist Administration Class and Widget Overview* chapter in the *Inside Cúram Decision Assist Guide* for more details.

Chapter 9

Custom Data Conversion and Sorting

9.1 Objective

This chapter describes how to customize the data formatting, parsing, validation and sorting behavior of a Cúram web application.

9.2 Prerequisites

You should be familiar with the concept of domain definitions described in the *Domain Definitions* chapter of the *Cúram Modeling Reference Guide*, the development of client application pages, and basic Java programming.

9.3 Introduction

Custom data conversion and sorting allows most aspects of the management of data in the presentation layer of Cúram applications to be customized. Customizations can control how data is formatted, parsed, validated and sorted; error reporting can also be customized and controlled. Operations are performed on data values according to a well-defined data life-cycle and, at each stage, the operations can be customized. To understand how, when, and where to customize the operations, you must first understand the operations available and how they fit into the life-cycle.



Unsupported Customizations

This chapter describes the supported mechanisms for the customization of data conversion and comparison operations. For completeness, and to aid understanding, some operations are described, but the corresponding customization mechanisms are not documented, as customization of these operations is not supported (or not supported using the programmatic mechanisms described here).

The descriptions of the Java interfaces and classes presented here

may be incomplete, as unsupported methods may be omitted from their descriptions for clarity. However, the JavaDoc documentation for these interfaces and classes may include more information and describe more comprehensive customization mechanisms, but only the mechanisms described here are supported.

9.4 Data Conversion and Sorting Operations

There are a number of operations that are carried out on data values by the client infrastructure. Some are controlled by the domain definition options that were set in the UML model and are performed automatically, others are controlled by domain-specific plug-ins that can be overridden and customized; these plug-ins will be described later. First, the operations that are performed on the data values need to be understood:

format

When data is retrieved from the application server, it is represented by a Java object appropriate to the root domain of the data. For example, a value in the `SVR_INT64` domain is represented as a `java.lang.Long` object. The *format* operation is responsible for converting these objects to their string representation, as it is the string representation that must be embedded in the XHTML stream returned to the web browser.

A format operation is only required to return a non-null string; there are no other limitations. However, each domain-specific formatter will usually return a string representation of the Java object according to the usual conventions. For example, a money value may have a currency symbol added during formatting and be limited to two significant digits after the decimal point. For most data values, the formatter should generate a string representation that can later be converted back into the original data value.

pre-parse

When a user enters values in a form on an application page and submits the form to the client application, the web browser submits all of these values in string format. These string values need to be parsed to create the appropriate Java object representations, but first a *pre-parse* operation is performed to prepare the string for parsing.

The UML model supports several domain definition options that are recognized by the pre-parse operation (see the *Cúram Modeling Reference Guide* for more information on domain definition options). The domain definition options may indicate that leading and trailing whitespace characters should be trimmed from the string, that all sequences of whitespace characters should be compressed to single space characters, and that the string should be converted to upper-case. The pre-parse operation applies these options automatically to the string values and the modified string values are then ready to be parsed. The pre-parse operation is controlled and customized by setting these domain

definition options in the UML model.

parse

After the pre-parse operation has completed, the *parse* operation must convert the resulting string value into its Java object representation before it can be submitted to the application server. In general, the parse operation is the reverse of the format operation. If the format operation formatted a money value to a string and added a currency symbol and grouping separator (e.g., thousands separator) characters, the parse operation must be able to remove these additions and create a Java object representation of the actual money value.

All that is required of the parse operation is to produce a Java object, it does not validate that value. However, while not explicitly a validation operation, the parse operation usually needs to perform some validation to ensure that the value can be parsed correctly. For example, a date may later be determined to be invalid if it is out of range, but the parse operation must first determine what the date value is and may fail if the string does not represent a date in any recognized format.

pre-validate

Like the pre-parse operation, the *pre-validate* operation is performed to apply domain definition options defined in the UML model. However, unlike the pre-parse operation, different domain definition options are applied to data values depending on the domain. The data is not modified. String and BLOB values are tested to ensure that they do not exceed their maximum or minimum defined sizes (or lengths), while numeric values are tested to ensure that they do not exceed their maximum or minimum values. Any failures will be reported as errors. See Table 9.5, *Behavior of the Pre-Validate Operations* for a detailed description of the actual validations performed.

validate

The pre-validate operation is convenient and is applied automatically, but there are situations where it may not be able to validate data sufficiently. The *validate* operation is a catch-all that allows any kind of validation to be performed that is not possible using UML domain definition options alone. For example, ID values may be tested to see if their check-digit is valid. Errors can be reported if any value does not meet such specific conditions. Data is not modified by this operation.

compare

When a list of data is returned from the server, the sort order of the values in the list is determined using the *compare* operation. This sort order is used to support the sorting of lists on application pages when users click on the column headers. The compare operation is passed two data values (in their Java object representations, *not* in their formatted string representations) and must return a positive or negative number to indicate which comes first in the sort order. Like the format operation, the compare operation is not restricted in what calculations it performs, but it will typically sort values alphabetically or numerically.

Each data conversion operation has access to information about the active user's locale and to information about the domain being processed. It is also possible for one operation to access and execute any of the operations should that be necessary. For example, a format operation might format values differently for each locale and a compare operation might invoke the format operation before making a comparison.

9.5 Data Conversion Life Cycle

The CDEJ infrastructure is responsible for the retrieval of data from the application server, the display of this data, the processing of user input, and the submission of data back to the application server. This process has a well-defined life cycle. Operations at each stage in the life cycle are performed in a domain-specific manner.

Not all data goes through each stage in the life cycle. Some data is displayed but not modified or resubmitted by the user (read-only); some data is created by the user and submitted without any initial value being retrieved from the application server (write-only); and some data is retrieved, modified by the user, and then resubmitted to the application server (read-write).

In the context of the value of a single property, the life cycle for reading the value is as follows:

Read-only Life Cycle

1. The value is fetched from the application server by invoking a business operation.
2. If the value is one of a list of values for the same property, the related values are sorted using the compare operation and the resulting sort order is recorded.
3. The value is formatted to a string representation by the format operation and is stored for later display.
4. When the page is displayed, the value is retrieved and inserted into the XHTML stream.

The life cycle for writing a value is as follows:

Write-only Life Cycle

1. A string representation of the value is entered on a form by the user and the value submitted.
2. The domain definition options for whitespace compression and trimming and for upper-case translation are applied to the string value by the pre-parse operation. The value remains in string form.
3. If the business operation has declared the value to be mandatory, the value is checked to ensure that it is not empty or `null`. An error will

be reported if this check fails.

4. The value is parsed from its string representation by the parse operation and the resulting native Java object replaces the string value.
5. The domain definition options for the size range, value range, and pattern match are applied by the pre-validate operation is applicable. The value is not modified by this operation. If a validation fails, an error will be reported.
6. The value is validated by the validate operation to apply any arbitrary validation rules. Again, the value is not modified by this operation and validation failures are reported.
7. The parsed and validated value is sent to the application server.

For a value that is treated as read-write, the life cycle is simply the combination of the read-only life cycle followed by the write-only life cycle.

9.6 The Domain Hierarchy and Domain Plug-ins

At each step in data life-cycle, knowledge of a value's domain is required to ensure that the correct processing is performed. Embedding this domain information in the application is one of the tasks performed by the application code generators. With this information available, the application can invoke data conversion and comparison operations tailored for each domain.

Not only is information about each domain available at run-time, information about the relationships between these domains is also available. A model of the *domain hierarchy* is maintained in memory using tree structures and all the necessary information about how values in the domains should be processed “hangs” from these trees.

The domain hierarchy is composed of nodes implementing the `curam.util.common.domain.Domain` interface. The main methods declared in this interface are listed below. For more information see the Cúram JavaDoc documentation for this interface.

- **getName()**. This method is used to get the name of this domain.
- **getParent()**. This method is used to get the parent domain of this domain if it exists.
- **getRootDomain()**. This method is used to get the ultimate root domain of this domain.
- **getChildren()**. This method is used to get the list of children of this domain.
- **getPlugIn()**. This method is used to get the named plug-in object associated with this domain.

For the purposes of writing custom data conversion and comparison opera-

tions, this interface is rarely used directly, but it is instructive of the mechanism by which custom code is integrated into an application.

Each domain has a unique name: the name defined for it in the UML model. As domains can be derived from other domains, parent-children relationships exist, and these are also represented. Also, the root domain (the ultimate ancestor of any domain) is readily accessible. A root domain is one that does not have a parent domain. Several root domains (for dates, strings, integers, etc.) are supported in the Cúram application, so the domain hierarchy is represented by a “forest” of separate trees, rather than a single tree. All information about a domain, other than its name and relationships to other domains, is provided via *domain plug-ins*.

As described in the list above, the `curam.util.common.domain.Domain` interface also describes a method for the retrieval of plug-ins, `getPlugIn`, that takes the name of the type of plug-in required. The method returns the plug-in configured for the domain or the equivalent plug-in configured for the nearest ancestor domain if none has been configured directly; this is the simple inheritance mechanism. Domain plug-ins are Java classes that implement the data conversion and comparison operations and other features that are specific to each domain. There are four supported plug-in types, each with a unique plug-in name:

“converter”

Converter plug-ins are responsible for implementing the format, prepare, parse, pre-validate, and validate operations for each domain. Converter plug-ins can be customized to influence the appearance of values on an application page, to support the parsing of new data formats, and to prevent the submission of invalid data.

“comparator”

Comparator plug-ins are responsible for implementing the compare operation for each domain. Comparator plug-ins can be customized to influence the sorting of data.

“default”

Default plug-ins are responsible for providing default values for each domain when no value is available. While this type of plug-in can be customized freely, there will rarely be any need to modify the implementations provided within the Cúram application.

“options”

Options plug-ins are responsible for providing access to the domain definition options that were defined in the UML model. This type of plug-in is built into the client infrastructure and cannot be customized.

The mechanism used to configure the domain plug-ins exploits the domain hierarchy to simplify the configuration dramatically: very few domains need to be configured, as domains that are not configured will inherit the configuration from their ancestor domains. Each root domain needs to be configured (so that every domain has an ancestor from which it can inherit its

configuration), and a small number of specialized sub-domains are also configured further (the most notable being `CODETABLE_CODE`, a derivative of the root domain `SVR_STRING`). In all, less than 1% of domains are directly configured, so the configuration information is very manageable. The Cúram application comes complete with plug-in implementations and configuration information for all the domains used by the reference application; modifications are only required to handle specialized custom extensions.

9.7 Overview of Domain Plug-ins

9.7.1 Common Features of Plug-ins

Domain plug-ins are just Java classes that conform to a well-defined interfaces. There is a base interface that describes common features of all domain plug-ins and more specialized interfaces for each type of plug-in. At run-time, the infrastructure co-ordinates instantiation and invocation of all plug-ins, so the process of writing plug-ins is straightforward: methods need to be implemented that perform the data conversion and comparison operations and very little else needs to be considered.

All plug-in classes implement the `curam.util.common.domain.DomainPlugIn` interface. This defines some common operations and provides access to basic information that the plug-in may require. The main methods declared in this interface are listed below. For more information see the Cúram JavaDoc documentation.

- **`getName()`**. This method is used to get the name of this plug-in (one of the four plug-in names described above).
- **`getLocale()`**. This method is used to get the locale associated with this plug-in instance.
- **`getDomain()`**. This method is used to get the domain applicable to this plug-in instance.
- **`getInstance()`**. The final method is used to get an instance of a domain plug-in; it is not invoked in custom code. Instantiation issues are described in more detail in Section 9.13.2, *Plug-in Instance Management*. You should use the default implementations of these methods provided by the Cúram plug-in classes.

The methods of the `DomainPlugIn` interface do not really do anything interesting. Derived interfaces define the specific operations that each type of plug-in performs.

9.7.2 Converter Plug-ins

The `DomainConverter` interface is the one most likely to be used for customizations. It defines several simple methods that perform the main

data conversion operations. They are listed as follows. For more information see the Cúram JavaDoc documentation for this interface.

- **format()**. This method is used to format the given object to a string representation.
- **parse()**. This method is used to parse the given string representation into an object.
- **validate()**. This method is used to validate an object according to the domain-specific constraints. It may throw an exception if the object is invalid, but does not modify the object or return any value.
- **getDomainClass()**. This method returns the class object that indicates the required type of the object that is passed to the other converter methods or returned by them.
- **getGenericLocale()**. This method is used to get the locale to be used when formatting or parsing generic values. This should be the “en_US” locale and you should not change this value; it does not matter if this locale is not otherwise used in your application.
- **formatGeneric()**. This method is used to format the given object to a generic string representation.
- **parseGeneric()**. This method is used to parse the given generic string representation into an object of the appropriate type for the associated domain.

As described above, the `formatGeneric` and `parseGeneric` methods are similar to the `format` and `parse` methods, but they are used when converting the values of the domain definition options entered in the UML model by developers or of values embedded in XML-based data. Domain definition option values—for example, maximum date values, minimum size values, or regular expressions used for pattern matching—are extracted from the UML model at build-time and are parsed to their Java object representations at run-time, so that they can be used when validating data entered by a user. A similar process is used when extracting values from XML data returned from the application server and when constructing XML data before it is returned to the application server. The default implementations of the `formatGeneric` and `parseGeneric` methods are sufficient for all purposes (see Section 9.13.4, *Generic Parse Operations* for information on protecting the generic parse operation from side-effects).

It is by implementing these converter methods or overriding existing implementations of them that most customizations are performed. The simple method signatures disguise the fact that, via the inherited `DomainPlugIn` interface, each method has access to the active user's locale and the full domain information if necessary.

Implementations of the pre-parse and pre-validate operations are provided for all of the root domains in the Cúram application. As these operations are controlled completely by the setting of domain definition options in the

UML model, there is rarely any need to customize them programmatically. However, there are circumstances where custom error messages are required, so you may need to “wrap” these operations to intercept and replace error messages (this is described in detail in Section 9.12.6, *Custom Error Reporting*). These operations are defined on a separate `ClientDomainConverter` interface. They are listed as follows. For more information about these methods, see the Cúram JavaDoc documentation for this interface.

- **`preParse()`**. This method prepares a string for parsing by applying the relevant domain options. For example, the string may have whitespace removed or compressed, or may be converted to upper-case. The locale is used for the conversion to upper-case, if that is required.
- **`preValidate()`**. This method performs the standard validation checks that are controlled by the domain options specified in the UML model. The checks include the maximum and minimum size, the maximum and minimum value, and the matching of a pattern. The specific data-type of the object will determine which of these checks are appropriate. The options and comparator are available from the domain.

Access to the `ClientDomainConverter` interface is only supported for the purposes of error message interception. However, as all converter plug-ins created for use by the client infrastructure must implement this interface, you must sub-class an existing converter plug-in class (or abstract class) when creating custom converter plug-ins to inherit an appropriate implementation.

9.7.3 Comparator Plug-ins

The `DomainComparator` interface is used to control sort orders and it extends the `DomainPlugIn` interface and the standard `java.util.Comparator` interface. For more information about `DomainComparator`, see the Cúram JavaDoc documentation.

The `java.util.Comparator` interface defines a `compare` method that takes two `java.lang.Object` arguments and returns an integer that is positive if the first argument comes before the second argument in the sort order, negative if it comes after, and zero if the objects are equal. (See the JavaDoc documentation for the `java.util.Comparator` interface for more details.) An `equals` method is also defined by that interface, but it is of lesser importance; all Java classes inherit an implementation of the `equals` method from `java.lang.Object` or from another ancestor class and no further implementation is necessary.

9.7.4 Default Value Plug-ins

The `DomainDefault` interface is used to define default values for domains where no default value is available. The main methods in this interface are listed as follows. For more information about these methods, see

the Cúram JavaDoc documentation for this interface.

- **getAssumedDefault()**. This method is used to get the default value that will be assumed when a user clears a field on a form and submits no value.
- **getDisplayedDefault()**. This method is used to get the default value that should be displayed when an input field has no initial value to display.

From the methods listed above, we can see there are two types of default value: the value assumed when no value is available to send to the application server, and the value displayed when no initial value has been defined for a form field on an application page. The two default values are often the same, but there are some cases where they need to be different.

The assumed default value is needed when a form is submitted and the form data contains no value for a field that was not defined to be mandatory. The web client never submits `null` data values to the application server, so it must assume some value for the field and then submit that. The assumed value is nearly always intuitive: zero for any kind of number, an empty string for any string value, or a zero date or date-time for such values. The actual assumed default values used in the Cúram application are listed in Table 9.7, *Out-of-the-Box Default Value Plug-ins*.

The displayed default value is needed when a form field has not been initialized with any other value (as is usual on forms used to create new entities). The UIM `FIELD` element has a `USE_DEFAULT` attribute that defaults to `true`, so, unless that attribute is set to `false`, the displayed default value may be used. The displayed default value for numbers and strings is usually the same as that used as the assumed default value, but for dates and times, the current date and time is used instead of the zero date and time. Like the assumed default values, the displayed default values are likely to be sufficient for most applications, so you are unlikely to need to customize them.

There is also a third source for default values: there is a domain definition option for a default value supported in the UML model. However, if no such option is set, it is the plug-in's displayed default value that is used as a fallback, so the two can be treated in the same way. If only the displayed default value needs to be customized, it is easier to do this using the UML domain definition option rather than writing and configuring a new plug-in class, but the assumed default value can only be modified via a plug-in.

The default code used for values in a code-table domain is controlled via the application's code-table administration interface. You should not attempt to control it programmatically.

9.8 Domain Plug-in Configuration

Domain plug-ins are configured by means of an XML configuration file. The format is simple: the file contains a `domains` root element; for each domain to be configured, a `domain` element is inserted; within that element, `plug-in` elements are used to specify the name of the type of plug-

in and the Java class that implements the operations of that type of plug-in. The domain elements are not nested within other domain elements to reflect the domain hierarchy. The configuration information is relatively “flat”; each entry configures a separate domain and the inheritance of plug-ins is determined automatically. Here is a sample of such a configuration file:

```
<dc:domains
  <dc:domain name="SVR_INT64">
    <dc:plug-in name="converter" class=
      "curam.util.client.domain.convert.SvrInt64Converter"/>
    <dc:plug-in name="comparator" class=
      "curam.util.client.domain.compare.SvrInt64Comparator"/>
    <dc:plug-in name="default" class=
      "curam.util.client.domain.defaults.SvrInt64Default"/>
  </dc:domain>
  <dc:domain name="INTERNAL_ID">
    <dc:plug-in name="converter" class=
      "curam.util.client.domain.convert.InternalIDConverter"/>
  </dc:domain>
</dc:domains>
```

Example 9.1 Sample Domain Configuration

The configuration elements are defined in the XML namespace shown above. In the example, the namespace declaration assigns the prefix “dc” to this namespace, so that prefix is used before the element names. While you *must* declare this namespace in your configuration file, you can declare it to be a default namespace and omit the prefix, or even use a different prefix, but you must not omit the namespace declaration.

The example shows the configuration of two domains (these are the actual default configurations for these domains, as provided in the out-of-the-box Cúram application). Three plug-ins are configured for the Cúram root domain SVR_INT64. This is a complete set of plug-ins, as the “options” plug-in is built-in and is never directly configured. All descendant domains of SVR_INT64 will inherit these plug-ins unless further configured. Such a configuration is provided for the INTERNAL_ID domain. This domain is a descendant of SVR_INT64, but a different converter plug-in is configured; the comparator and default plug-ins will be inherited from SVR_INT64. This particular configuration is used within the Cúram application to override the format operation for INTERNAL_ID values so that grouping separators are not used in the string representations of the integers. An integer formatted by the SvrInt64Converter plug-in as “1,234,567” will be formatted by the InternalIDConverter class as “1234567”. This ensures that values such as case identifiers (the CASE_ID domain is a descendant of the INTERNAL_ID domain) are not represented as ordinary numerical values, but as more abstract unique key values. However, sorting and the calculation of default values remains unchanged, as these plug-ins are not overridden and the inherited plug-ins will be used.

There is a master configuration file called domains-config.xml located in your CDEJ installation's lib/curam/xml/config folder. This file contains the complete domain configuration information for all of the Cúram root domains and some descendant domains. You must not make any

changes to this file; it is overwritten each time the development environment is upgraded. However, the information in this file is useful when you need to make customizations. You can override or extend any configuration setting in this file using the mechanism described here.

Domain plug-in configuration follows the typical pattern used for when configuring other aspects of application components. You create configuration files, place them in component folders, and the component order determines which parts of each file take precedence when the files are merged together. A single custom configuration results and this may override or extend the master configuration without limitation. The `domain` elements in the configuration are merged where they have the same domain name defined in the `name` attribute. The `plug-in` elements of the merged domains are then collected and those with the same `name` attribute value as an existing `plug-in` element take precedence over that setting. New domain configurations can also be introduced. If the newly configured domain has descendant domains, they will inherit the new configuration. When configuring plug-ins, the name returned by a plug-in's `getName` method must match the `name` attribute value defined on the `plug-in` element in the configuration file; this helps to avoid mistakes in the configuration file.

The configuration files that you place in your component folders must be named `DomainsConfig.xml` (a slightly different name to the master configuration file to prevent confusion of the two). You can create one or more of these files (one in each component), but a single file is probably sufficient for most purposes. The format is just that shown in the example above. Further configuration examples are included in Section 9.12, *Customization Guidelines*.

9.9 Out-of-the-Box Domain Plug-ins

9.9.1 Extending Existing Plug-ins

Domain plug-ins for all of the root domain definitions (and a few others) are provided in the out-of-the-box Cúram application. Rather than write your own plug-in implementation from scratch, it is far easier to extend one of these existing plug-ins. The supplied plug-ins are suitable for the majority of uses, but all can be overridden in whole or in part as necessary, or used as the basis for new plug-ins that customize the processing of values in new domains. The details of these supplied plug-ins and the behavior of their operations are described in the sections below.

Abstract plug-in classes are also provided to be used as the basis of new plug-ins. These abstract classes are used by the Cúram plug-ins themselves and provide some useful functionality that is rarely necessary to override. The abstract classes you might use are:

- `curam.util.client.domain.convert.AbstractConverter`

- `curam.util.client.domain.compare.AbstractComparator`
- `curam.util.client.domain.defaults.AbstractDefault`

Their behavior is as follows:

Abstract Plug-in Class	Behavior
<code>AbstractConverter</code>	<p>Returns the correct name for this type of plug-in: “converter”.</p> <p>Formats an object that is an instance of <code>java.lang.Number</code> using the standard Java locale-specific number format. Other object types are formatted by calling their <code>toString</code> method.</p> <p>Pre-parses an object by trimming leading and trailing whitespace, compressing sequences of spaces, and converting to uppercase if specified by the UML domain definition options for the domain.</p> <p>Does not implement any parse operation.</p> <p>Pre-validates an object by checking its maximum and minimum values if these are specified by the UML domain definition options for the domain.</p> <p>Validates an object by throwing a <code>java.lang.NullPointerException</code> if an object is null, but otherwise performs no validation.</p> <p>Performs generic parsing by invoking the ordinary parse operation that must be implemented in the sub-class. See Section 9.13.4, <i>Generic Parse Operations</i> for information on protecting the generic parse operation from side-effects.</p> <p>Performs generic formatting by invoking the object's <code>toString</code> method.</p> <p>Returns the correct value for the generic locale.</p>
<code>AbstractComparator</code>	<p>Returns the correct name for this type of plug-in: “comparator”.</p>
<code>AbstractDefault</code>	<p>Returns the correct name for this type of plug-in: “default”.</p> <p>Defines constants with suitable assumed de-</p>

Abstract Plug-in Class	Behavior
	<p>fault values for each of the root domains.</p> <p>Returns the displayed default value by looking up the default value defined in the UML domain definition options, or, if not found there, returns the assumed default value.</p> <p>Does not implement <code>getAssumedDefault</code>.</p>

Table 9.1 Behavior of the Abstract Plug-in Classes

These abstract classes are used by the Cúram plug-in classes and all extend the `curam.util.common.domain.AbstractDomainPlugIn` class. This class implements the locale and domain properties of the `DomainPlugIn` interface and also provides the plug-in instance management implementation that should be used by all plug-ins (see Section 9.13.2, *Plug-in Instance Management* for details).

While it is possible to write plug-ins from scratch, you should follow the guidelines presented in this chapter and extend either the existing plug-in classes or their abstract base classes. Other approaches cannot be supported due to the complexity of some features, such as instance management and generic parsing, that are best avoided and the default implementations used. Reusing these classes will also ensure that your code will be protected from changes to the plug-in interfaces, as default implementations of new interface methods will be inherited during upgrades and no custom code changes should be necessary.

9.9.2 Converter Plug-ins

Converter plug-ins implement the format, parse, validate, and related operations. The following converter plug-ins are provided out-of-the-box. While most are pre-configured against certain domains, others are left to be configured as described in Section 9.8, *Domain Plug-in Configuration* (all of the plug-ins are in the `curam.util.client.domain.convert` Java package):

Domain	Converter Plug-in Class
SVR_BLOB	SvrBlobConverter
SVR_BOOLEAN	SvrBooleanConverter
SVR_CHAR	SvrCharConverter
SVR_DATE	SvrDateConverter
SVR_DATETIME	DateTimeConverter
CURAM_TIME	CuramTimeConverter
SVR_DOUBLE	SvrDoubleConverter

Domain	Converter Plug-in Class
SVR_FLOAT	SvrFloatConverter
SVR_INT8	SvrInt8Converter
SVR_INT16	SvrInt16Converter
SVR_INT32	SvrInt32Converter
SVR_INT64	SvrInt64Converter
INTERNAL_ID	InternalIDConverter
SVR_MONEY	SvrMoneyConverter
SVR_STRING	SvrStringConverter
SVR_UNBOUNDED_STRING	SvrStringConverter
LOCALIZED_MESSAGE	LocalizedMessageConverter
CODETABLE_CODE	CodeTableCodeConverter
N/A	SvrInt8BareConverter
N/A	SvrInt16BareConverter
N/A	SvrInt32BareConverter
N/A	SvrInt64BareConverter

Table 9.2 Out-of-the-Box Converter Plug-ins

The format operations of these plug-ins determine the string representations of data values that appear on application pages. The format operations behave as follows:

Plug-in Class	Formatting Behavior
SvrBlobConverter	Formatted as base-64 encoded strings. The base-64 encoding scheme is defined in RFC 2045 [http://ietf.org/rfc/rfc2045.txt].
SvrBooleanConverter	Formatted as the string values <code>true</code> or <code>false</code> . These values are not locale-aware. Cúram application pages rarely display formatted Boolean values directly, instead, check-boxes are used or values are translated to locale-specific strings.
SvrCharConverter	Formatted as Unicode characters, not as numbers.
SvrDateConverter	Formatted using the application date format. If the format includes month or day names, these are localized using the active user's locale. If the date is the system “zero” date, an empty string is returned.

Plug-in Class	Formatting Behavior
<code>DateTimeConverter</code>	Formatted using the application date and time formats and the user's preferred time zone. If the format includes month or day names, these are localized using the active user's locale. If the date-time is the system "zero" date-time, an empty string is returned.
<code>CuramTimeConverter</code>	Formatted using the application time format. If the date-time is the system "zero" date-time, an empty string is returned.
<code>SvrDoubleConverter</code>	Formatted as numbers with grouping separator (e.g., thousands separator) and decimal point characters appropriate for the active user's locale.
<code>SvrFloatConverter</code>	Formatted in the same manner as the <code>SvrDoubleConverter</code> .
<code>SvrInt8Converter</code>	Formatted as numbers with grouping separator (e.g., thousands separator) characters appropriate for the active user's locale, but without any decimal point.
<code>SvrInt16Converter</code>	Formatted in the same manner as the <code>SvrInt8Converter</code> .
<code>SvrInt32Converter</code>	Formatted in the same manner as the <code>SvrInt8Converter</code> .
<code>SvrInt64Converter</code>	Formatted in the same manner as the <code>SvrInt8Converter</code> .
<code>InternalIDConverter</code>	Formatted as numbers in a non-locale-specific manner without grouping separator characters.
<code>SvrInt8BareConverter</code>	Formatted in the same manner as <code>InternalIDConverter</code> .
<code>SvrInt16BareConverter</code>	Formatted in the same manner as <code>InternalIDConverter</code> .
<code>SvrInt32BareConverter</code>	Formatted in the same manner as <code>InternalIDConverter</code> .
<code>SvrInt64BareConverter</code>	Formatted in the same manner as <code>InternalIDConverter</code> .
<code>SvrMoneyConverter</code>	Formatted in the same manner as the <code>SvrDoubleConverter</code> , but with exactly two significant digits after

Plug-in Class	Formatting Behavior
	the decimal point.
<code>SvrStringConverter</code>	Formatted literally, i.e., strings are not changed by the format operation.
<code>LocalizedMessageConverter</code>	Formatted by decoding the message information, localizing the string indicated by the message catalog details, and replacing any encoded string arguments. The active user's locale is used throughout.
<code>CodeTableCodeConverter</code>	Formatted as the code description corresponding to the code value using the active user's locale and the domain's associated code-table.

Table 9.3 Behavior of the Format Operations

Pre-parse operations are used to perform string-related operations, indicated by domain definition options set in the UML model, before the strings are parsed to their Java object representations. The operations performed are the same for all root domains and are as follows: trimming of leading whitespace, trimming of trailing whitespace, compression of sequences of whitespace characters to a single space character, and conversion to uppercase. The pre-parse operations should be customized via the domain definition options in the UML model. Customization of these options via domain plug-ins is not necessary and not supported.

Parse operations are used to interpret string values submitted from a form on an application page or via parameters to a URL and convert them to their Java object representations. The string values received from the web browser are interpreted as being in the UTF-8 encoding. This encoding is used when creating the Unicode Java strings that are passed to the parse operations. The parse operations behave as follows:

Plug-in Class	Parsing Behavior
<code>SvrBlobConverter</code>	Parsed as a base-64 encoded string.
<code>SvrBooleanConverter</code>	Recognizes any of <code>true</code> , <code>yes</code> , or <code>on</code> as Boolean <code>true</code> values, and any of <code>false</code> , <code>no</code> , or <code>off</code> as Boolean <code>false</code> values. The parsing is not case-sensitive or locale-aware. Other values are reported as errors.
<code>SvrCharConverter</code>	Parsed as a single Unicode character. The presence of extra characters is reported as an error.
<code>SvrDateConverter</code>	Parsed using the application date format and the active user's locale.

Plug-in Class	Parsing Behavior
<code>DateTimeConverter</code>	Parsed using the application date and time formats and the active user's locale. The user's preferred time zone is assumed.
<code>CuramTimeConverter</code>	Parsed using the application time format. The server's time zone is assumed.
<code>SvrDoubleConverter</code>	Parsed as a number with optional grouping separator characters and decimal point characters appropriate for the active user's locale.
<code>SvrFloatConverter</code>	Parsed in the same manner as <code>SVR_DOUBLE</code> values.
<code>SvrInt8Converter</code>	Parsed as a number with optional grouping separator characters appropriate for the active user's locale. The presence of a decimal point is treated as an error.
<code>SvrInt16Converter</code>	Parsed in the same manner as the <code>SvrInt8Converter</code> .
<code>SvrInt32Converter</code>	Parsed in the same manner as the <code>SvrInt8Converter</code> .
<code>SvrInt64Converter</code>	Parsed in the same manner as the <code>SvrInt8Converter</code> .
<code>InternalIDConverter</code>	Parsed in a non-locale-specific manner. Grouping separators are not permitted and for negative values the minus sign must be on the left.
<code>SvrInt8BareConverter</code>	Parsed in the same manner as the <code>InternalIDConverter</code> .
<code>SvrInt16BareConverter</code>	Parsed in the same manner as the <code>InternalIDConverter</code> .
<code>SvrInt32BareConverter</code>	Parsed in the same manner as the <code>InternalIDConverter</code> .
<code>SvrInt64BareConverter</code>	Parsed in the same manner as the <code>InternalIDConverter</code> .
<code>SvrMoneyConverter</code>	Parsed in the same manner as <code>SVR_DOUBLE</code> values, but the magnitude of the values are limited to $1e13$ to avoid the possibility of rounding errors.
<code>SvrStringConverter</code>	Parsed literally, i.e., strings are not changed by the parse operation.

Plug-in Class	Parsing Behavior
LocalizedMessageConverter	Parsed literally in the same manner as the SvrStringConverter. Localized messages are not supported as input values, so this parser is never invoked.
CodeTableCodeConverter	Parsed literally as a code value in the domain's associated code-table. An error is reported if the code is not defined in that code-table.

Table 9.4 Behavior of the Parse Operations

Pre-validate operations are used to perform validation checks, indicated by domain definition options set in the UML model, after values have been parsed to their Java object representations. The checks performed are not the same for all domains. The possible validation checks are: maximum size (length), minimum size (length), maximum value, minimum value, and pattern match. The maximum and minimum values are checked using the compare operation. The pre-validate checks applied as follows:

Plug-in Class	Max./Min. Size	Max./Min Value	Pattern Match
SvrBlobConverter	Yes	No	No
SvrBooleanConverter	No	Yes	No
SvrCharConverter	No	Yes	No
SvrDateConverter	No	Yes	No
DateTimeConverter	No	Yes	No
CuramTimeConverter	No	Yes	No
SvrDoubleConverter	No	Yes	No
SvrFloatConverter	No	Yes	No
SvrInt8Converter	No	Yes	No
SvrInt16Converter	No	Yes	No
SvrInt32Converter	No	Yes	No
SvrInt64Converter	No	Yes	No
InternalIDConverter	No	Yes	No
SvrInt8BareConverter	No	Yes	No
SvrInt16BareConverter	No	Yes	No
SvrInt32BareConverter	No	Yes	No
SvrInt64BareConverter	No	Yes	No
SvrMoneyConverter	No	Yes	No
LocalizedMessageConverter	Yes	No	Yes

Plug-in Class	Max./Min. Size	Max./Min. Value	Pattern Match
<code>SvrStringConverter</code>	Yes	No	Yes
<code>CodeTableCodeConverter</code>	Yes	No	No

Table 9.5 Behavior of the Pre-Validate Operations

The pre-validate operations should be customized via the domain definition options in the UML model. Customization of these options via domain plug-ins is not necessary and not supported.

The default implementations of the validate operations do not perform any extra validations.

9.9.3 Comparator Plug-ins

Comparator plug-ins implement the compare operations that determine the sort order of lists of values. Comparator plug-ins are provided for the following domains (all of the plug-ins are in the `curam.util.client.domain.compare` package):

Domain	Plug-in Class	Behavior
<code>SVR_BLOB</code>	<code>SvrBlobComparator</code>	Not sorted, as there is no useful sort order for these non-human-readable values.
<code>SVR_BOOLEAN</code>	<code>SvrBooleanComparator</code>	Sorted with Boolean <code>true</code> values before <code>false</code> values.
<code>SVR_CHAR</code>	<code>SvrCharComparator</code>	Sorted strictly numerically with no locale-aware processing.
<code>SVR_DATE</code>	<code>SvrDateComparator</code>	Sorted chronologically with the earliest date first.
<code>SVR_DATETIME</code>	<code>SvrDateTimeComparator</code>	Sorted chronologically with the earliest date-time first.
<code>CURAM_TIME</code>	<code>CuramTimeComparator</code>	Sorted chronologically with the earliest time first. <code>CURAM_TIME</code> is based on the <code>SVR_DATETIME</code> domain, so values may include date information, but for comparisons, the date part is ignored and only the time part is used to determine the sort order.

Domain	Plug-in Class	Behavior
SVR_DOUBLE	SvrDoubleComparator	Sorted numerically; smallest value first.
SVR_FLOAT	SvrFloatComparator	Sorted in the same manner as SVR_DOUBLE values.
SVR_INT8	SvrInt8Comparator	Sorted in the same manner as SVR_DOUBLE values.
SVR_INT16	SvrInt16Comparator	Sorted in the same manner as SVR_DOUBLE values.
SVR_INT32	SvrInt32Comparator	Sorted in the same manner as SVR_DOUBLE values.
SVR_INT64	SvrInt64Comparator	Sorted in the same manner as SVR_DOUBLE values.
SVR_MONEY	SvrMoneyComparator	Sorted in the same manner as SVR_DOUBLE values.
SVR_STRING	SvrStringComparator	Sorted lexicographically based on the numeric Unicode value of each character in the string. The comparison is not locale-aware.
SVR_STRING	SvrStringCaseInsensitiveComparator	Sorted identically to SvrStringComparator except the case is ignored.
SVR_STRING	SvrStringLocaleAwareComparator	Sorted according to the sorting rules defined by Java for the locale.
SVR_UNBOUNDED_STRING	SvrStringComparator	Sorted in the same manner as SVR_STRING values.
CODETABLE_CODE	CodeTableCodeComparator	Sorted according to the defined code-table sort order for the code values. If the defined sort orders are equal, the code descriptions are sorted lexicographically based on the numeric Unicode value of each character in the string. The comparison is not locale-aware.
CODETABLE_CODE	CodeTableCodeCaseInsensitiveComparator	Sorted identically to CodeTableCodeComparator except case is ignored.
CODETABLE_CODE	CodeTableCodeLocaleAwareComparator	Similar to the above, but

Domain	Plug-in Class	Behavior
DE	aleAwareComparator	the comparison of code descriptions uses the sorting rules defined by Java for the locale.

Table 9.6 Out-of-the-Box Comparator Plug-ins

The `SvrStringComparator` and `CodeTableCodeComparator` classes are configured by default to sort values in the `SVR_STRING` and `CODETABLE_CODE` domains respectively. If locale-aware sorting is required, the default plug-in configuration can be overridden to use the `SvrStringLocaleAwareComparator` and `CodeTableCodeLocaleAwareComparator` classes instead. If case-insensitive sorting is required, override using `SvrStringCaseInsensitiveComparator` and `CodeTableCodeCaseInsensitiveComparator`. See Section 9.8, *Domain Plug-in Configuration* above for details on overriding the default plug-in configuration. Using these locale-aware comparators, lists will be sorted according to the expected sorting rules of the active locale. However, applying these sorting rules takes more time, so there will be some performance degradation. The implementation of locale-aware sorting uses Java's built-in sorting rules, so the availability of correct sorting rules for each locale depends on the Java JRE being used.

9.9.4 Default Value Plug-ins

Default value plug-ins supply the default values used when no values are available. Default value plug-ins are provided for the following domains (all of the plug-ins are in the `curam.util.client.domain.defaults` package):

Domain	Plug-in Class	Assumed Value	Displayed Value
SVR_BLOB	<code>SvrBlobDefault</code>	Empty BLOB	Empty BLOB
SVR_BOOLEAN	<code>SvrBooleanDefault</code>	False	False
SVR_CHAR	<code>SvrCharDefault</code>	Character zero	Character zero
SVR_DATE	<code>SvrDateDefault</code>	Zero date	Current date
SVR_DATETIME	<code>SvrDateTimeDefault</code>	Zero date-time	Current date-midnight
SVR_DATETIME	<code>SvrDate-TimeDefaultCurrentTime</code>	Zero date-time	Current date - Current time
SVR_DOUBLE	<code>SvrDoubleDefault</code>	Zero	Zero

Domain	Plug-in Class	Assumed Value	Displayed Value
SVR_FLOAT	SvrFloatDefault	Zero	Zero
SVR_INT8	SvrInt8Default	Zero	Zero
SVR_INT16	SvrInt16Default	Zero	Zero
SVR_INT32	SvrInt32Default	Zero	Zero
SVR_INT64	SvrInt64Default	Zero	Zero
SVR_MONEY	SvrMoneyDefault	Zero	Zero
SVR_STRING	SvrStringDefault	Empty string	Empty string
SVR_UNBOUNDED_STRING	SvrStringDefault	Empty string	Empty string
CODETABLE_CODE	CodeTable-CodeDefault	Empty code string	Empty code string

Table 9.7 Out-of-the-Box Default Value Plug-ins

Within the Cúram application, the zero date and time is represented as midnight on January 1,0001; this is interpreted as if no date and time has been set at all.

Also, the default value for a code-table code is an empty code string; a different mechanism is used to define default code-table codes during code-table administration.

SvrDateTimeDefault plug-in is time zone aware and the displayed value it returns is offset by the difference between the user and server time zones. The configured converter plug-in is expected to also consider time zone settings and offset the value accordingly. The end result is that the time part of date-time value is set to midnight regardless the time zone settings.

9.10 Error Reporting

9.10.1 Exception Classes

Many customizations require the addition of exception handling and error reporting code. All the necessary infrastructure is provided to make this as simple as possible. A simple formulaic approach can be followed that will provide all of the necessary functionality. Before looking at how you can write customizations, you must first learn the necessary error reporting techniques.

All of the plug-in methods that throw exceptions, throw one of two exception types:

- `curam.util.common.domain.DomainException`

- `curam.util.client.domain.convert.ConversionException`

`ConversionException` is derived from `DomainException`, so instances of these exceptions can both be treated as `DomainException` objects when convenient. The `ConversionException` class is used for exceptions that are thrown by the methods of converter plug-ins. Unlike a `DomainException`, a `ConversionException` can be associated with a particular property of a server interface so that error messages reported to a user can indicate the label of the field in error and an error icon can be placed beside that field. The only exceptions that custom code normally needs to throw are instances of `ConversionException`, so this is the only exception class that needs to be understood to implement your own exception handling and reporting.

Conversion exceptions (and most other exceptions in the client infrastructure) carry information about the error message that needs to be reported, but not the error message itself. When an exception is thrown, the identifier of the localized error message string, the values that will be substituted for the placeholders in that string, and any causal exception object are included in the exception details. Each exception class can be associated with an error message catalog (a set of localized Java properties files) that is used when the localized message string is resolved from the message identifier. The localization and substitution steps are not performed until the message is reported to the user, so the exception can be propagated and augmented with more information for some time before the message string becomes fixed. This allows, in the case of conversion exceptions, the field label to be added automatically by the infrastructure after your custom code has thrown the exception and makes it very easy to integrate your error reporting requirements into the system.

9.10.2 Custom Exception Classes

The purpose of a custom exception class is to integrate the look-up of localized message strings in a custom message catalog into the mechanism used for error reporting in the client infrastructure. If you only need one error message catalog, you will only need one custom exception class, but there is no restriction on the number of exception classes or message catalogs you can create.

Implementing custom exception handling using a custom exception class is formulaic. As the custom exception class must integrate into the existing message reporting system, only numeric message identifiers are supported for custom exceptions and there is very little room for deviation from the prescribed approach. You cannot, for example, use literal message strings in your code, you must use references to externalized strings.

Here is an example of a custom exception class:

```
public class CustomConversionException
```

```

    extends ConversionException {

    private static final MessageLocalizer MESSAGE_LOCALIZER
        = new CatalogMessageLocalizer("custom.ErrorMessages");

    public CustomConversionException(int messageID) {
        super(messageID);
    }

    public CustomConversionException(int messageID,
        String[] messageArgs) {
        super(messageID, messageArgs);
    }

    public CustomConversionException(int messageID,
        String messageArg) {
        super(messageID, messageArg);
    }

    public MessageLocalizer getMessageLocalizer() {
        return MESSAGE_LOCALIZER;
    }
}

```

Example 9.2 Custom Exception Class

This class extends `ConversionException` and implements a number of constructors simply by invoking the equivalent constructors in the super-class. You only need to implement the constructors that you intend to use, the rest of the constructors in the super-class can be ignored (Java classes do not inherit constructors, hence the need to re-implement them). The available constructors are described in the JavaDoc. Next, it defines a static `MessageLocalizer` field and instantiates it with a `CatalogMessageLocalizer` object that takes your custom catalog name as its argument. The `getMessageLocalizer` method then returns this static object. That is all there is to it.

When you throw exceptions of this type, you need to pass your message identifier and optional arguments to the relevant constructor. You can define constants for your numeric message identifiers in this class if you wish. Your message strings can contain placeholders such as “%1s”, “%2s”, etc., to be replaced by the argument strings (only string types are supported). For an array of arguments, “%1s” will be replaced by the first argument in the array (index zero), and so on. The special argument “%0s” can be used to represent the name of the field in error, but you will not need to provide any matching argument string for that value; it will be substituted automatically. You can also use the same placeholder several times in a single message if you want the same value to be inserted in more than one place. Here is a sample message catalog file containing a single message:

```
-200000=ERROR: The field '%0s' contains an invalid value '%1s'.
```

Example 9.3 Custom Message Catalog

The file is a standard Java properties file where each line contains a numeric identifier and a message string separated by an equals character. A collection of properties files with the same base name but with locale codes ap-

pended is treated as a single message catalog. The custom exception class in the example above refers to the message catalog as “custom.ErrorMessage”, so the properties files should be located on the Java classpath in the custom package folder and in files named `ErrorMessages.properties`, `ErrorMessages_en_US.properties`, `ErrorMessages_fr_CA.properties`, etc., as you would do for any other custom properties files. There should be one properties file for each locale that your application supports. The selection of the correct locale-specific properties file at run-time is completely automatic once you have written your custom exception class as shown above.

Ensuring that these files end up on the classpath is simply a matter of placing them in their appropriate package folders below your web application's `<client-dir>/<custom>/javasource` folder, where `custom` is the name of a custom component. (see Section 3.6, *Project Folder Structure* for details). The Java source files for your custom exceptions should also be placed below the `<client-dir>/<custom>/javasource` folder in the appropriate folders for the package names you have used.

When throwing a custom exception, the code will look like this (assuming you have decided not to use constants for your error message identifiers):

```
throw new CustomConversionException(-200000, myInvalidValue);
```

Example 9.4 Throwing a Custom Exception

Remember, it is not necessary to pass any argument corresponding to the “%0s” placeholder; it will be calculated and substituted automatically.



Numeric Message Identifiers

When creating message catalog files, try to ensure that the error numbers do not conflict with the numbers of existing Cúram error messages, as this may cause confusion when errors are being investigated. Values below -200000 should be safe to use, though conflicting numbers will not actually cause any application problems, as the message catalogs are separate from those used by the infrastructure.

If you examine the constructors of the `ConversionException` class, you will note that many accept a `java.lang.Throwable` object as the last argument. You can implement similar constructors and pass `Throwable` objects (usually other exception objects) to your custom exceptions when you want your custom exception to include the exception that caused it. This is often very useful as error messages for both exceptions will be reported automatically and both stack traces will be included on an application error page if the error page is required. In fact, there is no imposed limit to the length of the chain of exceptions that can be built this way; the exception that you add to your own may already contain a reference to another exception, and so on.

This example show how you can even report two separate error messages at

once. Perhaps one is a generic message that states that a field does not contain a valid value and another suggests the expected format for that value. You will have to implement the appropriate constructor to support this, but the reporting mechanism is automatic.

```
throw new CustomConversionException(
    -200000, myInvalidValue,
    new CustomConversionException(-200003));
```

Example 9.5 Throwing Multiple Exceptions

9.10.3 Reusing Cúram Error Messages

It is possible to reuse existing Cúram error messages for your own purposes and avoid writing your own exception class, but this reuse is *not supported for upgrades*, as Cúram error messages are regularly modified and reorganized and your code could cease to function correctly if it depended on Cúram error messages that had been modified or removed. These internal changes within the Cúram application are not normally announced in any release notes. However, you may decide that the benefit of reusing the messages, and the relative ease of manually correcting problems introduced when upgrading to new Cúram releases, outweighs this lack of support during upgrades.

Each Cúram error message described in the Cúram Web Client Error Message Guide has an associated error name and number. This error name is the same as the name of a Java constant that is defined to be equal to the error number. While you can use the error number directly, it is safer to use the constant. All of the constants are implemented by the `ConversionException` class, so you can access them as static fields. Each error message may also contain optional placeholders for dynamic values that will be substituted for the placeholders when the error is reported. If you reuse an error message, you must provide a value for each placeholder it uses except the “zero” placeholder represented by “%0s”.

Here is an example of how you might reuse existing Cúram error messages:

```
// Error message taking one argument.
throw new ConversionException(
    ConversionException.ERR_CONV_PARSE_FAILED,
    myInvalidValue);

// Error message taking two arguments.
throw new ConversionException(
    ConversionException.ERR_CONV_OUT_OF_RANGE,
    new String[] { MY_MAX_VALUE, MY_MIN_VALUE });
```

Example 9.6 Reusing Cúram Error Messages (Unsupported)

9.11 Java Object Representations

The data conversion and comparison operations manipulate strings and other Java objects. Each value in a root domain is represented by an object of a

corresponding Java class. The Java class used by a root domain is the same for all descendant domains of that root domain and cannot be changed. When customizing the operations, knowledge of the type of data being processed is important. The table below shows the Java class used for data objects for each of the root domains.

Domain	Java Class
SVR_BLOB	curam.util.type.Blob
SVR_BOOLEAN	java.lang.Boolean
SVR_CHAR	java.lang.Character
SVR_DATE	curam.util.type.Date
SVR_DATETIME	curam.util.type.DateTime
SVR_DOUBLE	java.lang.Double
SVR_FLOAT	java.lang.Float
SVR_INT8	java.lang.Byte
SVR_INT16	java.lang.Short
SVR_INT32	java.lang.Integer
SVR_INT64	java.lang.Long
SVR_MONEY	curam.util.type.Money
SVR_STRING	java.lang.String
SVR_UNBOUNDED_STRING	java.lang.String
CODETABLE_CODE	curam.util.common.util.CodeItem

Table 9.8 Classes Used for Java Object Representations

Though derived from SVR_STRING, the Java class used for CODETABLE_CODE is different to that of its parent. This is the only exception to the rule that the Java class used is the same for all descendant domains of a root domain.

9.12 Customization Guidelines

9.12.1 Where to Start

Most customizations aim to control one or more of the data conversion or sorting operations. Guidelines are provided in the following sections to show you how each of these operations can be customized. Following these guidelines will ensure that your customizations are as simple and effective as possible.

When you have written your custom plug-ins, you need to configure them and ensure that the Java classes are available at run-time. Configuration was described in Section 9.8, *Domain Plug-in Configuration*. The Java source

files for your custom plug-in classes are added to the web application in exactly the same way as the Java source code files for your custom exception classes (see Section 9.10.2, *Custom Exception Classes*): they are placed in their appropriate package folders in your `<client-dir>/<custom>/javasource` folder, (where `<custom>` is the name of a custom component).

9.12.2 Custom Formatting

Custom formatting may be required when a value displayed on an application page is not in the required format. A custom formatter might be used to pad values with extra characters, so that they appear to be the same length; insert a currency symbol into money values; format numeric values without grouping separator characters; or even take a date value based on the Gregorian calendar and format it after converting it to another calendar system.

Guidelines for Custom Formatting

1. Identify an existing converter plug-in class that you want to customize. It will most likely be the converter that is already configured for the domain in question or inherited by it from an ancestor domain.
2. Create a new sub-class of the relevant converter plug-in and override the `format` method.
3. In the implementation of the method, you can perform some processing before or after invoking the super-class's method of the same name, or implement the formatting code from scratch.
4. Configure your new plug-in for the relevant domains.

The calendar scenario is somewhat unrealistic because the date selector widget would not be compatible, but inserting a currency symbol, or an analogous operation, is something that you may want to do. If multiple currencies are supported, then domains such as `US_DOLLAR_AMOUNT` or `EURO_AMOUNT` might be used to represent values in each currency (though the out-of-the-box Cúram application uses a different scheme for representing money values in different currencies). Custom converter plug-ins may then be written to format money values for each of these domains by adding the appropriate currency symbol.

This example shows how a converter plug-in can be written that takes a money value and prefixes the formatted numeric value with a dollar symbol. The out-of-the-box Cúram application comes with a converter plug-in that formats money values, but without any currency symbol, so you can reuse its format operation to simplify the implementation.

```
/**
 * Converter that supports the use of a dollar symbol for
 * money values.
 */
```

```

public class USDollarConverter
    extends SvrMoneyConverter {
    public String format(Object data)
        throws ConversionException {
        return "$" + super.format(data);
    }
}

```

Example 9.7 Custom Formatting for Currency Values

The implementation is very trivial: the super-class does all the work and returns a nicely formatted money value; the customization just adds the dollar symbol.

The configuration file for this customization is shown below. The file might also include entries for other customizations that have been made. As the comparator and default value plug-ins have not been customized, they do not appear in the configuration. These plug-ins will be inherited from the ancestors of the US_DOLLAR_AMOUNT domain (probably the SVR_MONEY domain).

```

<dc:domains xmlns:dc=
    <dc:plug-in name="converter"
                class="custom.USDollarConverter"/>
    </dc:domain>
</dc:domains>

```

Example 9.8 Configuration for Custom Formatting

Values displayed on an application page (or even those passed behind the scenes in hidden page connections) may be submitted back to the web application. If you write a formatter that inserts a currency symbol, or you allow users to insert currency symbols in values that they type in, then you will need to accommodate such values in the parse operation. The next section will demonstrate the custom parse operation required to match this custom format operation.

Another common need for custom formatting is to format integer values without grouping separator characters. For example, an integer value that represents the year “2005” should probably be formatted as “2005” and not “2,005”. If the year value is represented by the YEAR_VALUE domain and that domain is derived from the SVR_INT16 domain, the custom format operation would look like this:

```

/**
 * Converter that formats year values without adding grouping
 * separator characters.
 */
public class YearValueConverter
    extends SvrInt16Converter {
    public String format(Object data)
        throws ConversionException {
        return data.toString();
    }
}

```

Example 9.9 Custom Formatting without Grouping

This converter overrides the `format` method of the `SvrInt16Converter` class and simply converts the `data` object (a `java.lang.Short`) to a string. Unlike the routines used by the superclass, the `toString` method will not do any locale-aware formatting or add any grouping separator characters. The `parse` method is not overridden, so values that are entered with or without grouping separator characters will be accepted. This converter is configured in the same way that the currency symbol converter was configured.

9.12.3 Custom Parsing

Custom parsing is implemented when users must enter values in a form that existing parse operations do not recognize or when some other processing must be performed on values before they are submitted to the application server. Custom parsing may be as simple as a routine that first removes a currency symbol from a numeric value before parsing it, where the currency symbol may have been entered by a user or added by a custom format operation. It could also be something more unusual: a translation of a date to another calendar system, a routine that pads string values, or an arbitrary calculation on numeric values.

Guidelines for Custom Parsing

1. Identify an existing converter plug-in class that you want to customize. It will most likely be the converter that is already configured for the domain in question or inherited by it from an ancestor domain.
2. Create a new sub-class of the relevant converter plug-in and override the `parse` method.
3. In the implementation of the method, you can perform some processing before or after invoking the super-class's method of the same name, or implement the parsing code from scratch.
4. Configure your new plug-in for the relevant domains.

The currency symbol scenario is continued in this example to complement the example shown for a custom format operation above. The example below shows the same class developed to format money values with a currency symbol; the class is now extended with a corresponding parse operation. In a case like this, you do not write separate converter plug-ins for formatting and parsing; you must implement both operations in the same converter plug-in and then associate the plug-in with the appropriate domain.

```
/**
 * Converter that supports the use of a dollar symbol for
 * money values.
 */
public class USDollarConverter
    extends SvrMoneyConverter {
    public String format(Object data)
```



```

        throws ConversionException {
    return "$" + super.format(data);
    }

    public Object parse(String data)
        throws ConversionException {
        if (data.startsWith("$")) {
            return super.parse(data.substring(1));
        }
        return super.parse(data);
    }
}

```

Example 9.10 Custom Parsing for Currency Values

The value passed to the `parse` method is the same value that was entered by the user; it is possible that it contains no currency symbol or it might contain space characters between the currency symbol and the value. You can use the UML domain definition options to ensure that the pre-parse operation will have removed any whitespace before the currency symbol, or simply report an error if the currency symbol or a digit is not the first character. The `parse` method above assumes that the currency symbol is the optional first character and then leaves all other decisions up to the `parse` method of the super-class. This is probably the best approach, as it limits the number of formatting rules that a user needs to be aware of and keeps the code as simple as possible.

The configuration for this plug-in is unchanged from that shown for the custom format operation.

9.12.4 Custom Validation

Custom validation can be performed in two ways: by setting the domain definition options in the UML model, or by implementing a `validate` operation in a custom converter plug-in. It is also possible to combine both ways to meet your validation requirements.

The domain definition options in the UML model are limited to a small number of validations that are described in the *Cúram Modeling Reference Guide* and summarized in Table 9.5, *Behavior of the Pre-Validate Operations* above. If the domain definition options meet your needs, you should use them in preference to any programmatic alternative. If the options meet only some of your needs, you should use them and also create a custom converter plug-in to complete the validations. If the options are not useful, you should create a custom converter plug-in and implement all the validations there. Some uses for custom validation routines might include the validation of check digits or the imposition of any other arbitrary restrictions on the permitted values.

Guidelines for Custom Validation

1. Identify an existing converter plug-in class that you want to customize. It will most likely be the converter that is already configured for the domain in question or inherited by it from an ancestor domain.

2. Create a new sub-class of the relevant converter plug-in and override the `validate` method.
3. In the implementation of the method, invoke the super-class's method of the same name to perform any existing validations (if that is appropriate).
4. Complete the implementation by performing your validations and throwing an exception if any validation fails.
5. Configure your new plug-in for the relevant domains.

In this example, a new converter plug-in is created that extends the `InternalIDConverter` plug-in with a validation that only permits even numbers. The `InternalIDConverter` is derived from the `SvrInt64Converter` class that is configured for use by the `SVR_INT64` domain. Values in this domain are represented by `java.lang.Long` objects.

```
/**
 * Reports ID numbers as invalid if they are odd.
 */
public class EvenIDConverter
    extends InternalIDConverter {
    public void validate(Object data)
        throws ConversionException {
        // Perform any existing validations first.
        super.validate(data);

        if (((Long) data).longValue() % 2 != 0) {
            throw new CustomConversionException(-200010);
        }
    }
}
```

Example 9.11 Custom Validation for Odd Numbers

The error message entry in the custom message catalog may look like this:

```
-200010=ERROR: The field '%0s' must be an even number.
```

Example 9.12 Custom Validation Failure Message

If this validation is to be applied to the `EVEN_ID` and the `NOT_ODD_ID` domains, then the configuration will look like this:

```
<dc:domains xmlns:dc=
  <dc:domain name="EVEN_ID">
    <dc:plug-in name="converter"
      class="custom.EvenIDConverter"/>
  </dc:domain>
  <dc:domain name="NOT_ODD_ID">
    <dc:plug-in name="converter"
      class="custom.EvenIDConverter"/>
  </dc:domain>
</dc:domains>
```

Example 9.13 Configuration for Custom Validation

9.12.5 Custom Sorting

When lists of values are displayed in an application page, a user can sort the list by clicking on the column headers. The sort order of the rows will be determined by the sort order of the values in the selected column. Successive clicks on a column header alternate between the forward and reverse sort order for that column. The sort order for any type of data can be customized easily, though the sort-order for code-table codes must be controlled using the code-table administration interface. The sort order is calculated when responding to a user's request, so the user's active locale is available by calling the inherited `getLocale` method and can be used to influence the sort order in a locale-specific manner.

The domain comparator plug-ins are responsible for making the comparisons that control the sort order. The sorting algorithms swap the position of values in their value lists depending on the value returned by the `compare` method of the plug-in. The comparator plug-ins used in the Cúram application behave as described in Section 9.9.3, *Comparator Plug-ins*. These sort orders are simple and intuitive, but may not meet the needs of some specialized domains. In these cases, custom sort orders may be required and there is no limitation on what order can be used.



What Values are Compared?

All compare operations are performed by invoking the comparator plug-ins `compare` method. This takes two `java.lang.Object` arguments. The method is invoked automatically by the client infrastructure *before the values are formatted*. This means that the objects passed are of the types shown in Section 9.11, *Java Object Representations*, not formatted string representations of the values.

In most cases, having access to Java object representations makes the comparisons much easier to perform: comparing dates and numbers is much easier when they are represented by objects that conveniently provide a `compareTo` method that returns the same values that the `compare` method must return. However, there are some situations where, for example, encoded strings are decoded by the format operation and comparing them before they are formatted is not simple or would involve the duplication of the formatting code. In these cases, it is possible to invoke the appropriate formatter and compare the results. This will be described later.

The general guidelines for implementing a custom comparator plug-in to control the sort order for a domain are as follows:

Guidelines for Custom Comparators

1. Create a new sub-class of the `AbstractComparator` class described in Section 9.9.1, *Extending Existing Plug-ins*.
2. Implement the `compare` method to perform your custom comparison.

3. Configure your new plug-in for the relevant domains.

To illustrate this, you will see how to write a comparator that compares string values as if they were numbers. Some of the entities in the Cúram application use a string-based domain for their key values to support the use of identifiers that may not just contain digits. Sorting of these types works well in most cases, but there can be problems. Because the base domain is a string, the values are sorted lexicographically, not numerically. If the values are all of the same length, this is not a problem, but if the lengths differ, the sorting becomes confusing. For example, the string values “22” and “33” will be sorted into the order “22”, “33”, but if the values are “22” and “3”, the sort order will be “22”, “3”, because the character “2” comes before the character “3” in a lexicographical sort and representations of numbers with positional digits are not recognized.

There are a number of ways to solve this problem:

- The string values could be stored in the database with leading zeros used to pad all values to the same length, this would trick the lexicographical sorting into working correctly (the lexicographical sort order for “22” and “03” is “03”, “22”). If the leading zeros were not desired for display purposes, they could be stripped by the format operation and replaced by the parse operation. Legacy data, however, would need to be updated to conform to the new format.
- Write a custom comparison routine that parses the numeric values from the strings and then performs the comparison. This would work fine, but the parsing is a little complicated and it may be complicated further if the values have trailing check letters or other non-digit characters.
- Pad the value with zeros for the purposes of making the comparison, but do this inside the compare operation, so that no other application changes are necessary.

The latter solution is, perhaps, the easiest to achieve. Here is an example of a custom comparator plug-in that does this for values that are limited to no more than ten characters:

```
/**
 * Compares string values after padding them with leading
 * zeros to make the sorting work correctly for numeric
 * values. Values must not be longer than ten characters.
 */
public class IDComparator
    extends AbstractComparator {
    public int compare(Object s1, Object s2) {
        return _pad((String) s1).compareTo(_pad((String) s2));
    }

    private String _pad(String s) {
        return "0000000000".substring(0, 10 - s.length()) + s;
    }
}
```

Example 9.14 Sorting Strings Numerically

The `_pad` method pads a value with leading zeros, so that all returned strings will be ten characters long and numeric values will be compared correctly as the positional digits will all be aligned correctly. No change needs to be made to the format or parse operations or to any existing values in the database; the sort order is entirely controlled by this simple comparator code. While the numeric values could have been parsed from the strings and a numeric comparison made, this sample code is much simpler and more efficient.

Another need for custom sorting arises when values are in an encoded form that is decoded by the format operation. In this case, sorting of the encoded form may not be meaningful. For example, if a domain exists that uses an encoded string containing several localized messages and their locale codes like this “en|Hello|es|Hola”, calculating the sort orders for such strings is meaningless. The string could be decoded, but, as decoding must be done by the format operation, it is simpler to invoke the format operation instead and compare the values that it returns.

```
/**
 * Compares two encoded message strings using their
 * formatted values.
 */
public class MessageComparator
    extends AbstractComparator {
    public int compare(Object value1, Object value2) {
        final DomainConverter converter;

        try {
            converter = ((ClientDomain) getDomain())
                .getConverter(getLocale());
            return converter.format(value1)
                .compareTo(converter.format(value2));
        } catch (Exception e) {
            // Do nothing except report the values to be equal.
            return 0;
        }
    }
}
```

Example 9.15 Sorting Formatted Values

This code retrieves the converter plug-in that implements the format operation for the same domain as that of the values being compared. The returned converter will also be aware of the active user's locale. The exact mechanism behind this is unimportant, simply copying the code above is all that is needed. Other uses of the `ClientDomain` class are not supported. The exception handling is simple: it does nothing. The `compare` method is not declared to throw exceptions, and thrown run-time exceptions trigger an application error page, so there is not much useful error handling that can be performed. The reason that none is attempted at all is that if the converter cannot be retrieved or the format operation fails, it will be for reasons beyond the control of the comparator plug-in and these reasons will cause failures in other places that will be reported in time. In fact, the sorting operation is carried out just before the infrastructure formats all of the values ready for display, so the very next operation will detect and report the errors that may have been ignored by the comparator.

A final example shows how to make the Cúram application zero date (January 1,0001), appear after all other dates instead of before them:

```
/**
 * Compares dates, but places the zero date at the end,
 * rather than the start, or the sort order.
 */
public class ZeroDateComparator
    extends AbstractComparator {
    public int compare(Object value1, Object value2) {
        final Date date1 = (Date) value1;
        final Date date2 = (Date) value2;

        if (Date.kZeroDate.equals(date1)
            && !Date.kZeroDate.equals(date2)) {
            return -1;
        } else if (!Date.kZeroDate.equals(date1)
            && Date.kZeroDate.equals(date2)) {
            return 1;
        }
        return date1.compareTo(date2);
    }
}
```

Example 9.16 Sorting Zero Dates

The comparator returns a negative number (the magnitude is not important) if the first date is the zero date and the second date is not the zero date to indicate that the first date comes after the second in the sort order. Likewise, a positive number is returned if the first date is not the zero date and the second date is the zero date to indicate that the order is correct. Otherwise, the dates are compared as normal. This causes the zero date to be positioned after all other dates instead of before them in the sort order.

This type of manipulation should be used with caution: the comparator plugins are also used during pre-validation to check a value against the maximum and minimum values defined for its domain in the UML model's domain definition options. In this case, if the UML domain definition options define a maximum date and no date is set, then the zero date will be assumed and this will appear to be later than all other dates, including the maximum date, and the pre-validation check will always fail with an error. If no maximum value is specified in the model, then this comparator will work without problems.

To override the default comparator for all dates with this new comparator, the configuration will look like this:

```
<dc:domains xmlns:dc=
  <dc:domain name="SVR_DATE">
    <dc:plug-in name="comparator"
      class="custom.ZeroDateComparator"/>
  </dc:domain>
</dc:domains>
```

Example 9.17 Configuration for Custom Sorting

Now, all date values for all domains that are descendants of the root SVR_DATE domain, and values in the root domain itself, will be sorted ac-

ording to the new rules. There is no need to configure any other domains, as they will all inherit this new comparator (unless, of course, a descendant domain has been configured with another comparator that will override any inherited comparator). This comparator could also be applied more selectively to descendant domains of `SVR_DATE`.

9.12.6 Custom Error Reporting

It is possible that a plug-in performs the operations exactly as you require, but you need to customize the error reporting. One area of the Cúram application where this may happen is in the pre-validation operation when the pattern matching option is applied. A pattern is a regular expression defined in the UML model. When this validation fails, the error reports that the data was “not in a recognized format”, as few users would be able to interpret the meaning of a regular expression if presented to them. If the format is a common and intuitive one (a phone number, say), then this message will probably suffice. However, if the format is more obscure, the error message may need to be changed to present a human-readable description of the format (correctly localized). There are two ways to achieve this:

- Remove the pattern option from the UML model and implement your own pattern match validation as you would for any type of custom validation.
- Intercept the exception from the pre-validation operation and replace it with a different exception carrying your alternative error message.

A custom validation is possible and you will just need to follow the usual guidelines for such a customization, but it is complicated by the need to access the pattern information and perform the pattern matching operation. As you would then need to report your custom error message, it is much simpler to let the existing infrastructure do all the pattern matching and just focus on the error message.

Custom error reporting is really only applicable to the `parse` and `preValidate` methods of a converter plug-in. These are the only methods that may be invoked and passed values that a user has entered and that a user may be able to correct in response to an error message. The converter plug-ins supplied with the out-of-the-box Cúram application do not report any errors from their `validate` methods, so, unless you want to customize another custom converter plug-in, the `validate` method can be ignored.

Guidelines for Intercepting Exceptions

1. Identify the method that is generating the exception that carries the error message that you want to customize. The likely candidates are the converter plug-in's `parse` and `preValidate` methods.
2. Create a new sub-class of the relevant converter plug-in and override the appropriate method.
3. In the implementation of the method, invoke the super-class's method

- of the same name and catch any exception thrown.
4. Test the error number on the caught exception to ensure it is the one you want to override.
 5. If the error number is correct, throw a new exception carrying your error message, otherwise, re-throw the caught exception, as it is not the one you wish to override.
 6. Configure your new plug-in for the relevant domains.

This example shows how this might be done to override the pattern match failure message. The custom exception class described in Section 9.10.2, *Custom Exception Classes* is used.

```
/**
 * Reports that social security numbers must match the format
 * "xxx-xx-xxxx" when the regular expression defined in the
 * UML model "\d{3}\-\d{2}\-\d{4}" does not match a social
 * security number entered by a user.
 */
public class SSNConverter
    extends SvrStringConverter {
    public void preValidate(Object data)
        throws ConversionException {
        try {
            super.preValidate(data);
        } catch (ConversionException e) {
            if (e.getMessageObject().getMessageID()
                == e.ERR_CONV_NO_MATCH) {
                throw new CustomConversionException(-200001);
            }
            throw e;
        }
    }
}
```

Example 9.18 Custom Error Reporting

The error message entry in the custom message catalog will look like this:

```
-200001=ERROR: The field '%0s' must use the format 'xxx-xx-xxxx'.
```

Example 9.19 Custom Pattern Match Failure Message

Domains that require this converter can be configured in the same manner as shown for the other converters above.

The same warnings apply to the interception of error messages as those that apply to the reuse of error message (see Section 9.10.3, *Reusing Cúram Error Messages*): Cúram error messages are subject to change without notice. However, in the specific case of the pattern match failure message, the error -122128 - ERR_CONV_NO_MATCH will be preserved, as the possible need to intercept this message is recognized.

9.12.7 Custom Default Values

It is unlikely that you will ever need to customize a default value plug-in for

a domain. The displayed default value can be customized using the respective UML domain definition option. The predefined assumed default values for the domains are probably sufficient for every need. However, in the unlikely event that you need to customize an assumed default value, the steps are little different from those for other plug-ins.

Another reason for customizing a default value plug-in is where the displayed default value is not fixed and cannot be defined in the UML model. An example of this is the use of the current date as a displayed default value.

Guidelines for Custom Default Values

1. Identify an existing default value plug-in class that you want to customize.
2. Create a new sub-class of the relevant default value plug-in and override the `getDisplayDefault` method.
3. The implementation of the method should simply return a value compatible with the Java type used to represent values for the relevant root domain. These Java types are listed in Section 9.11, *Java Object Representations*.
4. Configure your new plug-in for the relevant domains.

In this example, the displayed default value for an interest rate is calculated dynamically using a notional `CentralBank` class that somehow returns the current interest rate.

```
/**
 * Returns the current interest rate by contacting the
 * central bank!
 */
public class InterestRateDefault
    extends SvrFloatDefault {
    public Object getDisplayedDefault()
        throws DomainException {
        try {
            return new Float(CentralBank.getInterestRate());
        } catch (Exception e) {
            throw new CustomDomainException(-200099, e);
        }
    }
}
```

Example 9.20 Custom Default Date-Time Value

The example assumes that the `InterestRateDefault` class will be associated with a descendant of the `SVR_FLOAT` domain that requires a default value to be of the `java.lang.Float` type. By extending the `SvrFloatDefault` class, the new default value plug-in will automatically use zero as the assumed default interest rate value.

The exception handling uses a `CustomDomainException` class. As the `getDisplayDefault` method throws a `DomainException`, and

not a `ConversionException`, you could create such a custom exception class by deriving it from `DomainException` in exactly the same way as the `CustomConversionException` class was derived from `ConversionException` in Section 9.10.2, *Custom Exception Classes*. You might note that, as the `DomainException` class is an ancestor of the `CustomConversionException` class that the `CustomConversionException` class could be used here instead. This will work, but you must not attempt to report a message containing the “%0s” placeholder for the field label, as automatic replacement of the field label is not supported when a `DomainException` type is expected.

The example above shows the unknown exception thrown by the `CentralBank` class being added to the new custom exception. You only need to implement the appropriate constructor to support this. The super-class already has a constructor with the same signature, so your constructor's implementation need only call that. There is no need to extract a string value or stack trace from the exception; all will be reported correctly when necessary.

9.13 Advanced Topics

9.13.1 Type Checking and Null Checking

You may have noticed that none of the examples in this chapter show the string or object values passed to the methods being checked to see if they are `null` or of the wrong type. The reason is that it is not necessary. The client infrastructure guarantees that no method will be called with a `null` value and that no conversion operation will be invoked for an object that is not compatible with the class returned by the converter plug-in's `getDomainClass` method. Your custom code need never include any error handling and reporting code for these checks.

9.13.2 Plug-in Instance Management

For efficiency, a Cúram client application pools the minimum number of domain plug-in instances possible. This reduces the overhead involved in creating new plug-in instances each time their operations are invoked, but it does impose some restrictions on the way plug-ins can be written.

Domain plug-ins maintain state information: a reference to the domain and the active user's locale. Custom code can access this state information by calling the `getDomain` and `getLocale` methods and use it as required. The potential for concurrent access to plug-ins in typical multi-threaded servers impacts the way the plug-in instances (with their state information) are managed. If concurrent requests are received from users who are using different locales, then the same plug-in instance cannot be used when servicing these requests, as only one locale value can be set in a plug-in instance. However, as any Cúram application only supports a finite number of locales, maintaining a single plug-in instance for each locale is sufficient to

avoid concurrency problems or synchronization overheads. This, of course, has to be multiplied by the number of domains, as the domain information also constitutes state. The result is that each domain in the domain hierarchy accesses a pool of plug-in instances specific to that domain and each pool contains one instance of each type of plug-in for each locale.

This instance management system is entirely driven by the plug-ins themselves. Each type of plug-in can implement its own instantiation strategy most appropriate to its needs. However, to avoid over-complicating instance management, the `AbstractDomainPlugIn` class (see Section 9.9.1, *Extending Existing Plug-ins*) implements the single, consistent pooling strategy that balances efficiency against other considerations.

While it would be more efficient to dispense with the domain and locale state information and pass these values to the various converter and comparator methods, this poses several other problems that make this approach less desirable:

- The method signatures would be complicated by values that may not be used.
- Some method signatures, such as the `compare` method of the `java.util.Comparable` interface would not be compatible.
- The addition of new state information in the future would break all existing implementations. Using accessor methods for state information allows the abstract super-classes to implement the accessors and the signatures of the other interface methods can remain unchanged. During an upgrade no changes would need to be made to any existing custom code that has followed the guidelines and extended these abstract super-classes or other classes derived from them.

It is this latter point that is most important, successful upgrades depend on custom code that does not attempt to implement the plug-in interfaces from scratch. This is why such an approach cannot be supported.

The pooling strategy used means that there is one main limitation on how plug-ins can be written: plug-ins must not attempt to store any state information. In short, *no customization should add fields to a plug-in class* and attempt to store information in them; concurrent application requests will probably cause such a plug-in to fail intermittently or introduce obscure bugs.

Domain plug-in classes must also provide a default constructor (i.e., a constructor that takes no arguments). However, any Java class that does not explicitly define a default constructor will automatically have one defined for it if the default constructor of an ancestor class is visible. For custom plug-in classes that extend the plug-in classes and abstract plug-in classes provided with the out-of-the-box Cúram application, no explicit default constructor is required.

9.13.3 Naming Conventions

Custom domain plug-in classes may implement utility methods to support the implementation of the main interface methods. An example is the `_pad` method shown in Example 9.14, *Sorting Strings Numerically*. To avoid inadvertently overriding another inherited method, or using a method name that conflicts with a method introduced in a later Cúram release, you should prefix such utility methods with an underscore character as shown. Underscore characters will not be used in the client infrastructure, so they will guarantee that no naming conflict will arise in the future. For similar reasons, do not create classes in packages that might conflict with Cúram package names. All Cúram packages begin with “curam”, so avoiding that name is sufficient. The examples in this chapter used the package name prefix “custom”, but this is not a requirement.

9.13.4 Generic Parse Operations

The generic parse operation, performed by the `DomainConverter` interface's `parseGeneric` method, needs some explanation, so that care can be taken not to disable its operation by mistake. The generic parse operation is responsible for parsing the string representation of values defined in the UML model's domain definition options. Domain options for maximum, minimum and default values are expressed in formats that are not locale-specific, as the UML model is not locale-aware. Each of the root domains accepts values in a particular format (e.g., ISO-8601 format for `SVR_DATE` domains) and customization of this format is not supported. Therefore, the default implementations of the `parseGeneric` method must be respected.

For some domains, the format supported by the converter's `parse` method is the same as the format supported by the `parseGeneric` method. The default implementation of the `parseGeneric` method in the `AbstractConverter` class just calls the `parse` method (which is not implemented in this class). Therefore, if you sub-class the `AbstractConverter` class and implement a `parse` method, the same implementation will be used by the `parseGeneric` method. This may be what you require, but, if it is not, you may want to implement a different `parseGeneric` method.

All of the out-of-the-box, concrete converter classes separate the implementations of the two methods, so you can override one without changing the behavior of the other. Again, this may be what you require, but, if it is not, you may want to override both methods.

9.13.5 Code-Tables

Data conversion and sorting for code-table domains should be managed via the code-table administration interface. While the client infrastructure uses the same plug-in mechanism described here to manage code-table values, the customization of code-table-related plug-ins is not supported. Code-table data is more complex to handle (formatting and parsing are not symmetrical operations as they are for other types) and all of the necessary customiza-

tions can be accomplished without resorting to programmatic means.

The formatting of code-table values is achieved by modifying the descriptions of each code. Parsing operations receive the code values and simply pass them on. Pre-parsing, pre-validation, and validation are not important. Default codes and custom sort orders are controlled entirely via the administration interface.

Appendix A

Unsupported Dynamic UIM features

A.1 Introduction

This appendix lists the elements and attributes (features) that are not supported in dynamic UIM.

A.2 PAGE

Name	Feature Type
FIELD	Child Element
CONTAINER	Child Element
WIDGET	Child Element
INCLUDE	Child Element
SHORTCUT_TITLE	Child Element
TAB_NAME	Child Element
JSP_SCRIPTLET	Child Element
SCRIPT	Child Element
SCRIPT_FILE	Attribute
POPUP_PAGE	Attribute
APPEND_COLON	Attribute
HIDE_CONDITIONAL_LINKS	Attribute
COMPONENT_STYLE	Attribute
TYPE	Attribute

Table A.1 Unsupported PAGE Features

A.3 PAGE TITLE

For full details on the supported features of this element in static UIM, see Section 5.9.27, *PAGE_TITLE*.

Name	Feature Type
DESCRIPTION	Child Element
ICON	Attribute

Table A.2 Unsupported PAGE_TITLE Features

A.4 CLUSTER

For full details on the supported features of this element in static UIM, see Section 5.9.5, *CLUSTER*.

Name	Feature Type	Supported/Unsupported attribute values
TITLE	Child Element	
DESCRIPTION	Child Element	
WIDGET	Child Element	
SUMMARY	Attribute	
TAB_ORDER	Attribute	

Table A.3 Unsupported CLUSTER Features

A.5 LIST

For full details on the supported features of this element in static UIM, see Section 5.9.23, *LIST*.

Name	Feature Type	Supported/Unsupported attribute values
TITLE	Child Element	
DESCRIPTION	Child Element	
FOOTER_ROW	Child Element	
ACTION_CONTROL	Child Element	
SUMMARY	Attribute	
SORTABLE	Attribute	

Name	Feature Type	Supported/Unsupported attribute values
PAGINATED	Attribute	
DEFAULT_PAGE_SIZE	Attribute	
PAGINATION_THRESHOLD	Attribute	

Table A.4 Unsupported LIST Features

A.6 FIELD

For full details on the supported features of this element in static UIM, see Section 5.9.11, *FIELD*.

Name	Feature Type
LABEL	Child Element
SCRIPT	Child Element
EDITABLE	Attribute
LABEL_ABBREVIATION	Attribute
DESCRIPTION	Attribute
INITIAL_FOCUS	Attribute
ALT_TEXT	Attribute
CONTROL	Attribute
CONFIG	Attribute

Table A.5 Unsupported FIELD Features

A.7 CONTAINER

For full details on the supported features of this element in static UIM, see Section 5.9.8, *CONTAINER*.

Name	Feature Type
IMAGE	Child Element
LABEL_ABBREVIATION	Attribute

Table A.6 Unsupported CONTAINER Features

A.8 ACTION_SET

For full details on the supported features of this element in static UIM, see Section 5.9.4, *ACTION_SET*.

Name	Feature Type
CONDITION	Child Element
SEPARATOR	Child Element
TOP	Attribute
BOTTOM	Attribute

Table A.7 Unsupported ACTION_SET Features

A.9 WIDGET

For full details on the supported features of this element in static UIM, see Section 5.10.2, *WIDGET*.

Name	Feature Type	Supported/Unsupported attribute values
WIDTH	Attribute	
WIDTH_UNITS	Attribute	
ALIGNMENT	Attribute	
HAS_CONFIRM_PAGE	Attribute	
CONFIG	Attribute	
COMPONENT_STYLE	Attribute	
TYPE	Attribute	Only the value SINGLESELECT is supported, all other values are unsupported

Table A.8 Unsupported WIDGET Features

A.10 ACTION_CONTROL

For full details on the supported features of this element in static UIM, see Section 5.9.3, *ACTION_CONTROL*.

Name	Feature Type	Supported/Unsupported attribute values
CONNECT	Child Element	
SCRIPT	Child Element	

Name	Feature Type	Supported/Unsupported attribute values
CONDITION	Child Element	
LABEL_ABBREVIATION	Attribute	
IMAGE	Attribute	
CONFIRM	Attribute	
DEFAULT	Attribute	
ACTION_ID	Attribute	
ALIGNMENT	Attribute	
TYPE	Attribute	Only the values ACTION and SUBMIT ¹ are supported, all other values are unsupported

Table A.9 Unsupported ACTION_CONTROL Features

A.11 LINK

For full details on the supported features of this element in static UIM, see Section 5.9.22, *LINK*.

Name	Feature Type
CONDITION	Child Element
PAGE_ID_REF	Attribute
SAVE_LINK	Attribute
URL	Attribute
URI_REF	Attribute
URI_SOURCE_NAME	Attribute
URI_SOURCE_PROPERTY	Attribute
SET_HIERARCHY_RETURN_PAGE	Attribute
USE_HIERARCHY_RETURN_PAGE	Attribute
HOME_PAGE	Attribute

Table A.10 Unsupported LINK Features

A.12 INLINE_PAGE

For full details on the supported features of this element in static UIM, see Section 5.9.17, *INLINE_PAGE*.

Name	Feature Type
URI_SOURCE_NAME	Attribute
URI_SOURCE_PROPERTY	Attribute

Table A.11 Unsupported INLINE_PAGE Features

A.13 MENU

For full details on the supported features of this element in static UIM, see Section 5.9.24, *MENU*.

Name	Feature Type	Supported/Unsupported attribute values
CONNECT	Child Element	
MODE	Attribute	Only the value <code>IN_PAGE_NAVIGATION</code> is supported, all other values are unsupported.

Table A.12 Unsupported MENU Features

A.14 SERVER_INTERFACE

For full details on the supported features of this element in static UIM, see Section 5.9.29, *SERVER_INTERFACE*.

Name	Feature Type
ACTION_ID_PROPERTY	Attribute

Table A.13 Unsupported SERVER_INTERFACE Features

A.15 INFORMATIONAL

Only Informationals whose connections endpoints are associated with a server interface defined in the `DISPLAY` phase, are supported. See Section 5.9.16, *INFORMATIONAL* for more details on informationals.). Informationals with other any type of connection endpoints are not supported.

Notes

¹An action of type `SUBMIT` is not supported within a list action menu or a page level action menu. A list action menu is an `ACTION_SET` element within a `LIST` that has a value of `LIST_ROW_MENU` on its `TYPE` attribute. A page level action menu is an `ACTION_SET` defined at the `PAGE` level. See the Section 5.9.4, *ACTION_SET* for further details. All other submit actions are supported.

Appendix B

Maintaining Dynamic UIM Pages

This appendix provides details on how to load dynamic UIM pages into the application resource store.

The way you store your screens differs depending on whether you are working in a development environment or a running system.



Caution

Currently the development of custom dynamic UIM pages is only supported for the presentation of decision details only. Refer to the the chapter *Calculating and Displaying Decision Details* in the *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules* documentation for more details.

Development of dynamic UIM for any purpose beyond that described in this guide is not supported.

B.1 Working in a Development Environment

In order to load a dynamic UIM page into the resource store, you must add two separate entries to the `AppResource.dmx` file in the custom component, each entry corresponding to a dynamic UIM file and an associated properties file.

The following is an example of how to add the `DUIMSample` dynamic UIM page to the `AppResource.dmx` file, so that it will be loaded into the application resource store at build time.

```
<row>
  <attribute name="resourceid">
    <value>1</value>
  </attribute>
  <attribute name="localeIdentifier">
    <value/>
  </attribute>
  <attribute name="name">
    <value>DUIMSample</value>
  </attribute>
</row>
```

```

</attribute>
<attribute name="contentType">
<value>text/plain</value>
</attribute>
<attribute name="contentDisposition">
<value>inline</value>
</attribute>
<attribute name="content">
<value>./custom/data/initial/clob/DUIMSample.uim</value>
</attribute>
<attribute name="internal">
<value>0</value>
</attribute>
<attribute name="lastWritten">
<value>2011-06-13-19.29.40</value>
</attribute>
<attribute name="versionNo">
<value>1</value>
</attribute>
<attribute name="category">
<value>RS_XML</value>
</attribute>
</row>

```

```

<row>
<attribute name="resourceid">
<value>2</value>
</attribute>
<attribute name="localeIdentifier">
<value/>
</attribute>
<attribute name="name">
<value>DUIMSample.properties</value>
</attribute>
<attribute name="contentType">
<value>text/plain</value>
</attribute>
<attribute name="contentDisposition">
<value>inline</value>
</attribute>
<attribute name="content">
<value>./custom/data/initial/clob/DUIMSample.properties</value>
</attribute>
<attribute name="internal">
<value>0</value>
</attribute>
<attribute name="lastWritten">
<value>2011-06-13-19.29.40</value>
</attribute>
<attribute name="versionNo">
<value>1</value>
</attribute>
<attribute name="category">
<value>RS_PROP</value>
</attribute>
</row>

```



Note

The value of the *contentType* attribute specifies the location on the file system that each entry (dynamic UIM file and associated properties file) can be uploaded from. The value of the *category* attribute in the `AppResource.dmx` categorizes a dynamic UIM page resource so that they can be distinguished from other kinds of resources in the resource store. The dynamic UIM file should be cat-

egorized (as shown in the example) as a *RS_XML* resource. The associated properties file should be categorized as *RS_PROP*. Each dynamic UIM resource that is added to the `AppResource.dmx` should also be given the same value so that they all belong to the same category. See the section below for details of how new dynamic UIM pages are loaded into the resource store at runtime. The value of the *localeIdentifier* attribute should be empty (as in the example) if the user's required locale is English. Otherwise the actual locale should be used as the value for this attribute for both the UIM and properties file.

B.2 Working in a Running System

In order to navigate to the home dynamic UIM administration screen in the application, the user must do the following:

- Log into the “admin” application.
- From the shortcut menu, select the “Dynamic UIM” menu item from the “Dynamic UIM” category. This should open the home dynamic UIM administration screen

A user can maintain dynamic UIM pages in the resource store by performing the following actions:

- Add a dynamic UIM page to the Resource Store
- Edit a dynamic UIM page in the Resource Store
- Delete a dynamic UIM page from the Resource Store
- Validate a dynamic UIM page in the Resource Store

B.2.1 Search for Dynamic UIM Pages by Category

In order to view the current list of dynamic UIM pages in the resource store you must perform a search based on the resource store category. This can be done from the home dynamic UIM administration screen as follows:

- Select a menu item for the drop-down list on “Category Search” field.
- Click on the “Search” button. This will return the list of dynamic UIM pages for the selected category.

B.2.2 Uploading a Dynamic UIM page to the Resource Store

From the home dynamic UIM administration screen, a dynamic UIM page can be added to the resource store by doing the following

- Select the *New...* page level action control. This will open a modal dialog page with four mandatory fields.

- Enter the value of the page *Page ID* field. The value must be the same as the value of the `PAGE_ID` attribute in the UIM file that is being uploaded, otherwise an error message will be displayed.
- Select the locale from the drop-down list on the *locale* field. The default is locale is English.
- Use the “Browse” button on the “UIM File” field to navigate to the dynamic UIM file that is to be uploaded to the resource store. As indicated, this is a mandatory field.
- Use the “Browse” button on the “Properties File” field to navigate to the associated properties file to upload to the resource store. As indicated, this is a mandatory field.

B.2.3 Editing a Dynamic UIM page in the resource store

From the home dynamic UIM administration, a dynamic UIM page can be added to the resource store by doing the following:

- From the list of dynamic UIM pages displayed, navigate to the dynamic UIM page that you would like to edit (by `Page ID`), and select the “Edit...” menu item for the list action menu. This should open a modal dialog page with three fields.
- If you would like to download the current version of the dynamic UIM file and associated properties file (to be edited) from the Resource Store the locale file system, then select the “Download” button and save the zip file - containing both aforementioned files - to the file system. The dynamic UIM file and associated properties file can then be unzipped from the downloaded zip and edited as required.
- Use the “Browse” button on the “UIM File” field to navigate to the dynamic UIM file that is to be uploaded to the resource store. As indicated, this is a mandatory field.
- Use the “Browse” button on the “Properties File” field to navigate to the associated properties file to upload to the resource store. As indicated, this is a mandatory field.

B.2.4 Deleting a Dynamic UIM File from the Resource Store

From the home dynamic UIM administration, a dynamic UIM page can be deleted from the resource store by doing the following:

- From the list of dynamic UIM pages displayed, navigate to the dynamic UIM page that you would like to edit (by `Page ID`), and select the “Delete...” menu item for the list action menu. As a result of this action a modal dialog will be displayed, with a message looking for confirmation that you want to delete the selected dynamic UIM page from the resource store.

- The *Yes* button should be selected to delete the dynamic UIM page from the resource store. A new search for dynamic UIM pages in the resource store should reflect the fact that this dynamic UIM page has been removed from the resource store.

B.2.5 Validating a dynamic UIM file in the resource store

From the home dynamic UIM administration, a dynamic UIM page can be validated in the resource store by doing the following:

- From the list of dynamic UIM pages displayed, navigate to the dynamic UIM page that you would like to edit (by `Page ID`), and select the “Validate...” menu item for the list action menu. As a result of this action a modal dialog will be displayed, with a message stating whether the validation has passed or failed. If the validation fails, then the source of the error page will appear in the dialog and the full details of the error can be found in the server logs.

B.2.6 Publish dynamic UIM files

The changes to the dynamic UIM files will not be made public until they are intentionally published to the resource store. This can be done by selecting the “Publish...” page action control from the home dynamic UIM administration screen. This action will open a modal dialog page asking for confirmation that the changes are to be published to the resource store.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typograph-

ical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept F6, Bldg 1
294 Route 100
Somers NY 10589-3216
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources.

IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products

should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This publication documents intended programming interfaces that allow the customer to write programs to obtain the services of IBM Cúram Social Program Management.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/us/en/copytrade.shtml>.

Adobe, the Adobe logo, Adobe SVG Viewer, Adobe Reader, Adobe Flash, and Adobe Flex are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Apache is a trademark of Apache Software Foundation.

Microsoft, Windows, Internet Explorer, and Word, are trademarks of Microsoft Corporation in the United States, other countries, or both.

Mozilla, is registered trademarks of Mozilla Foundation.

UNIX is a registered trademark of the Open Group in the United States and other countries.

WebLogic Server, Java and all Java-based trademarks and logos are registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.