Cúram 8.1.3

Web Client Reference Manual

Note

Before using this information and the product it supports, read the information in Notices on page 411

Edition

This edition applies to Cúram 8.1, 8.1.1, 8.1.2, and 8.1.3.

Contents

1 Web client reference

Use this information to learn how to develop a standard Cúram web client. The standard web client has an HTML user interface that is generated by a middle-tier web application. It conforms to the Java™ EE architecture and is driven by JavaServer pages and servlet technology. This HTML user interface uses standard browser and Web 2.0 technologies, including JavaScript™ and cascading style sheets.

2 Web client overview

Learn about the concepts and terminology that are related to the Cúram Client Development Environment (CDEJ).

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.

- 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 the content.
- 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.
- User interface metadata (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.
- Graphical layout options available to a developer are restricted to enforce a consistent user interface across the whole application.

2.1 User interface metadata (UIM)

User interface metadata (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.

UIM limits the variety of interface layout options that are available to developers, and defaults user interface characteristics based on the known formats of server interfaces. Consequently, 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 consist of 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.

Page content metadata

Users can display and enter server data in the main content area of an application. Page content metadata is used to create the content area. The basic unit of data is a field. Each field is either an output or input parameter of a server interface.

Some XML elements correspond to the user interface elements such as `PAGE`, `FIELD`, `CLUSTER`, `LIST`, `ACTION_CONTROL`, `ACTION_SET`. The `CONNECT` element is an important construct that allows fields to be associated with parameters to server interfaces. In addition to 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 to help with language localization of user interfaces.

Other XML elements, such as `PAGE_PARAMETER` and `SERVER_INTERFACE`, do not have visual representations in a UIM page but are important to the function of the page. A server interface is a method that is implemented by using the Server Development Environment (SDEJ). 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 run. 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 is mapped to input parameters of process phase server methods and the methods are called. After execution of process phase methods, the flow of control is determined by the **Submit** action. By default, submit returns to the same page, or you can specify a link to a new target page.

The following example shows an extract of UIM used to create the content area. The extract displays how the major elements that make up a screen of content area, such as clusters and lists, are represented in UIM.

```
<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>
```

Related reference

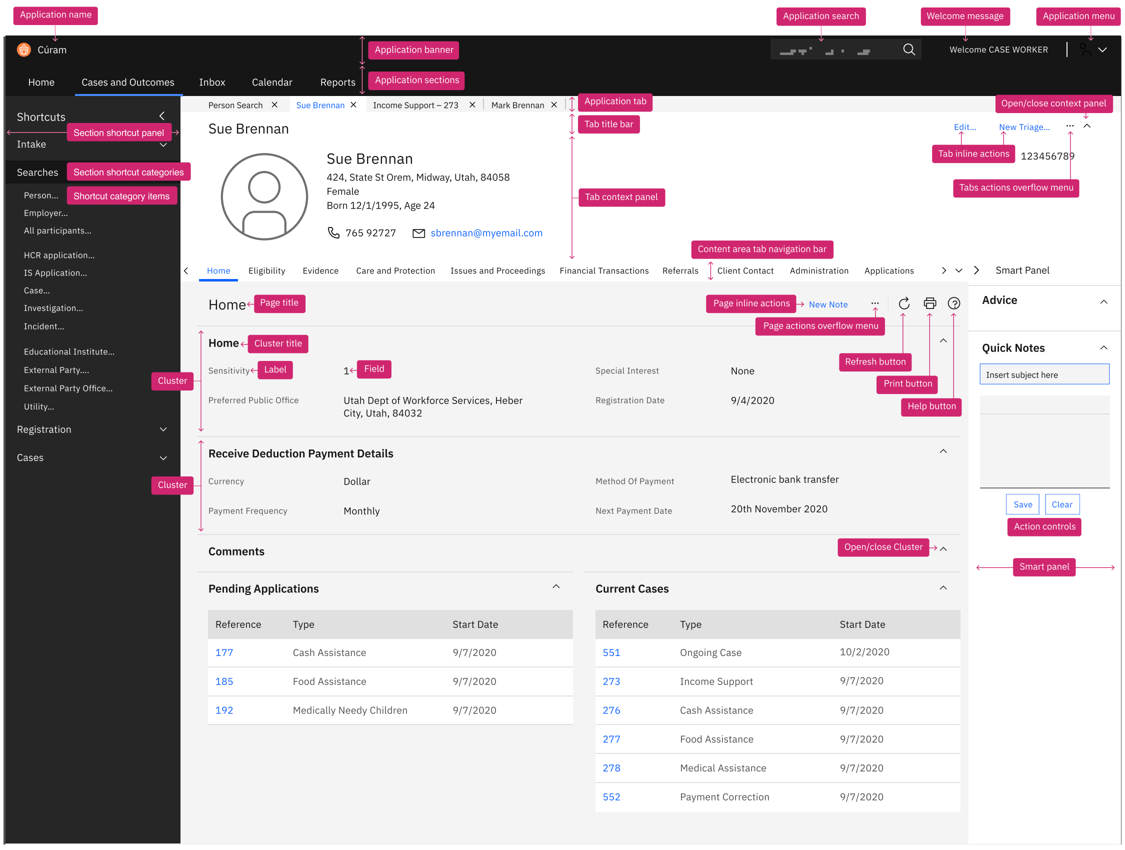
UIM reference on page 323

User interface metadata (UIM) is an XML dialect that is used to specify the contents of the Cúram web application client pages. UIM files must be well-formed XML.

2.2 Application user interface overview

The application user interface contains elements that are implemented in UIM or Carbon components.

In 8.0.0, some UIM components were updated with Carbon components and add-ons on page 30 based on the IBM® Carbon Design System.



Sue Brennan

Sue Brennan
424, State St Orem, Midway, Utah, 84058
Female
Born 12/1/1995, Age 24
765 92727
sbrennan@myemail.com

123456789

Home

Home

Sensitivity	1	Special Interest	None
Preferred Public Office	Utah Dept of Workforce Services, Heber City, Utah, 84032	Registration Date	9/4/2020

Receive Deduction Payment Details

Currency	Dollar	Method Of Payment	Electronic bank transfer
Payment Frequency	Monthly	Next Payment Date	20th November 2020

Comments

Pending Applications

Reference	Type	Start Date
177	Cash Assistance	9/7/2020
185	Food Assistance	9/7/2020
192	Medically Needy Children	9/7/2020

Current Cases

Reference	Type	Start Date
551	Ongoing Case	10/2/2020
273	Income Support	9/7/2020
276	Cash Assistance	9/7/2020
277	Food Assistance	9/7/2020
278	Medical Assistance	9/7/2020
552	Payment Correction	9/7/2020

The following user interface elements are shown in Figure 1.

- **Application banner**

  An application is defined to present a specific view of the data for a user or user role. The application banner runs along the top of the application and shows the overall context of the application.
- **Application name**

  A name is defined for an application on the application banner.
- **Application search**

  The application search element provides a search function on the application banner.
- **Welcome message**

  A welcome message is displayed on the application banner.
- **Application menu**

  An application menu on the application banner allows up to three configurable options for a specific view.

  - A link to log out of the application
  - A link to open the user preferences
  - A link to change the language of the view

  By default, the log out and preferences links are shown. An extra link to information about the application is always shown and cannot be hidden.
- **Application sections**

  An application contains one or more sections that allow quick access to some of the more common user tasks and activities. Application sections are displayed as tabs under the application banner. An application section is a collection of tabs and an optional section shortcut panel to provide navigation in the section.
- **Application tab**

  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 consists of a logical grouping of UIM pages.
- **Tab title bar**

  A title can be defined for the application tab.
- **Tab inline actions and the tab actions overflow menu**

  Actions can be associated with a tab.

  By default, the first few actions menu items are displayed inline, typically the most important. The remaining actions are available from the actions overflow menus.

  Up to 8.0.3, all tab actions are available in a tab actions overflow menu on the tab title bar.

  You can enable the standardized behaviour for inline menu items. For more information, see 3.11 Tab, page, and list actions menus on page 168.

- **Tab context panel**

  A tab contains a context panel at the top of the tab, which contains contextual information that is associated with the data that is displayed in the tab.
- **Context panel open and close chevron**

  By default, the content panel is opened. You can choose to have this context information always available when you work with the data on the tab, or close it to present more content on the page.
- **Section shortcut panel**

  Each section can have a section shortcut panel displayed vertically at the side of the section, which is collapsed by default but can be expanded to show section shortcut categories, which contain shortcut category items.
- **Content area tab navigation bar**

  A tab consists of one or more pages of information, containing standard UIM components or Carbon components. The pages can be navigated by using a navigation bar located under the context panel, which contains navigation tabs that link to single pages or sets of pages. Where a navigation tab links to a set of pages, a page group navigation bar is displayed in the page.
- **Page title**

  A title can be defined for the page.
- **Page inline actions and the page actions overflow menu**

  Actions can be associated with a page.

  By default, the first few actions menu items are displayed inline, typically the most important. The remaining actions are available from the actions overflow menus.

  Up to version 8.0.3, page actions menu items display inline on the page title bar for one or two actions. For three or more actions, all actions are displayed in a page actions overflow menu.

  You can enable the standardized behaviour for inline menu items. For more information, see 3.11 Tab, page, and list actions menus on page 168.
- **Refresh button**

  A default action control that refreshes user interface content is automatically displayed along the top of the page.
- **Print button**

  A default action control that prints user interface content is automatically displayed along the top of the page.
- **Help button**

  A default action control that displays help content in a new window is automatically displayed along the top of the page.
- **Cluster**

  A cluster is a rectangular area in a page 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 can be editable. If editable, the fields appear as a control, such as a text area, drop-down menu, or checkbox.

- **Cluster title**

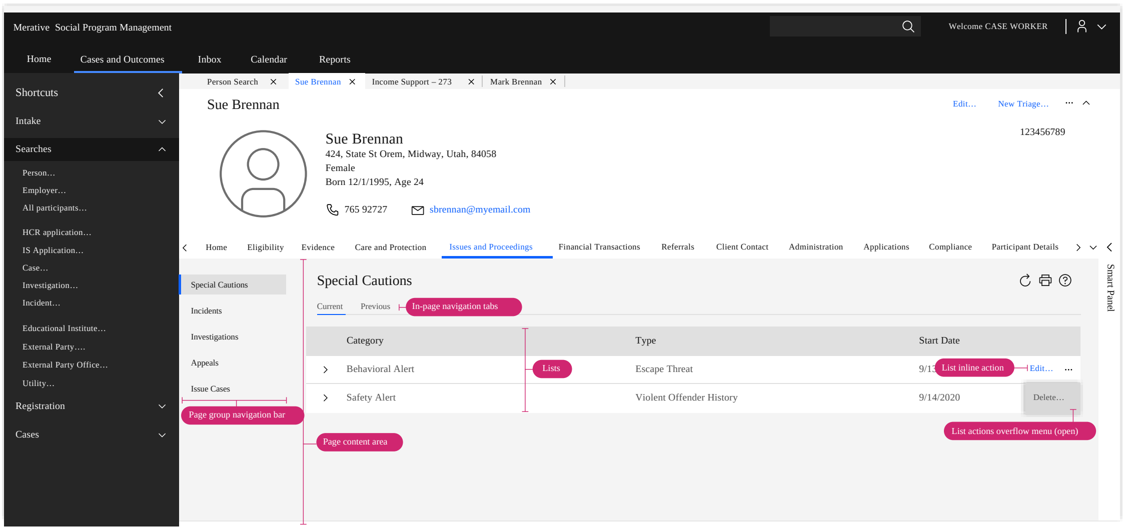
  A cluster title contains text that identifies the cluster in a page.
- **Label**

  A field label is a read-only or 'output' label that identifies the data a field.
- **Field**

  Fields are visually organized into clusters and lists on a page. There can be zero or more of each on a page. Clusters and lists can have a title that describes the type of data displayed.
- **Action controls**

  Action control 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 that are associated with clusters, lists, or the 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 links to another page it can also send parameters to the target page. These parameters are typically used as keys to retrieve server data that populates the target page. By default, action controls display on the widget with which they are associated.
- **Smart panel**

  An optional smart panel displays extra contextual information to the side of the tab, such as quick notes that relate to a case or advice that was given to a client.
- **Cluster open or close chevron**
  You can open or close the cluster as needed.



Category	Type	Start Date
Behavioral Alert	Escape Threat	9/13
Safety Alert	Violent Offender History	9/14/2020

The following user interface elements are shown in Figure 2.

- **In-page navigation tabs**

  A page can contain several tabs of information.
- **Page content area**

  The page content area displays the currently selected UIM page.
- **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.
- **Lists**

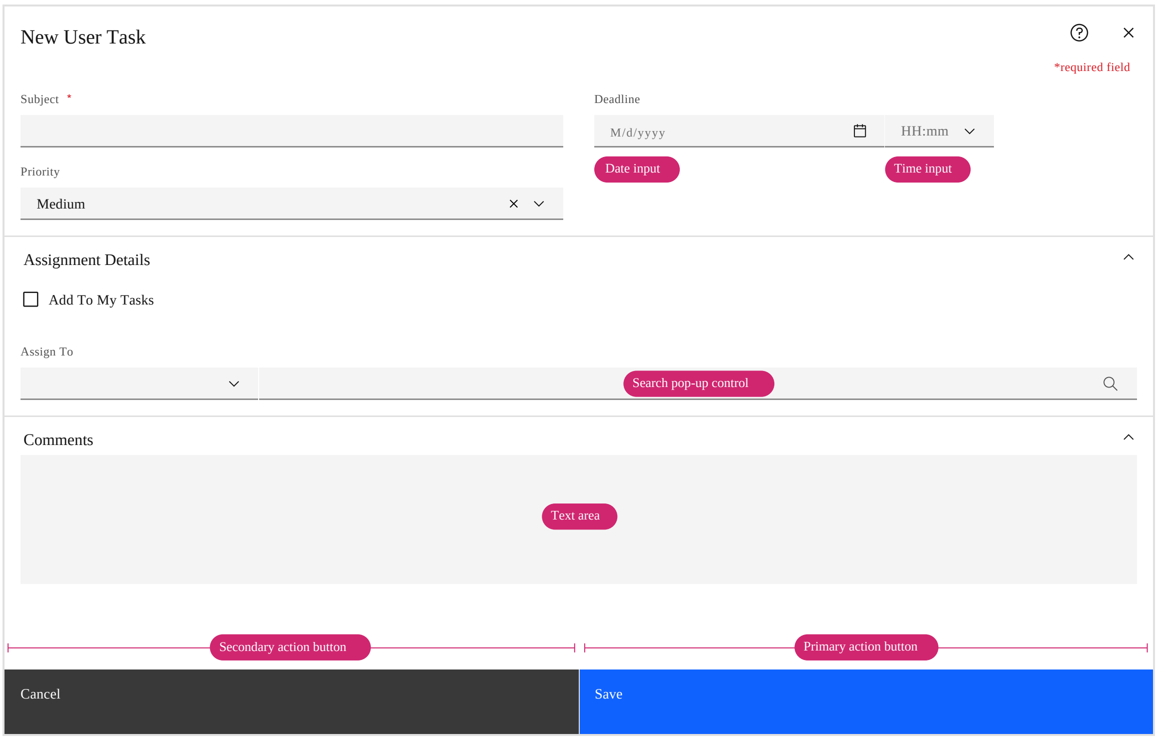
  A list is used to display rows of repeating or indexed fields. As in clusters, fields can include associated labels that are displayed as column headings in the list.
- **List inline actions and the list actions overflow menu**

  Actions can be associated with a list item.

  By default, the first few actions menu items are displayed inline, typically the most important. The remaining actions are available from the actions overflow menus.

  Up to 8.0.3, list actions are placed in a list actions menu on the end of the list row.

  You can enable the standardized behaviour for inline menu items. For more information, see 3.11 Tab, page, and list actions menus on page 168.


New User Task
*required field
Subject *
Deadline
M/d/yyyy
HH:mm
Date input
Time input
Priority
Medium
Assignment Details
Add To My Tasks
Assign To
Search pop-up control
Comments
Text area
Secondary action button
Primary action button
Cancel
Save

The following user interface elements are shown in Figure 3.

- **Date input**

  The Date Time component is a combination of Date input and Time input, which is used in various parts of the application, for example to schedule a task, to create contact logs, for meeting details, and for meeting minutes.

  Date input enables you to type or select a date from the calendar.

- **Time input**

  Time input enables you to enter a time or choose from a list of suggested times.

- **Search pop-up control**

  The search pop-up control opens a context-sensitive search modal dialog.

- **Text area**

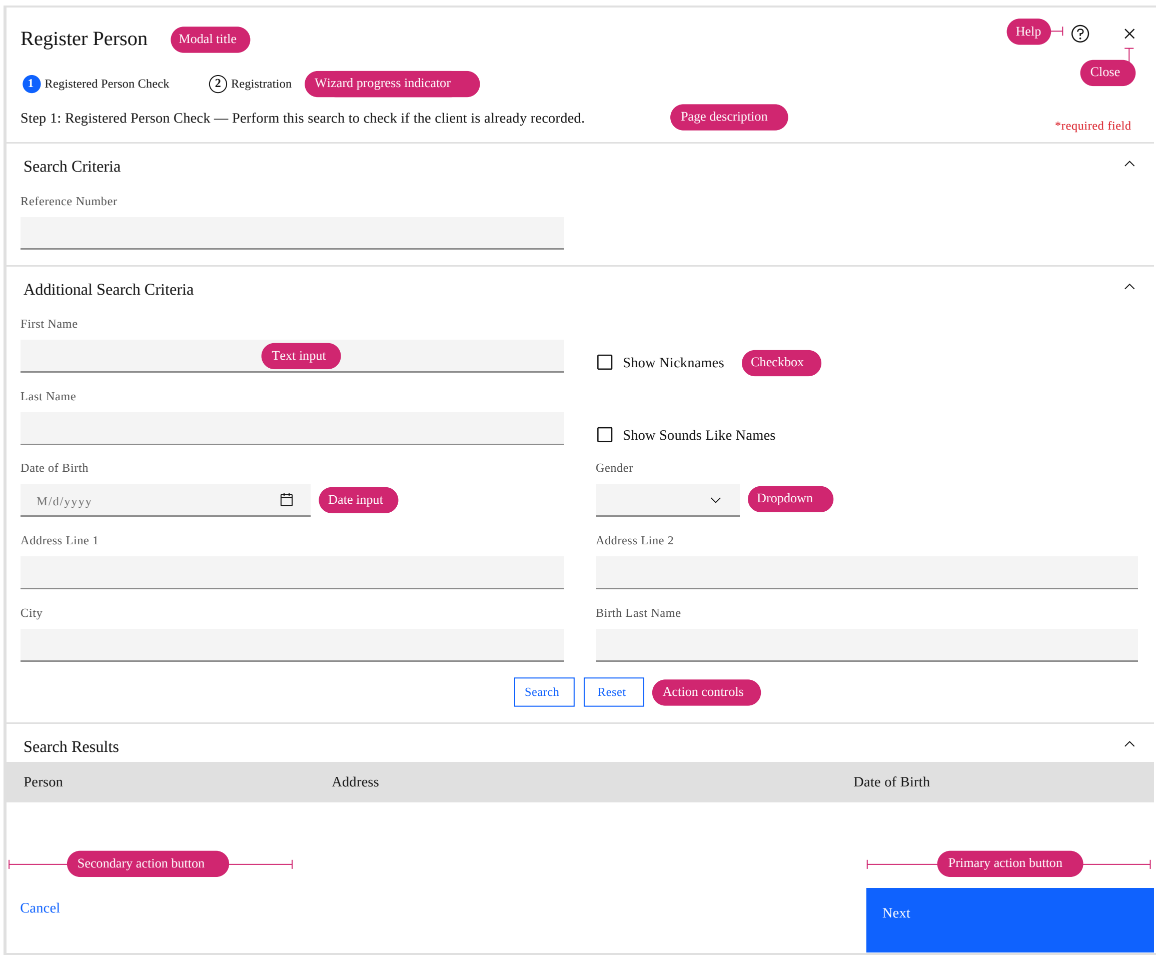
  A text area enables you to input content and data. The component can be used for long form entries.

- **Primary action button**

  A primary modal button is an action control that is used to submit form data, to link to related pages, or to open a modal dialog. By default, the primary action buttons are displayed last.

- **Secondary action button**

  Secondary action buttons can be used only with a primary action button as part of a pair. The secondary modal button's function is to perform the negative action of the set, such as **Cancel** or **Back**.


Register Person
Modal title
Help
Close
1 Registered Person Check
2 Registration
Wizard progress indicator
Step 1: Registered Person Check — Perform this search to check if the client is already recorded.
Page description
*required field
Search Criteria
Reference Number
Additional Search Criteria
First Name
Text input
Show Nicknames
Checkbox
Last Name
Show Sounds Like Names
Date of Birth
M/d/yyyy
Date input
Gender
Dropdown
Address Line 1
Address Line 2
City
Birth Last Name
Search
Reset
Action controls
Search Results
Person
Address
Date of Birth
Secondary action button
Primary action button
Cancel
Next

The following user interface elements are shown in Figure 4.

- **Modal title**

  The modal title contains text that identifies the current modal dialog in a wizard.
- **Help**

  An action control that displays help content in a new window by selecting the **(?)** icon.
- **Close**

  An action control that closes the window by selecting the **(X)** icon.
- **Wizard progress indicator**

  Indicates the sequence of pages in the wizard and highlights the current page in the sequence.
- **Page description**

  A description can be defined for a whole UIM page.
- **Text input**

  Text input enables you to interact with and input content and data. This component can be used for long and short form entries.
- **Checkbox**

  Checkboxes are used when there are multiple items to select in a list. You can select none, one, or any number of items.
- **Date input**

  Date input enables you to type a date or select a date from the calendar.
- **Dropdown**

  Dropdowns present a menu list of options from which you can select an option.

Carbon components and add-ons

The Cúram user interface (UI) uses a number of components that are based on the IBM® Carbon Design System. For more information about the IBM® Carbon Design System, see the related link.

The following UIM components were updated with the Carbon equivalent components:

- Checkbox
- Date and time
- Date input
- Dropdown
- Modal dialog and modal buttons
- Multi-selection checkbox
- Text input
- Text area
- Time input

The following Carbon add-ons were created by using the IBM® Carbon Design System:

- Cluster
- Search pop-up control

> **Note:** While not a Carbon component, the code table hierarchy dropdown was updated to match the Carbon Combo box dropdown style without changing the behavior from previous versions. To open the dropdown, you must click the chevron directly.

The Cúram application uses Carbon styles and assets for font, color, and icons. IBM® Plex is the primary font throughout the application.

Related concepts

Cúram applications on page 31
When a user logs in to Cúram, they are presented with a view that is specific to their role, which is an application. An application is a collection of user interface elements, mostly based on UIM pages, combined to create specific content for a particular user or role.

Related reference

Application configuration on page 101
An application is a collection of user interface elements, based on UIM pages or Carbon components, that are combined to create content for a specific user or role. You create web client applications by configuring application configuration files.

Related information

Carbon Design System

2.3 Cúram applications

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

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

Activating any of these links results 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 by using a number of different files. For example, an application is defined by using an XML file with the extension *`.app`*. Each section that is referenced in the application is defined by using an XML file with the extension *`.sec`*. Any tabs that are referenced by the section are defined by using an XML file with the extension *`.tab`*.

In the following example, an application configuration `.app` file creates an application that contains two sections, and an application banner with a quick search facility.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

Separating the configuration into multiple files allows the reuse of different elements across multiple applications. For example, a common inbox section can be defined and referenced by multiple applications.

Related concepts

Application user interface overview on page 20
The application user interface contains elements that are implemented in UIM or Carbon components.

Related reference

Application configuration on page 101
An application is a collection of user interface elements, based on UIM pages or Carbon components, that are combined to create content for a specific user or role. You create web client applications by configuring application configuration files.

2.4 Page context

UIM pages are displayed in different contexts within an application. The context of the UIM page can result in different behavior for some of the elements.

UIM pages are displayed in the following main contexts:

- Content Area

The content area is where the main content for an application is displayed. When displayed in the content area, a page automatically contains Refresh, Print, and Help buttons in the page title bar.

> **Note:** The Cúram application does not support the web browser **File** > **Print** functionality. The print button prints only the contents of the Content Area.

- Context Panel

  A context panel displays a specific type of UIM page that displays common information for the tab.
- 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. For more information, see `LIST` element on page 368.
- 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. For more information, see Modal dialogs on page 366.
- 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 Tab `smart-panel` element on page 147.

2.5 Page appearance

The application and page metadata provide limited scope to specify the position and layout of user interface elements.

Note the position and layout of the following features:

- 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 the previous user interface elements.

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 must be used.

You can configure page-specific actions by using the UIM ACTION_SET and ACTION_CONTROL elements.

For more information about page actions menus, see 3.11 Tab, page, and list actions menus on page 168.

2.6 The application controller JSP and web client URL

A single JavaServer Pages (JSP) file, `AppController.do`, renders the Cúram client on the browser and the URL always ends with `AppController.do`. The URL does not change as the user navigates between separate pages within the application so the browser back button is not supported.

Direct browsing

You can access a page directly by typing its full URL into the browser's navigation bar, for example, `http://host:port/Curam/en_US/SomePage.do`. On receipt of the request, the browser is automatically redirected to `AppController.do`, which loads the requested page. The process by which the page is loaded depends on whether the page is associated with a tab.

If you access a page directly, the session and its associated tabs is first restored, then a request is sent for the specified page. The page is then loaded in its associated section and tab. However, if this page is not associated with a tab, it is loaded in the currently selected tab. In the case of a new session, this is the **Home** tab.

Tabs changed in this way 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. For more information about tab restoration and session management, see 5.2 Tab Restoration on page 192.

2.7 Web client development environment

Use this information to understand the structure of the web client application project, including related files in the server application, and how to develop, build and deploy the web client application.

The CDEJ transforms files that are specified in user interface metadata (UIM) format into the JavaServer Pages (JSP) to 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*.

You can divide the web client application project into different functional components for ease of development. You can then introduce application changes and updates by dropping in a new component, which automatically overrides the artifacts of another component where appropriate.

If you need more complex pages, you can extend UIM by using the Cúram UI Add-ons Development Environment to develop JavaScript components with the IBM® Carbon Design System, and deploy them into your web client applications.

Outline of the client development process

A summary of the typical steps in the process, which include customizing files and running a number of provided build scripts.

1. Install the Cúram development environment. For more information, see the *Development Environment Installation Guide*.
2. The installation creates both a client application (CDEJ) and a server application (SDEJ) project on your file system that contain all the source files. The client application files include the UIM files for the pages, the application configuration files, the images, and any other resources that the application requires.
3. Create new source files or customize existing files.
4. Build the application and deploy it to an application server. During development, you can deploy the application to an application server embedded in your integrated development environment.
5. When deployed, you can test your application with a web browser. For example,

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

The Cúram application and CDEJ installation folders

The Cúram Application and the CDEJ are installed ready for further development and customization in your project.

The Cúram Application is divided into two main parts:

- The server application that defines the business entities and business logic of the application.
- The web client application that defines how this information is presented to the user.

The actual folder locations vary depending on where they are installed and whether you are developing a Cúram Application, additional applications, or samples.

- ***`<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 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 SDEJ, the tools and infrastructure required to build and run server applications. Typically this is a folder called *`CuramSDEJ`*.

For example, if you install into *`C:/Curam`*, then you have the following folders.

- *`<app-dir>`* is *`C:/Curam`*
- *`<client-dir>`* is *`C:/Curam/webclient`*
- *`<server-dir>`* is *`C:/Curam/EJBServer`*
- *`<cdej-dir>`* is *`C:/Curam/CuramCDEJ`*

CDEJ project folder structure

A Cúram web client application project is organized into a folder structure that is recognized by the CDEJ when the application is built. 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
```

- ***`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 Server interface reference documentation on page 48 for more details.
- ***`buildlogs`***
  Log files generated from each build. See Build Logs on page 47 for more details.
- ***`components`***
  The top-level folder for the application components. Each sub-folder of this folder contains a separate application component. For more information, see Client application component folders on page 38.
- ***`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 to 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. See 9 Custom data conversion and sorting on page 271 for more information about 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 example shows an entry in the *webclient/.classpath* file:

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

  Where <*custom*> represents the name of a custom directory.
- ***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 *Initial_ApplicationConfiguration.properties* file, that is described in Application configuration properties on page 50.
- ***project***
  Configuration files used when customizing the application deployment descriptors. See Customizing the web application descriptor on page 55 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 Deployment on page 50 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 Page previews on page 48 for more information.
- ***WebContent/WEB-INF***
  The standard folder which must exist in every 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 development. The `<server-dir>` project maintains a similar structure to the web client, specifically in relation to the `component` folder.

- **`components/<component-name>/clientapps`**
  Application configuration artifacts. These are the XML configuration files for defining applications, sections, tabs, etc. For more information see 3 Application configuration on page 101.
- **`components/<component-name>/tab`**
  Application configuration artifacts predefined in the Cúram application. XML configuration files shipped with the `core` and other out-of-the-box components 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 then overrides these artifacts.

Client application 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. Never edit files in the core application, which ensures that core application upgrades 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.

Client application component order

You can have any number of application components, but they are processed in a strict *component order*. This order determines the priority that is given to artifacts that share the same name but are in different components. Component order is fundamental to how 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” gives 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 that is not listed in the component order is not included in the build and a warning is displayed to indicate that these components have been ignored. If you do not set the component order at all, the default component order includes all components in alphabetical order.

Note: The SERVER_COMPONENT_ORDER order, used for the `<server-dir>` project, *always* includes all component folders that exist in the `components` folder. If they are omitted from the SERVER_COMPONENT_ORDER environment variable, they are automatically added to the end of the component order in alphabetical order. For more information, see the *Server Developer's 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 prepends 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.

Client application 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 Client application locales on page 41). 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 3 Application configuration on page 101 for details.
- **Image Files**
  Images file referenced from your UIM pages or views can be added to your component's `Images` sub-folder. See Images on page 58 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 might want to customize the appearance of a page to reference your own image file for a logo. This image file is a custom resource.

Client application locales

A locale describes a user's language and country and determines what the user sees in the pages that they access in their web browser. While the data that is displayed largely remains the same when a user's default locale is changed, the labels for the data display in the relevant language.

Note: The following items are also detemined based on the user's locale and display in the relevant language:

- Number formats, for example, the '.' separator as opposed to the ',' separator
- Start day of the week, for example, Monday
- Collation, such as sorting
- Currency symbol placement and spacing

Locales are specified by a simple identifier that contains a two-character, lowercase language code optionally followed by an underscore character and a two-character, uppercase 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. The regional variation can help to identify differences in the dialect or usage of the language, American English in this example, but it can also affect the way dates and numbers are formatted.

The language and country codes are 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 localized your application for that locale. For more information about the supported locales, see the documentation that is provided by the vendor of your JRE. For more information about the procedure for localizing the web client application, 4 Localization on page 177.

Before you build a Cúram application that is localized for a number of locales, you must specify what locales you want to include. 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*. You can provide locales with no regional variation. However, we recommend that you specify a regional variation where possible to ensure that the standard Java libraries can format the currency as expected.

In addition to determining what to build, the LOCALE_LIST determines which languages an internal user can select at run time. Internal users can change their application view language by selecting Language from their application menu. When an administrator configures a user's default locale, all locales that are enabled on the Locale code table are available for selection. However, some of these locales might not be installed. Therefore, we recommend that you keep the locales that are enabled in the Locale code table in sync with the values in the LOCALE_LIST.

Note: Although administrators can change the default locale for all users, we recommend that they do not change it for users that use the following functionality. This functionality is available only in the English language.

- Cúram Income Support application module
- Cúram Child Welfare Structured Decision Making (SDM) add-on module

We also recommend that the default locale is not changed for administrator users because the administration applications contain features that do not currently support localization.

Certain operations, such as the generation of page previews (see Page previews on page 48), are only performed for the default locale.

Improving Build Performance The CDEJ does most of the translation work for the application's locales during the build process. From a single UIM file, it produces one JSP file for each locale in the locale list. If your application supports many locales, you might find it convenient when you develop the application to omit some locale codes from the locale list, as this improves the build performance. You can replace the locales when you want to view or test all of the localized pages.

Related concepts

Language toggle on page 180
You can configure whether internal users can change their default locale to update the language in which field labels, tabs, and shortcut menu items are displayed.

Configuring the currency symbol, placement, and spacing

By setting application properties, system administrators can configure the currency symbol and its placement and spacing. You can base the currency symbol placement and spacing on user locale or on global application properties.

Note: The application properties apply to fields that are modelled using the CURAM_AMOUNT domain definition. In some other cases, development effort is required to define the currency symbol, placement, and spacing.

Table 1: Currency symbol, placement, and spacing configuration properties

Property Name	Default value	Description
`curam.financial.localizedcurrencysymbolplacement`	false	A setting that defines whether the placement and spacing of the currency symbol is determined based on user locale or global properties. When set to true, the placement and spacing are automatically determined based on user locale using standard Java libraries. When set to false, the placement and spacing are determined by the global `curam.financial.currencysymbol` and `curam.financial.currencysymbolplacement` properties.

Property Name	Default value	Description
`curam.financial.currencysymbol`	$	A setting that defines which currency symbol to display and whether to insert a space between the currency symbol and the monetary amout. To insert a space, you must enter a non-breaking space and not a plain space. On Microsoft Windows, press Ctrl+Shift+Spacebar. On an Apple Mac, press Option +Spacebar.
`curam.financial.currencysymbolplacement`	before	A setting that defines whether the currency symbol is displayed before or after the monetary amount.

The three currency symbol placement and spacing properties apply only to fields on UIM pages that are modelled by using the CURAM_AMOUNT domain definition. For the following artifacts, developers must define the currency symbol, placement, and spacing at development time:

- Localized text in properties files
- Server-side constructed messages

Defining the currency symbol, placement, and spacing in properties files

For each language that is installed in the system, you must add the currency symbol, placement, and spacing to the localized text in the relevant properties file.

For example, if English and French are installed, you can add the currency symbol, placement, and spacing rules as follows.

1. For example, add the following entry to the English properties file:

   ```
   IncomeSummary.Text=${0} per month
   ```

2. Add the following corresponding entry to the French properties file:

   ```
   IncomeSummary.Text={0} $ par mois
   ```

Note: The following areas also use properties files to determine the currency symbol, placement, and spacing. You must configure the relevant properties files to display the correct currency symbol, placement, and spacing based on the user's locale:

- Fields in dynamic UIM display rule pages such as description text that does not use the CURAM_AMOUNT domain for monetary amounts
- Dynamic evidence summary pages that display currency symbols
- IEG scripts that display currency symbols in descriptive text

For more information about how to configure the properties files to determine the currency symbol, placement, and spacing for these areas, see the following documentation:

- For more information about configuring the properties files for the UIM display rule pages, see the *How to Build a Product Guide* guide.
- For more information about configuring the properties files for dynamic evidence summary pages, see the *Configuring Evidence Guide* guide.
- For more information about configuring the properties files for IEG scripts, see the *Authoring Scripts using Intelligent Evidence Gathering Guide* guide.

Inserting the currency symbol into custom messages

In some areas of the application, developers might have constructed descriptive text on the server-side in custom Java code and displayed it on client pages. You might need to insert a currency symbol into these messages when rendering monetary amounts. For example, this might be a requirement for the following items:

- Messages and validations that are constructed in Java code
- Text that is constructed in Java code for the Eligibility Viewer
- Text that is constructed in Java code to display a summary of entitlement

For custom code that constructs these types of messages, Cúram provides an API named `curam.core.sl.impl.CurrencySymbol.formatCurrencyAmount()`. When you call this API, it inserts the appropriate currency symbol for a given monetary amount.

The API contains similar logic that is used to format currency symbols for CURAM_AMOUNT domains. If an administrator sets the `curam.financial.localizedcurrencysymbolplacement` property to true, the placement and spacing of the currency symbol is based on the user's locale. Otherwise, the values for the following global properties are used:

- `curam.financial.currencysymbol`
- `curam.financial.currencysymbolplacement`

For more information, see the JavaDoc for the `curam.core.sl.impl.CurrencySymbol.formatCurrencyAmount()` API.

Building an application

Use the following information to help you to build a standard Cúram web client application.

Build targets

You build client applications by using Apache® Ant build scripts. These build scripts define ordered sequences of processing steps called *targets*.

To invoke a target, open a command prompt, change to the `<client-dir>` folder, and 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 is referred to as **`build`** in this information. 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` cleans all existing generated output before building the full web client application.

The following build targets are available for Cúram client projects.

- **client**
  Builds the client application. For more information, see Full and incremental builds on page 46.
- **clean**
  Deletes all of output generated by the other build targets. For more information, see Full and incremental builds on page 46.
- **beandoc**
  Generates reference documentation for the façade server interfaces. For more information, see Server interface reference documentation on page 48.
- **client-with-previews**
  Builds the client application and also generates previews of the pages in HTML format in the `<client-dir>/WebContent/Previews` folder. For more information, see Page previews on page 48.
- **uimgen**
  Generates skeleton UIM pages from the façade server interface definitions. For more information, see UIM Generator Tool on page 49.

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.

Table 2: Environment Variables

Name	Required	Description
CURAMCDEJ	Yes	The location of the installed CDEJ infrastructure, denoted by `<cdej-dir>`. See The Cúram application and CDEJ installation folders on page 35 for details.
CLIENT_DIR	Yes	The location of your web client application, denoted by `<client-dir>`. See The Cúram application and CDEJ installation folders on page 35 for details.

Name	Required	Description
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 to be supported by the application. For more information, see Client application locales on page 41 for details.
CLIENT_COMPONENT_ORDER	No	Defines the prioritized order of the application's components. For more information, see Client application component order on page 39. 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. For more information, see 4.2 File encoding on page 177.
MULTIPLE_VALIDATION_ERRORS	No	Controls the number of errors that are reported during the build process before the build terminates. For more information, see Error reporting on page 47.

Related server build targets

The server application is also built using Apache Ant build scripts and some server targets are needed for the client application. The application configuration files are located in the `<server-dir>` project so 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. For more information, see 3.1 Configuration files on page 101 for more details.
- **database**
  The last step of the **`database`** target is to call the **`inserttabconfiguration`** target. For more information about the database target, see the *Server Developer's Guide*..

Full and incremental builds

The client build target generates a complete web client application. If no previous build output is present, running this target builds the entire application as a *full build*. On subsequent runs of this target, the build scripts compare your source files to the previously generated output files to detect what you have changed. The build then updates the minimum number of output files possible in an *incremental build*.

An incremental build is done automatically when the output of a previous build is present and is much faster than a full build. To run a full build again, you must first run the clean target to remove all of the output from the previous build.

Warning: Building after upgrading

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

Platform Specific Setting When running the client build target from a text-only interface, such as when you use a terminal emulator for access to a Linux® computer, you must add `-Djava.awt.headless=true` to theANT_OPTS environment setting.

Dependency checking

For most changes you need only the incremental build, as changes are detected automatically and only the dependent output files are updated. However, some changes are not detected and you might 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 need to run a full build manually, as the changes will not be detected automatically.

Dependency checking identifies changes to server interfaces that are used by UIM pages. Server interfaces are defined in the application's UML model and more information can be found in Server interface reference documentation on page 48. 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.

Build Logs

Each 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.

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 ran 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 done on files in groups of one hundred, so this option will cause all of the validations 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.

Server interface reference documentation

When you develop UIM pages, you 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. However, 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 the HTML files in the *`<client-dir>/build/bean-doc`* folder. To see 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 that are referenced by any of those domain definitions. Each of these lists provides further links for cross-references or providing more details. Viewing a class displays a list of its operations and selecting an operation shows 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 if 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 helps you to choose a name that is short and readable, but that won't cause any build errors later.

Beside many of the class, operation, and property names, you see a **Copy** button. However, for most browsers, the **Copy** button does not work. You must select the text and use the normal copying commands.

Page previews

You can produce page previews by running the client-with-previews build target. This generates static HTML pages for the default locale in the *`<client-dir>/WebContent/Previews`* folder. Open the pages in a browser to see approximately how the page will look when the application is running. You don't 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 *domain-defaults.xml* file uses a simple XML format. 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. An example *domain-defaults.xml* file with default preview values for domain definitions is shown.

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

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 replaced by an image of the typical output and no default value is required. For more information about complex fields, see 8 Domain-specific controls on page 215.

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 are asked to locate a *ServerAccessBeans.xml* file. This file is generated by theclient target in the *<client-dir>/build* folder.

After the UIM Generator has started, the screen contains the following elements:

- 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, complete the following steps:

1. Select the interface you wish to test from the tree. For example, **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.

External client applications

Because the *webclient* directory contains a mix of components that are targeted for different EAR packaging, it can be difficult to use a single development environment and component order to develop and test them. The external-client build target enables development testing to be done on these external client applications.

The external-client build target enables the creation of an environment and the build of the components specified for an EAR entry in the *deployment_packaging.xml*.

The target requires a -Dapp parameter that identifies the name of an EAR entry within the *deployment_packaging.xml*.

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

The build target copies the components specified for this EAR entry to a *webclient\build \apps\<app name>* directory. In the directory, it builds the project and creates the relevant Eclipse project configuration files to allow the project directory to be imported into Eclipse for development testing.

Deployment

A detailed description of the deployment procedure is provided in the deployment guide for your application server and operating system. However, a number of configuration settings are available in your web client application project prior to deployment.

Application configuration properties

The *ApplicationConfiguration.properties* file defines the most important application configuration settings. You might want to change some of the settings that are relevant to the client application.

The *ApplicationConfiguration.properties* file must be in the *curam/omega3* subfolder of the *<client-dir>/JavaSource* folder. When you create a new application, this folder contains a sample file named *Initial_ApplicationConfiguration.properties*. You must copy this file, rename it to *ApplicationConfiguration.properties*, and change the settings to match your requirements. For the installed Cúram application, this process is done for you already, but you might still want to change other settings.

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

Date and time properties

- **dateformat**

  `dateformat=M d yyyy`

  The format that is used by the date selector widget for entry and display of date fields.

  The value of can be set to one of the following formats:

  - Day-month-year order - `d M yyyy` (the default), `dd MM yyyy`.
  - Month-day-year order - `M d yyyy`, `MM dd yyyy`.
  - Year-month-day order - `yyyy M d`, `yyyy MM dd`.

  In these 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), and `yyyy` represents the four-digit year. An uppercase letter `M` is used for the month, as the lowercase letter `m` is used in Java™ applications to represent the minute value when formatting times. Using `MMM` or `MMMM` to represent the month name is not supported. The formats are specified by using a space character as a separator. The actual separator character that you want to use is specified separately.
- **dateseparator**

  `dateseparator=/`

  The date separator character that is applied to the specified date format. The value can be set to one of the following characters: forward slash (`/`) (the default), period (`.`), comma (`,`), or dash (`-`).
- **timeformat**

  `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**

  `timeseparator=:`

  The value of `timeseparator` can be set by using either a colon (`:`) or period (`.`). Where not specified, the colon (`:`) is used as the default.

Address properties

- **addressFormatType**

  `addressFormatType=US`

  Default address format for addresses in the application.
- **addressDefaultCountryCode**

  `addressDefaultCountryCode=US`

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

Upload properties

- **uploadMaximumSize**

  `uploadMaximumSize=-1`

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

  `uploadThresholdSize=1024`

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

  `uploadRepositoryPath=c:/temp`

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

Synchronizer token properties

- **use.synchronizer.token**

  `use.synchronizer.token=true`

  Whether to use a synchronizer token to prevent accidental resubmission of forms due to use of the browser's **Back** button. The value can be set to `true` (default) or `false`.
- **synchronizer.token.timeout**

  `synchronizer.token.timeout=1800`

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

Tab session properties

- **tabSessionUpdateCountThreshold**

  `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. After the threshold is reached, the recent updates are written and counting starts again from zero until the threshold is reached. A value of `1` 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, see 5 Session management on page 191.
- **tabSessionUpdatePeriodThreshold**

  `tabSessionUpdatePeriodThreshold=120`

Specifies the number of seconds that must elapse since the last time session data was persisted from the web tier to the database before a new update triggers 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, see 5 Session management on page 191.

Progress spinner properties

- **curam.progress.widget.enabled**

  `curam.progress.widget.enabled=true`

  Enables the Progress Spinner widget. The default value is `true`.

  When the value of this property is set to `true`, and the loading of content in any panel or modal dialog takes longer than 2 seconds, a progress spinner will appear to indicate that the system is busy.
- **curam.progress.widget.threshold**

  `curam.progress.widget.threshold=2000`

  Specifies the time offset in milliseconds for the progress spinner to be displayed. The default value is `2000` milliseconds (or 2 seconds).

  This property specifies how long the progress spinner should wait before being displayed. If the page content loads within this period, the progress spinner will not be shown.

Other properties

- **serverConnectionType**

  `serverConnectionType=single`

  Do not change this value.
- **errorpage.stacktrace.output**

  `errorpage.stacktrace.output=false`

  The value of this property is `true` or `false`, where `false` is the default value.

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

  You must set the property value to `false` in a production environment. The HTML error pages that contain the Java™ exception stack trace are not included in the Cúram application malicious code and filtering checks. Therefore, if you set the property to `true` in a production environment, the HTML error pages could potentially make the application more susceptible to injection attacks such as cross-site scripting and link injection.
- **dbtojms.credentials.getter**

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

  Specifies the name of the class that is used to obtain credentials to be used for triggering a `DBtoJMS` transfer. If not specified, a default set of credentials is used for this operation.

For more information about `DBtoJMS` and how to use this property, see the *Batch Processes Developer Guide*.

- **resourceCacheMaximumSize**

  ```
  resourceCacheMaximumSize=16000000
  ```

  Specifies the size of the application resource store cache. By default, the cache is limited to 16 MB (approx.) in size. When that limit is reached, the least recently used resources are 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: A single resource is not cached if it exceeds the size limit for the cache.

- **dynamicUIMInitModelOnStart**

  ```
  dynamicUIMInitModelOnStart=false
  ```

  Indicates whether the Dynamic User Interface Metadata (UIM) system needs to initialize the required information on the application model during startup or when it is first required for a Dynamic UIM page. The default value is `true` and it needs to be set to `false` to cause the model to be initialized when it is first required by a Dynamic UIM page.

  For more information, see 13.10 Dynamic UIM system initialization on page 409.

- **disable.context.panel.print**

  ```
  disable.context.panel.print=false
  ```

  The context panel print is enabled by default along with the main printable content area, which you can configure. You can disable the context panel print by configuring the following property:

  ```
  disable.context.panel.print=true
  ```

  The default value is false.

  If you disable the context panel print, you must perform a client build for the property to take effect.

- **sanitize.link.parameter**

  ```
  sanitize.link.parameter=true
  ```

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

  When the value of this property is set to `true`, any parameters in the request URL within the Cúram application that are built with this value are validated for security vulnerabilities. If tracing is enabled, any parameters in which possible security vulnerabilities are detected are logged and, to maintain security, the request is terminated at a specially created error page.

Related reference

Optimal browser support on page 205
A number of browsers and a range of browser versions are supported for use with Cúram. Users can be notified when they are not using the optimal version of a supported web browser. You can configure the range of supported versions for a browser, the message that users see, and the frequency at which the message is displayed.

Tracing server function calls

The *CDEJResources.properties* file contains a 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 are written to **Standard Out**.

Note: Due to classloader issues with Apache Log4j 2, the web-tier does not currently provide a configurable logging system in the same way as the server-tier.

Customizing the web application descriptor

The web application descriptor that is defined in a *web.xml* file is a standard Java EE web application file. A Cúram web application contains various settings that you might want to change, for example, server connection settings and the session timeout.

The default settings are 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
- **Oracle WebLogic Server**
  <cdej-dir>/ear/WLS/war/WEB-INF/web.xml
- Kubernetes **WebSphere® Application Server Liberty**
  <cdej-dir>/ear/WLP/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. For example, 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, for example `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 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, see 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 with the exception of the `web.xml` used at development time. For more information, see the Cúram deployment documentation. 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 following example.

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

By default the `web.xml` for both WebSphere and WebLogic application servers is configured to enforce HTTPS, a secure SSL connection between the web client and the server. This setting can be modified by changing thetransport-guarantee from `CONFIDENTIAL` to `NONE`.

Note: This change does not disable access to the Cúram web client over HTTPS, but enables additional access over HTTP. For more information, see the *Security Guide*.

Customizing the 404 or Page Not Found error response

The 404 or Not Found error message is an 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 IBM® WebSphere® Application Server, and Oracle WebLogic Server specify a default error page for the Cúram application when an HTTP 404 error is thrown by the application server. You can customize the error message on the default page.

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 `<client-dir>/components/<component_name>/` folder of the application and overriding the `error.message` property specified in that file.

Customization

You customize a Cúram web client application without modifying the original components or their artifacts. This approach makes it easier to upgrade a base application while preserving your custom changes to that application. Use this information to understand how the customization process works, and how you can modify or extend a base application.

Customizations are applied according to the component order. Make your changes in a separate component from the application's original components. The Cúram Application is installed with a number of components, including the core component and a number of other add-on components. Create a new component folder containing a new sub-folder called `components`.

Always add your new component name to the beginning of the component order to give it the highest priority when artifacts are being selected at build-time order. You can add more than one custom component, but you must decide their relative position in the component order. For more information about component order, see Client application component order on page 39.

To begin, your custom component is an empty folder. Customize the application by adding artifacts, such as UIM pages, configuration, and files 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.

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 manner, note artifacts added to the WebContent sub-folder will always override other delivered artifacts, as described in Custom resources on page 76.

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 is selected and the others ignored. 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 is selected and the others ignored. 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 is 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 specified in the following descriptions.

Externalized strings

All string values in UIM files and JavaScript must be externalized, which helps with 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 that must not contain the “=” character, and the value is the localized string value. Blank lines and lines that begin with a “#” character are ignored. The syntax is defined by the `java.util.Properties` class in your Java™ Runtime Environment. For more information, see the class API documentation.

The property value is reproduced in the final application page exactly as you type it in the properties file. The value can contain any character from any language. It is safely processed as you intended in the application, regardless of whether that character is reserved in XML, HTML, or elsewhere.

If you need to enter a character that you cannot generate from the keyboard in a property value, use the Unicode value of that character in a Unicode escape sequence, which is a backslash and a “u” character followed by the four-digit hexadecimal character code. For example, to enter a non-breaking space, the corresponding Unicode escaped sequence is “\u00a0”, see this sample properties file.

```
# 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
```

As you can see, dot (.) characters are a useful way to add some structure to your properties, but they are not required.

When you customize 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 don't need to add the corresponding page or view file to your component and you don't need to redefine any properties that you don't want to change.

Related tasks

Adding or updating help content on page 311
To add new help content, you add a help property to the UIM file for the page and add the help content to the associated properties file. To update existing help, complete the following steps. Adjust the steps if you are updating domain-specific controls.

Related reference

JavaScript externalized strings on page 183
All string values in JavaScript files are externalized to JavaScript property files (*`.js.properties`* files).

UIM externalized strings on page 183
All string values in UIM files are externalized to *`.properties`* files.

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.

SVG images are recommended so graphics can scale without pixelation. Most web browsers support images in SVG, PNG, GIF, and 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
```

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 4.3 Locales on page 179 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.

Image mapping

Images can also be used in the Cúram application to represent different values of displayed fields instead of presenting the value as text. For example, typical Boolean values of true or false might be represented by images of 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 mapping in UIM. If you use a property with a domain listed in the *ImageMapConfig.xml* file, it is automatically displayed as an image. See this sample *ImageMapConfig.xml* file.

```
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>
```

In the example, a field with domain type MY_BOOLEAN has been assigned an image mapping. For accessibility, you must specify an image mapping for each available locale even if the images are identical. The alternative text (alt text) for the image is different for different locales.

`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.

The `CuramLinks.properties` file

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 are merged. If a key is present in more than one *`CuramLinks.properties`* file, the component priority order decides which value is retained.

Runtime configuration XML files

Some XML files in *`<cdej-dir>/lib`* are used by the running client application. To change any of these files, copy the original file into the *`custom`* component subdirectory and modify the copied file.

The client generators use the XML file from the highest priority as specified by the *`CLIENT_COMPONENT_ORDER`* environment variable. The following files are used by the running client application:

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

For more information about customizing these configuration files, see 8 Domain-specific controls on page 215.

Login Pages

The default *`logon.jsp`* login page is in the *`lib/curam/web/jsp`* directory of the Client Development Environment. You can override this default page by placing a copy with your changes in a *`webclient/components/<custom component>/WebContent`* folder. However, there are some guidelines that you must follow.

Include the following JavaScript in the head section of the page to prevent the login page from being loaded in a dialog window.

```
<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>
```

If you want to use the `j_security_check` log-in mechanism, the form submitted from the page must 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 *Server Developer's 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.

JavaScript files

The UIM `SCRIPT` element allows events on the page to trigger JavaScript functions. Provide a relative path to the JavaScript file from your component folder.

For example, you can refer to the *`MyComponent/scripts/myScript.js`* in the `SCRIPT` tag as follows:

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

The paths that you specify are 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, your custom JavaScript might have to be updated in line with changes to HTML structure.

JavaScript APIs for use in the custom JavaScript code are provided within the Cúram application and documented in *`CuramCDEJ\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 can be impacted by changes to the Cúram application in future releases.

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

CSS

Cascading style sheets (`*.css`) define the appearance (colors, fonts, etc.) of the client pages when viewed in a web browser. Default CSS files are provided for the Cúram client application in the `WebContent/WEB-INF/css` folder. Never update the default CSS files. If you want to override particular styles or add new styles, you must create new CSS files in one of your application components.

Note: The underlying HTML and CSS source code used to style the Cúram user interface is not documented and no guarantee is made about its stability across Cúram releases. Therefore, any customization based on that HTML and CSS might be lost as new releases are taken on. The customizations may have to be re-applied by analyzing the HTML and CSS again.

You can view the source code by using browser developer tools.

Any CSS file located in the `component/<some-component>` folder or subfolder is automatically concatenated into the `custom.css` file. The `custom.css` file is included on all pages in the Cúram client application.

An example of customization is to view the CSS that is used to apply a font-size to a field's label. The same CSS selector can then be added to your custom CSS file and a different font-size specified. For example, assuming the HTML and CSS was analyzed and the CSS selector `.field .label` applies the label font-size, the following CSS can override the default. The CSS takes precedence over the Cúram style because custom CSS is included on the page after the default Cúram CSS.

```
.field .label {
font-size: 1rem;
line-height: 1.5;
}
```

Another customization technique is to create a new rule that is an extension of a Cúram rule. To continue the preceding example, a developer analyzes the HTML and sees that an element with the class `.image` is generated as a child of the `.label` element in the Cúram application. It is possible to create a new rule that is specific to the `.image` in this context. The following code outlines how to complete the customization:

```
.field .label {
font-size: 1rem;
line-height: 1.5;
}
.field .label .image {
width: 1.5rem;
height: 1.5rem;
}
```

Note: In the preceding example, if any of these class names change in the HTML then the customization of the `.image` element no longer applies as `.image` depends on being a child of `.label`, which is a child of `.field`.

Note: Some UIM elements support the `STYLE` tag, which allows specific styling to be added to any instance of that element. This styling always overrides the styling in CSS files. For more information, see 13 UIM reference on page 323.

If it is known where the customized image is needed, this maintenance concern can be mitigated by specifying a custom class such as *`.image--large`* by using the STYLE attribute and writing the styles with a simpler selector.

```
.image--large {
width: 1.5rem;
height: 1.5rem;
}
```

This style is more reusable, resulting in less CSS while it avoids dependencies on other classes. *`.image--large`* uses the BEM naming convention to indicate that it is a modifier of the *`.image`* style.

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 you to add application-specific styling to that page.

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

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

Browser-specific CSS

CSS can be specific to the browser used to view the web page. Developers should strive to use the same CSS on all browsers.

Application configuration files

Add the application configuration files for defining application, section and tabs to the *`<server-dir>\components\<component-name>\clientapps`* directory, where *`<component-name>`* is a custom component. Subfolders 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.

Do not modify files in the *`tab`* directory, which contains files that are included with existing components in the default Cúram application.

Note: The default 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. That is, you must not have a *`tab`* folder in *`EJBServer\components\custom`*.

When customizing the application configuration files that are included 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, requires you to add both *SimpleApp.app* and *SimpleApp.properties* to the *clientapps* folder. These files should be based on the merged version of the files. You can use the **inserttabconfiguration** target to get a development copy of the merged file. See the *Server Developer's 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. For example, *SimpleApp.app* must have an *id* of `SimpleApp`.
- The `id` attributes must not contain the period (.) or underscore (_) characters.
- You must add localizable text to a *.properties* file that matches the name of the configuration file. For example, *SimpleApp.app* needs a corresponding *SimpleApp.properties*.
- You can reuse properties files across configuration files. For example, *Person.nav* and *Person.tab* can share the same *Person.properties* file.
- Ensure that you add the proper namespace information when developing the XML files to allow for validation. For example:

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

General configuration

The *curam-config.xml* file contains a number of general-purpose configuration options that affect the appearance or behavior of the web client application. Use this information to understand the main elements of this configuration file.

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 are merged into one *curam-config.xml* file at generation time. The effect of the merge depends on each particular setting.

The following entries are global settings for the application and must only appear once in the final output. If you define one of these entries in a custom component, it overrides the entry of the core component.

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

The following entries are merged.

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

Note, however, that particular address formats can be overridden.

For example, a core component has 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>
```

Your custom component has 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>
```

It is the custom definition that appears in the final merged ***curam-config.xml*** file. This is because both address formats have the same name (`US`).

Dividing the configuration file

You can divide the *curam-config.xml* file into more manageable chunks. You can save one part of the configuration in a file with a different name.

Taking the previous address format configuration as an example, 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>
```

Save this with a file name that ends with *-config.xml* anywhere in your component, for example *address-config.xml*. The file must have the same `APP_CONFIG` root element as the full *curam-config.xml* file. If 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.

STATIC_CONTENT_SERVER

To ensure the successful deployment of Cúram on Kubernetes, you must follow the mandatory procedures that are outlined in this section. The steps also describe in detail how to configure static content for applications, including the Citizen Portal.

Configuring static content for Cúram

Configure static content for Cúram.

About this task

Kubernetes The procedure in this topic is mandatory for deploying Cúram on Kubernetes.

You must optimize an application server to serve dynamic content, and optimize an HTTP server to serve static content. To enable static content in Cúram, complete the following steps:

Procedure

1. Set the *STATIC_CONTENT_SERVER* element in the *`curam-config.XML`* file. The following code example shows how to specify the static content base URL:

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

   The forward slash at the end of the URL in the example is optional. Alternatively, you can specify a relative URL. For example:

   ```
   STATIC_CONTENT_SERVER>
     <URL>/CuramStatic/</URL>
   </STATIC_CONTENT_SERVER>
   ```

2. Do a full Cúram build to include the setting.
3. Use the *zip-static-content* target that is available in the *`webclient`* project to package the static content.

   The target creates a compressed *`StaticContent.zip`* file in the *`webclient\build`* directory. The *`StaticContent.zip`* file contains all the relevant static content to be relocated when the *STATIC_CONTENT_SERVER* setting is enabled. The *-Dstatic.content.zip* setting overwrites the default *`.zip`* file location. All the content in the *`.zip`* file is stored under a root folder called *`WebContent`*. The following sample shows an example of a target.

   ```
   build zip-static-content -Dstatic.content.zip=<myzipfile.zip>
   ```

   The following content is included in the *`.zip`* file:

   - *`WebContent/**/*.htc`*
   - *`WebContent/**/*.html`*
   - *`WebContent/**/*.htm`*
   - *`WebContent/**/*.bmp`*
   - *`WebContent/**/*.cur`*
   - *`WebContent/**/*.gif`*
   - *`WebContent/**/*.ico`*
   - *`WebContent/**/*.jpeg`*
   - *`WebContent/**/*.jpg`*
   - *`WebContent/**/*.mov`*
   - *`WebContent/**/*.png`*
   - *`WebContent/**/*.psd`*
   - *`WebContent/**/*.svg`*
   - *`WebContent/**/*.swc`*
   - *`WebContent/**/*.swf`*
   - *`WebContent/**/*.eot`*
   - *`WebContent/**/*.ttf`*
   - *`WebContent/**/*.woff`*
   - *`WebContent/**/*.woff2`*
   - *`WebContent/**/*.as`*

- *`WebContent/**/*.js`*
- *`WebContent/**/*.vbs`*
- *`WebContent/**/*.css`*
- *`WebContent/**/*.less`*
- *`WebContent/**/*.json`*

4. Set the *`Response Headers`*, as shown in the following example.

The relocation of static content to a separate server enables specific cache control response headers to be set for the content. You can set a cache control response header to provide an instruction to the browser to cache the content for a period of time, so as to reduce network traffic and improve performance. The `Expires` and `Cache-control` headers encourage the browser to cache static content.

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
Cache-control: max-age=86400
```

Note: The `Expires` value must match the formatting in the previous example. The `max-age` attribute value is defined in seconds.

After the headers are set, the browser caches the content until either the `max-age` value or the `Expires` date is reached. When the content is cached, no request is made to the server.

Configuring static content for other Cúram applications

Configure static content for other applications in Cúram.

Procedure

1. Set the *STATIC_CONTENT_SERVER* element in the *`curam-config.XML`* file or `curam-config.deployment`.

2. Use the following steps to do a full Cúram build:

a) Build the server code:

```
cd $SERVER_DIR
  ./build.sh clean server
```

b) Build the client code:

```
cd $CLIENT_DIR
  ./build.sh clean client
```

c) Build the *EAR* files:

```
cd $SERVER_DIR

   ./build.sh websphereEAR
```

Deploy the appropriate application *EAR* files for the application server

3. Build the external application and gather the relevant static content for the *EAR* files, then relocate the static content to a separate HTTP server. The following steps show a Citizen Portal example.

a) Build the external client for CitizenPortal:

```
cd $CLIENT_DIR

   ./build.sh external-client -Dapp=CitizenPortal
```

b) Compress the static content for the Citizen Portal *EAR* file:

```
cd $CLIENT_DIR/build/apps/CitizenPortal

  zip -r CitizenPortalStaticContent.zip WebContent/
```

c) Extract the contents of the `zip` file into a location that is accessible by the HTTP server:

```
unzip CitizenPortalStaticContent.zip -d <Exploded
 CitizenPortalStaticContent.zip location>
```

Enter the previous command on one line.

4. Configure the location to serve static content from the HTTP server:

1. Open the HTTP server configuration file, for example `httpd.conf`, in an editor.

2. Append the following lines to the file to define the locations for the extracted static content, where the specified directories are secure directories that will contain the required static content:

```
Alias /CitizenPortalStaticContent/ "<<Exploded
 CitizenPortalStaticContent.zip location>>/WebContent/"

<Directory "<<Exploded CitizenPortalStaticContent.zip location>>/
WebContent/">
    Options Indexes MultiViews
    AllowOverride None
    Require all granted
</Directory>
```

5. Restart the application server and the HTTP servers.

6. The static content is now served from the HTTP server.

For example, before you enable the Citizen Portal application to serve static content from a HTTP server, the URL for the `curam-extapp` CSS file is as shown in the following sample:

```
https:<host>/CitizenPortal/CDEJ/extapp/themes/curam-extapp/curam-
extapp.css
```

After you enable the static content server, the URL for the CSS file is as shown in the following sample:

```
https:<host>/CitizenPortalStaticContent/CDEJ/extapp/themes/curam-
extapp/curam-extapp.css
```

POPUP_PAGES

See 8.18 Pop-up pages on page 255.

MULTIPLE_POPUP_DOMAINS

See 8.18 Pop-up pages on page 255.

ERROR_PAGE

If an error occurs at run-time, the user is redirected to a page that is defined by the ERROR_PAGE setting. Depending on the error cause, two types of error page can be provided for reporting system or application failures. Alternatively, you can define a default page for reporting both kinds of errors.

This ERROR_PAGE section example is a system error.

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

The ERROR_PAGE section example has one default page.

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

Note: When overriding the `ERROR_PAGE` setting, your custom configuration cannot 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 msut use a `TYPE` attribute and must override both supported types of error page.

MULTIPLE_SELECT

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

A Multiple Select section example:

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

FILE_DOWNLOAD_CONFIG

For more information about file downloads, see `ACTION_CONTROL` element on page 326.

ENABLE_COLLAPSIBLE_CLUSTERS

By default this value is set to `true`. Set to `false` to disable collapsible clusters. A Disable Collapsible Clusters example is shown.

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

APPEND_COLON

Set to `true` to automatically append colons to `FIELD` and `CONTAINER` labels within `CLUSTER` elements. An Append Colon section example is shown.

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

ADDRESS_CONFIG

See 8 Domain-specific controls on page 215.

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. An Admin Section example is shown.

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

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

STATIC_CONTENT_SERVER

Configure static content for Cúram

Kubernetes The procedures in this topic are mandatory for deploying Cúram on Kubernetes.

An application server is optimized to serve dynamic content, while an HTTP server is optimized to serve static content. To enable static content in Cúram, set the *STATIC_CONTENT_SERVER* element in the *`curam-config.XML`* file and perform a full Cúram build as shown in this static content base URL example.

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

The forward slash at the end of the URL in the example is optional. You can also use a relative URL as shown in thisrelative URL example.

```
<STATIC_CONTENT_SERVER>
  <URL>/CuramStatic/</URL>
</STATIC_CONTENT_SERVER>
```

A full build is required to pick up this setting.

Where this option is used, the static content can be packaged by using the *zip-static-content* target available in the *`webclient`* project. This target creates a .zip file, *`StaticContent.zip`*, in the *`webclient\build`* directory. The *`StaticContent.zip`* file contains all relevant static content to be relocated when the *STATIC_CONTENT_SERVER* setting is enabled. The -*Dstatic.content.zip* setting can be used to overwrite the default .zip file location. All content in the .zip file is stored under a root folder called *`WebContent`*. A target example is shown.

```
build zip-static-content -Dstatic.content.zip=<myzipfile.zip>
```

The following content is included in the .zip file:

- *WebContent/**/*.htc*
- *WebContent/**/*.html*
- *WebContent/**/*.htm*
- *WebContent/**/*.bmp*
- *WebContent/**/*.cur*
- *WebContent/**/*.gif*
- *WebContent/**/*.ico*
- *WebContent/**/*.jpeg*
- *WebContent/**/*.jpg*
- *WebContent/**/*.mov*
- *WebContent/**/*.png*
- *WebContent/**/*.psd*
- *WebContent/**/*.svg*
- *WebContent/**/*.swc*
- *WebContent/**/*.swf*
- *WebContent/**/*.eot*
- *WebContent/**/*.ttf*
- *WebContent/**/*.woff*
- *WebContent/**/*.woff2*
- *WebContent/**/*.as*
- *WebContent/**/*.js*
- *WebContent/**/*.vbs*
- *WebContent/**/*.css*
- *WebContent/**/*.less*
- *WebContent/**/*.json*

The relocation of static content to a separate server allows for specific cache control response headers to be set for this content. Setting a cache control response header provides an instruction to the browser to cache this content for a period of time; the aim of which is to reduce network traffic and improve performance. The *Expires* and *Cache-control* headers encourage the browser to cache static content. Example Response Headers are shown.

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
Cache-control: max-age=86400
```

The *Expires* value must match the specific formatting shown to be recognized. The *max-age* attribute value is in seconds.

When the headers are set, the browser caches the content until the *max-age* value is reached or the *Expires* date is reached. When cached, no request is made to the server.

Related information

Performance tuning

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. A Field Error Indicators example is shown.

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

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 rights when the page is initially loaded. If a user fails authorization for *any* of the server functions, an authorization error message is displayed and the user is prevented from viewing the page. For example, if a user has authorization rights to access the DISPLAY phase server function, but not the ACTION phase, they cannot view the page.

The SECURITY_CHECK_ON_PAGE_LOAD setting in *`curam-config.xml`*, which is true by default, indicates that authorization checks should be performed before the page is loaded to ensure the user has access rights to *all* server functions referenced by SERVER_INTERFACE elements on the UIM page.

Setting the SECURITY_CHECK_ON_PAGE_LOAD attribute to false disables this initial authorization check and defer authorization to the point at which the server function is invoked. As a result, on an edit page for example, a user would require authorization rights for the DISPLAY phase server function at a minimum. If they did not have authorization rights for the ACTION phase server function, the page displays, but the user receives an authorization error message when the page is submitted.

To set SECURITY_CHECK_ON_PAGE_LOAD, and disable authorization on page load, add the following setting to the *`curam-config.xml`* file:

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

If the `SECURITY_CHECK_ON_PAGE_LOAD` element is not specified, it defaults to `TRUE`.

There is no security risk associated with this change, but the change has implications for auditing. When the authorization check is performed on page load, by default authorization failures are not added to the AuthorisationLog database table. This behavior can be modified by setting `curam.enable.logging.client.authcheck` to true using the Property Administration screens.

When the authorization check is deferred to the invocation of the server function, i.e. SECURITY_CHECK_ON_PAGE_LOAD is false, authorization failures are always logged. It is not possible to control or disable this behavior. As a result, the risk is that the AuthorisationLog database table is filled with noise in the form of authorization failures that are valid failures based on usage.

ENABLE_SELECT_ALL_CHECKBOX

The multi-select check-box `WIDGET` 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 clear all of the check-boxes at once. An Enable Select All Check-box example is shown.

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

If the `ENABLE_SELECT_ALL_CHECKBOX` element is not specified, it defaults to `FALSE`.

For more information, see the The MULTISELECT widget on page 398.

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>
```

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>
```

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>
```

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>
```

If the `SCROLLBAR_CONFIG` element is not specified, no `LIST` or `CLUSTER` element displays a vertical scrollbar.

PAGINATION

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

An example Pagination configuration is shown.

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

Table 3: Pagination configuration options

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

Custom resources

You can include custom files in the web application.

Warning: Before you use custom resources, ensure that you understand the affects. It is advised to first view the generated `WebContent` folder (located `webclient/WebContent`) and to be aware of what files exist in it. Placing a similar file in the `WebContent` folder of a component overwrites the currently existing file in the generated `WebContent` folder.

Files included in the application in this way take precedence over the merging and overriding process for other resources. For example, if you include a CSS file in this way, the contents of the file is not be included in the CSS overriding process described in CSS on page 62.

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 is 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.

It is also important to note that the files placed in a `WebContent` folder within a component are completely ignored during the build process and are not processed. They are merely copied across. For example, if you have a JavaScript properties file in the `WebContent` folder of your component, it is not processed.

Finally, when multiple components have a `WebContent` folder, they are copied based on component priority, but the copy is time-stamp based. The copy command always uses verbose output for these files so you can see exactly what files are being copied.

Complete the following steps to include files:

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 with any folder structure.
3. When you run the **`client`** build target, these files are copied directly to the `<client-dir>/WebContent`, which represents the root of the web application. The folder structure is maintained during the copy.

2.8 Deployment

A detailed description of the deployment procedure is provided in the deployment guide for your application server and operating system. However, a number of configuration settings are available in your web client application project prior to deployment.

Application configuration properties

The *ApplicationConfiguration.properties* file defines the most important application configuration settings. You might want to change some of the settings that are relevant to the client application.

The *ApplicationConfiguration.properties* file must be in the *curam/omega3* subfolder of the *<client-dir>/JavaSource* folder. When you create a new application, this folder contains a sample file named *Initial_ApplicationConfiguration.properties*. You must copy this file, rename it to *ApplicationConfiguration.properties*, and change the settings to match your requirements. For the installed Cúram application, this process is done for you already, but you might still want to change other settings.

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

Date and time properties

- **dateformat**

  ```
  dateformat=M d yyyy
  ```

  The format that is used by the date selector widget for entry and display of date fields.

  The value of can be set to one of the following formats:

  - Day-month-year order - `d M yyyy` (the default), `dd MM yyyy`.
  - Month-day-year order - `M d yyyy`, `MM dd yyyy`.
  - Year-month-day order - `yyyy M d`, `yyyy MM dd`.

  In these 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), and `yyyy` represents the four-digit year. An uppercase letter `M` is used for the month, as the lowercase letter `m` is used in Java™ applications to represent the minute value when formatting times. Using `MMM` or `MMMM` to represent the month name is not supported. The formats are specified by using a space character as a separator. The actual separator character that you want to use is specified separately.
- **dateseparator**

  ```
  dateseparator=/
  ```

  The date separator character that is applied to the specified date format. The value can be set to one of the following characters: forward slash (`/`) (the default), period (`.`), comma (`,`), or dash (`-`).
- **timeformat**

  ```
  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**

  ```
  timeseparator=:
  ```

The value of `timeseparator` can be set by using either a colon (`:`) or period (`.`). Where not specified, the colon (`:`) is used as the default.

Address properties

- **addressFormatType**

  `addressFormatType=US`

  Default address format for addresses in the application.
- **addressDefaultCountryCode**

  `addressDefaultCountryCode=US`

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

Upload properties

- **uploadMaximumSize**

  `uploadMaximumSize=-1`

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

  `uploadThresholdSize=1024`

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

  `uploadRepositoryPath=c:/temp`

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

Synchronizer token properties

- **use.synchronizer.token**

  `use.synchronizer.token=true`

  Whether to use a synchronizer token to prevent accidental resubmission of forms due to use of the browser's **Back** button. The value can be set to `true` (default) or `false`.
- **synchronizer.token.timeout**

  `synchronizer.token.timeout=1800`

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

Tab session properties

- **tabSessionUpdateCountThreshold**

  `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. After the threshold is reached, the recent updates are written and counting starts again from zero until the threshold is reached. A value of `1` 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, see 5 Session management on page 191.
- **tabSessionUpdatePeriodThreshold**

  `tabSessionUpdatePeriodThreshold=120`

  Specifies the number of seconds that must elapse since the last time session data was persisted from the web tier to the database before a new update triggers 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, see 5 Session management on page 191.

Progress spinner properties

- **curam.progress.widget.enabled**

  `curam.progress.widget.enabled=true`

  Enables the Progress Spinner widget. The default value is `true`.

  When the value of this property is set to `true`, and the loading of content in any panel or modal dialog takes longer than 2 seconds, a progress spinner will appear to indicate that the system is busy.
- **curam.progress.widget.threshold**

  `curam.progress.widget.threshold=2000`

  Specifies the time offset in milliseconds for the progress spinner to be displayed. The default value is `2000` milliseconds (or 2 seconds).

  This property specifies how long the progress spinner should wait before being displayed. If the page content loads within this period, the progress spinner will not be shown.

Other properties

- **serverConnectionType**

  `serverConnectionType=single`

  Do not change this value.
- **errorpage.stacktrace.output**

  `errorpage.stacktrace.output=false`

  The value of this property is `true` or `false`, where `false` is the default value.

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

You must set the property value to `false` in a production environment. The HTML error pages that contain the Java™ exception stack trace are not included in the Cúram application malicious code and filtering checks. Therefore, if you set the property to `true` in a production environment, the HTML error pages could potentially make the application more susceptible to injection attacks such as cross-site scripting and link injection.

- **dbtojms.credentials.getter**

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

  Specifies the name of the class that is used to obtain credentials to be used for triggering a `DBtoJMS` transfer. If not specified, a default set of credentials is used for this operation. For more information about `DBtoJMS` and how to use this property, see the *Batch Processes Developer Guide*.
- **resourceCacheMaximumSize**

  `resourceCacheMaximumSize=16000000`

  Specifies the size of the application resource store cache. By default, the cache is limited to 16 MB (approx.) in size. When that limit is reached, the least recently used resources are 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: A single resource is not cached if it exceeds the size limit for the cache.
- **dynamicUIMInitModelOnStart**

  `dynamicUIMInitModelOnStart=false`

  Indicates whether the Dynamic User Interface Metadata (UIM) system needs to initialize the required information on the application model during startup or when it is first required for a Dynamic UIM page. The default value is `true` and it needs to be set to `false` to cause the model to be initialized when it is first required by a Dynamic UIM page.

  For more information, see 13.10 Dynamic UIM system initialization on page 409.
- **disable.context.panel.print**

  `disable.context.panel.print=false`

  The context panel print is enabled by default along with the main printable content area, which you can configure. You can disable the context panel print by configuring the following property:

  `disable.context.panel.print=true`

  The default value is false.

  If you disable the context panel print, you must perform a client build for the property to take effect.
- **sanitize.link.parameter**

  `sanitize.link.parameter=true`

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

When the value of this property is set to `true`, any parameters in the request URL within the Cúram application that are built with this value are validated for security vulnerabilities. If tracing is enabled, any parameters in which possible security vulnerabilities are detected are logged and, to maintain security, the request is terminated at a specially created error page.

Related reference

Optimal browser support on page 205

A number of browsers and a range of browser versions are supported for use with Cúram. Users can be notified when they are not using the optimal version of a supported web browser. You can configure the range of supported versions for a browser, the message that users see, and the frequency at which the message is displayed.

Tracing server function calls

The *`CDEJResources.properties`* file contains a 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 are written to **Standard Out**.

Note: Due to classloader issues with Apache Log4j 2, the web-tier does not currently provide a configurable logging system in the same way as the server-tier.

Customizing the web application descriptor

The web application descriptor that is defined in a *`web.xml`* file is a standard Java EE web application file. A Cúram web application contains various settings that you might want to change, for example, server connection settings and the session timeout.

The default settings are 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`
- **Oracle WebLogic Server**
  `<cdej-dir>/ear/WLS/war/WEB-INF/web.xml`
- Kubernetes **WebSphere® Application Server Liberty**
  `<cdej-dir>/ear/WLP/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. For example, 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, for example 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 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, see 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 with the exception of the *web.xml* used at development time. For more information, see the Cúram deployment documentation. 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 following example.

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

By default the *web.xml* for both WebSphere and WebLogic application servers is configured to enforce HTTPS, a secure SSL connection between the web client and the server. This setting can be modified by changing thetransport-guarantee from CONFIDENTIAL to NONE.

Note: This change does not disable access to the Cúram web client over HTTPS, but enables additional access over HTTP. For more information, see the *Security Guide*.

Customizing the 404 or Page Not Found error response

The 404 or Not Found error message is an 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 IBM® WebSphere® Application Server, and Oracle WebLogic Server specify a default error page for the Cúram application when an HTTP 404 error is thrown by the application server. You can customize the error message on the default page.

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 `<client-dir>/components/<component_name>/` folder of the application and overriding the `error.message` property specified in that file.

2.9 Customization

You customize a Cúram web client application without modifying the original components or their artifacts. This approach makes it easier to upgrade a base application while preserving your custom changes to that application. Use this information to understand how the customization process works, and how you can modify or extend a base application.

Customizations are applied according to the component order. Make your changes in a separate component from the application's original components. The Cúram Application is installed with a number of components, including the core component and a number of other add-on components. Create a new component folder containing a new sub-folder called `components`. Always add your new component name to the beginning of the component order to give it the highest priority when artifacts are being selected at build-time order. You can add more than one custom component, but you must decide their relative position in the component order. For more information about component order, see Client application component order on page 39.

To begin, your custom component is an empty folder. Customize the application by adding artifacts, such as UIM pages, configuration, and files 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.

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 manner, note artifacts added to the WebContent sub-folder will always override other delivered artifacts, as described in Custom resources on page 76.

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 is selected and the others ignored. 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 is selected and the others ignored. 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 is 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 specified in the following descriptions.

Externalized strings

All string values in UIM files and JavaScript must be externalized, which helps with 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 that must not contain the "=" character, and the value is the localized string value. Blank lines and lines that begin with a "#" character are ignored. The syntax is defined by the `java.util.Properties` class in your Java™ Runtime Environment. For more information, see the class API documentation.

The property value is reproduced in the final application page exactly as you type it in the properties file. The value can contain any character from any language. It is safely processed as you intended in the application, regardless of whether that character is reserved in XML, HTML, or elsewhere.

If you need to enter a character that you cannot generate from the keyboard in a property value, use the Unicode value of that character in a Unicode escape sequence, which is a backslash and a "u" character followed by the four-digit hexadecimal character code. For example, to enter a non-breaking space, the corresponding Unicode escaped sequence is "\u00a0", see this sample properties file.

```
# 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
```

As you can see, dot (.) characters are a useful way to add some structure to your properties, but they are not required.

When you customize 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 don't need to add the corresponding page or view file to your component and you don't need to redefine any properties that you don't want to change.

Related tasks

Adding or updating help content on page 311

To add new help content, you add a help property to the UIM file for the page and add the help content to the associated properties file. To update existing help, complete the following steps. Adjust the steps if you are updating domain-specific controls.

Related reference

JavaScript externalized strings on page 183
All string values in JavaScript files are externalized to JavaScript property files (*.js.properties* files).

UIM externalized strings on page 183
All string values in UIM files are externalized to *.properties* files.

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.

SVG images are recommended so graphics can scale without pixelation. Most web browsers support images in SVG, PNG, GIF, and 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
```

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 4.3 Locales on page 179 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.

Image mapping

Images can also be used in the Cúram application to represent different values of displayed fields instead of presenting the value as text. For example, typical Boolean values of `true` or `false` might be represented by images of 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 mapping in UIM. If you use a property with a domain listed in the *`ImageMapConfig.xml`* file, it is automatically displayed as an image. See this sample *`ImageMapConfig.xml`* file.

```
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>
```

In the example, a field with domain type MY_BOOLEAN has been assigned an image mapping. For accessibility, you must specify an image mapping for each available locale even if the images are identical. The alternative text (alt text) for the image is different for different locales.

`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.

The *`CuramLinks.properties`* file

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 are merged. If a key is present in more than one *`CuramLinks.properties`* file, the component priority order decides which value is retained.

Runtime configuration XML files

Some XML files in *<cdej-dir>/lib* are used by the running client application. To change any of these files, copy the original file into the *custom* component subdirectory and modify the copied file.

The client generators use the XML file from the highest priority as specified by the *CLIENT_COMPONENT_ORDER* environment variable. The following files are used by the running client application:

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

For more information about customizing these configuration files, see 8 Domain-specific controls on page 215.

Login Pages

The default *logon.jsp* login page is in the *lib/curam/web/jsp* directory of the Client Development Environment. You can override this default page by placing a copy with your changes in a *webclient/components/<custom component>/WebContent* folder. However, there are some guidelines that you must follow.

Include the following JavaScript in the head section of the page to prevent the login page from being loaded in a dialog window.

```
<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>
```

If you want to use the `j_security_check` log-in mechanism, the form submitted from the page must 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 *Server Developer's 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.

JavaScript files

The UIM `SCRIPT` element allows events on the page to trigger JavaScript functions. Provide a relative path to the JavaScript file from your component folder.

For example, you can refer to the *`MyComponent/scripts/myScript.js`* in the `SCRIPT` tag as follows:

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

The paths that you specify are 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, your custom JavaScript might have to be updated in line with changes to HTML structure.

JavaScript APIs for use in the custom JavaScript code are provided within the Cúram application and documented in *`CuramCDEJ\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 can be impacted by changes to the Cúram application in future releases.

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

CSS

Cascading style sheets (*`*.css`*) define the appearance (colors, fonts, etc.) of the client pages when viewed in a web browser. Default CSS files are provided for the Cúram client application in the *`WebContent/WEB-INF/css`* folder. Never update the default CSS files. If you want to override particular styles or add new styles, you must create new CSS files in one of your application components.

Note: The underlying HTML and CSS source code used to style the Cúram user interface is not documented and no guarantee is made about its stability across Cúram releases. Therefore, any customization based on that HTML and CSS might be lost as new releases are taken on. The customizations may have to be re-applied by analyzing the HTML and CSS again.

You can view the source code by using browser developer tools.

Any CSS file located in the *`component/<some-component>`* folder or subfolder is automatically concatenated into the *`custom.css`* file. The *`custom.css`* file is included on all pages in the Cúram client application.

An example of customization is to view the CSS that is used to apply a font-size to a field's label. The same CSS selector can then be added to your custom CSS file and a different font-size specified. For example, assuming the HTML and CSS was analyzed and the CSS selector *`.field .label`* applies the label font-size, the following CSS can override the default. The

CSS takes precedence over the Cúram style because custom CSS is included on the page after the default Cúram CSS.

```
.field .label {
font-size: 1rem;
line-height: 1.5;
}
```

Another customization technique is to create a new rule that is an extension of a Cúram rule. To continue the preceding example, a developer analyzes the HTML and sees that an element with the class `.image` is generated as a child of the `.label` element in the Cúram application. It is possible to create a new rule that is specific to the `.image` in this context. The following code outlines how to complete the customization:

```
.field .label {
font-size: 1rem;
line-height: 1.5;
}
.field .label .image {
width: 1.5rem;
height: 1.5rem;
}
```

Note: In the preceding example, if any of these class names change in the HTML then the customization of the `.image` element no longer applies as `.image` depends on being a child of `.label`, which is a child of `.field`.

Note: Some UIM elements support the `STYLE` tag, which allows specific styling to be added to any instance of that element. This styling always overrides the styling in CSS files. For more information, see 13 UIM reference on page 323.

If it is known where the customized image is needed, this maintenance concern can be mitigated by specifying a custom class such as `.image--large` by using the STYLE attribute and writing the styles with a simpler selector.

```
.image--large {
width: 1.5rem;
height: 1.5rem;
}
```

This style is more reusable, resulting in less CSS while it avoids dependencies on other classes. `.image--large` uses the BEM naming convention to indicate that it is a modifier of the `.image` style.

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 you to add application-specific styling to that page.

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

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

Browser-specific CSS

CSS can be specific to the browser used to view the web page. Developers should strive to use the same CSS on all browsers.

Application configuration files

Add the application configuration files for defining application, section and tabs to the *`<server-dir>\components\<component-name>\clientapps`* directory, where *`<component-name>`* is a custom component. Subfolders 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.

Do not modify files in the *`tab`* directory, which contains files that are included with existing components in the default Cúram application.

Note: The default 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. That is, you must not have a *`tab`* folder in *`EJBServer\components\custom`*.

When customizing the application configuration files that are included 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, requires you to add both *`SimpleApp.app`* and *`SimpleApp.properties`* to the *`clientapps`* folder. These files should be based on the merged version of the files. You can use the **`inserttabconfiguration`** target to get a development copy of the merged file. See the *Server Developer's 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. For example, *`SimpleApp.app`* must have an *`id`* of `SimpleApp`.
- The `id` attributes must not contain the period (.) or underscore (_) characters.
- You must add localizable text to a *`.properties`* file that matches the name of the configuration file. For example, *`SimpleApp.app`* needs a corresponding *`SimpleApp.properties`*.
- You can reuse properties files across configuration files. For example, *`Person.nav`* and *`Person.tab`* can share the same *`Person.properties`* file.

- Ensure that you add the proper namespace information when developing the XML files to allow for validation. For example:

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

General configuration

The *curam-config.xml* file contains a number of general-purpose configuration options that affect the appearance or behavior of the web client application. Use this information to understand the main elements of this configuration file.

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 are merged into one *curam-config.xml* file at generation time. The effect of the merge depends on each particular setting.

The following entries are global settings for the application and must only appear once in the final output. If you define one of these entries in a custom component, it overrides the entry of the core component.

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

The following entries are merged.

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

Note, however, that particular address formats can be overridden.

For example, a core component has 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>
```

Your custom component has 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>
```

It is the custom definition that appears in the final merged *curam-config.xml* file. This is because both address formats have the same name (`US`).

Dividing the configuration file

You can divide the *curam-config.xml* file into more manageable chunks. You can save one part of the configuration in a file with a different name.

Taking the previous address format configuration as an example, 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>
```

Save this with a file name that ends with *-config.xml* anywhere in your component, for example *address-config.xml*. The file must have the same `APP_CONFIG` root element as the full *curam-config.xml* file. If 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.

POPUP_PAGES

See 8.18 Pop-up pages on page 255.

MULTIPLE_POPUP_DOMAINS

See 8.18 Pop-up pages on page 255.

ERROR_PAGE

If an error occurs at run-time, the user is redirected to a page that is defined by the ERROR_PAGE setting. Depending on the error cause, two types of error page can be provided for reporting system or application failures. Alternatively, you can define a default page for reporting both kinds of errors.

This ERROR_PAGE section example is a system error.

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

The ERROR_PAGE section example has one default page.

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

Note: When overriding the `ERROR_PAGE` setting, your custom configuration cannot 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 msut use a `TYPE` attribute and must override both supported types of error page.

MULTIPLE_SELECT

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

A Multiple Select section example:

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

FILE_DOWNLOAD_CONFIG

For more information about file downloads, see `ACTION_CONTROL` element on page 326.

ENABLE_COLLAPSIBLE_CLUSTERS

By default this value is set to `true`. Set to `false` to disable collapsible clusters. A Disable Collapsible Clusters example is shown.

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

APPEND_COLON

Set to `true` to automatically append colons to `FIELD` and `CONTAINER` labels within `CLUSTER` elements. An Append Colon section example is shown.

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

ADDRESS_CONFIG

See 8 Domain-specific controls on page 215.

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. An Admin Section example is shown.

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

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

STATIC_CONTENT_SERVER

Configure static content for Cúram

Kubernetes The procedures in this topic are mandatory for deploying Cúram on Kubernetes.

An application server is optimized to serve dynamic content, while an HTTP server is optimized to serve static content. To enable static content in Cúram, set the *STATIC_CONTENT_SERVER* element in the *`curam-config.XML`* file and perform a full Cúram build as shown in this static content base URL example.

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

The forward slash at the end of the URL in the example is optional. You can also use a relative URL as shown in thisrelative URL example.

```
<STATIC_CONTENT_SERVER>
  <URL>/CuramStatic/</URL>
</STATIC_CONTENT_SERVER>
```

A full build is required to pick up this setting.

Where this option is used, the static content can be packaged by using the *zip-static-content* target available in the `webclient` project. This target creates a .zip file, `StaticContent.zip`, in the `webclient\build` directory. The `StaticContent.zip` file contains all relevant static content to be relocated when the *STATIC_CONTENT_SERVER* setting is enabled. The -*Dstatic.content.zip* setting can be used to overwrite the default .zip file location. All content in the .zip file is stored under a root folder called `WebContent`. A target example is shown.

```
build zip-static-content -Dstatic.content.zip=<myzipfile.zip>
```

The following content is included in the .zip file:

- *WebContent/**/*.htc*
- *WebContent/**/*.html*
- *WebContent/**/*.htm*
- *WebContent/**/*.bmp*
- *WebContent/**/*.cur*
- *WebContent/**/*.gif*
- *WebContent/**/*.ico*
- *WebContent/**/*.jpeg*
- *WebContent/**/*.jpg*
- *WebContent/**/*.mov*
- *WebContent/**/*.png*
- *WebContent/**/*.psd*
- *WebContent/**/*.svg*
- *WebContent/**/*.swc*
- *WebContent/**/*.swf*
- *WebContent/**/*.eot*
- *WebContent/**/*.ttf*
- *WebContent/**/*.woff*
- *WebContent/**/*.woff2*
- *WebContent/**/*.as*
- *WebContent/**/*.js*
- *WebContent/**/*.vbs*
- *WebContent/**/*.css*
- *WebContent/**/*.less*
- *WebContent/**/*.json*

The relocation of static content to a separate server allows for specific cache control response headers to be set for this content. Setting a cache control response header provides an instruction to the browser to cache this content for a period of time; the aim of which is to reduce network traffic and improve performance. The *Expires* and *Cache-control* headers encourage the browser to cache static content. Example Response Headers are shown.

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
Cache-control: max-age=86400
```

The *Expires* value must match the specific formatting shown to be recognized. The *max-age* attribute value is in seconds.

When the headers are set, the browser caches the content until the *max-age* value is reached or the *Expires* date is reached. When cached, no request is made to the server.

Related information

Performance tuning

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. A Field Error Indicators example is shown.

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

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 rights when the page is initially loaded. If a user fails authorization for *any* of the server functions, an authorization error message is displayed and the user is prevented from viewing the page. For example, if a user has authorization rights to access the DISPLAY phase server function, but not the ACTION phase, they cannot view the page.

The SECURITY_CHECK_ON_PAGE_LOAD setting in *`curam-config.xml`*, which is true by default, indicates that authorization checks should be performed before the page is loaded to ensure the user has access rights to *all* server functions referenced by SERVER_INTERFACE elements on the UIM page.

Setting the SECURITY_CHECK_ON_PAGE_LOAD attribute to false disables this initial authorization check and defer authorization to the point at which the server function is invoked. As a result, on an edit page for example, a user would require authorization rights for the DISPLAY phase server function at a minimum. If they did not have authorization rights for the ACTION phase server function, the page displays, but the user receives an authorization error message when the page is submitted.

To set SECURITY_CHECK_ON_PAGE_LOAD, and disable authorization on page load, add the following setting to the `curam-config.xml` file:

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

If the `SECURITY_CHECK_ON_PAGE_LOAD` element is not specified, it defaults to `TRUE`.

There is no security risk associated with this change, but the change has implications for auditing. When the authorization check is performed on page load, by default authorization failures are not added to the AuthorisationLog database table. This behavior can be modified by setting `curam.enable.logging.client.authcheck` to true using the Property Administration screens.

When the authorization check is deferred to the invocation of the server function, i.e. SECURITY_CHECK_ON_PAGE_LOAD is false, authorization failures are always logged. It is not possible to control or disable this behavior. As a result, the risk is that the AuthorisationLog database table is filled with noise in the form of authorization failures that are valid failures based on usage.

ENABLE_SELECT_ALL_CHECKBOX

The multi-select check-box `WIDGET` 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 clear all of the check-boxes at once. An Enable Select All Check-box example is shown.

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

If the `ENABLE_SELECT_ALL_CHECKBOX` element is not specified, it defaults to `FALSE`.

For more information, see the The MULTISELECT widget on page 398.

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>
```

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>
```

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>
```

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>
```

If the `SCROLLBAR_CONFIG` element is not specified, no `LIST` or `CLUSTER` element displays a vertical scrollbar.

PAGINATION

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

An example Pagination configuration is shown.

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

Table 4: Pagination configuration options

Option Name	Required	Default	Description
`ENABLED`	No	true	Enables the ability to page through lists displayed in application pages. Any LIST longer than the configured minimum size display only the first "page" of data and the pagination controls are displayed below the list.
`DEFAULT_PAGE_SIZE`	No	15	Specifies the page size the list gets by default. The page size can be then changed at runtime by the user.

Option Name	Required	Default	Description
`PAGINATION_THRESHOLD`	No	Based on the DEFAULT_PAGE_SIZE value.	Specifies the minimum list size at which pagination is enabled. For shorter lists there is no pagination, even if otherwise pagination is switched on.

Custom resources

You can include custom files in the web application.

Warning: Before you use custom resources, ensure that you understand the affects. It is advised to first view the generated *`WebContent`* folder (located *`webclient/WebContent`*) and to be aware of what files exist in it. Placing a similar file in the *`WebContent`* folder of a component overwrites the currently existing file in the generated *`WebContent`* folder.

Files included in the application in this way take precedence over the merging and overriding process for other resources. For example, if you include a CSS file in this way, the contents of the file is not be included in the CSS overriding process described in CSS on page 62.

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 is 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.

It is also important to note that the files placed in a *`WebContent`* folder within a component are completely ignored during the build process and are not processed. They are merely copied across. For example, if you have a JavaScript properties file in the *`WebContent`* folder of your component, it is not processed.

Finally, when multiple components have a *`WebContent`* folder, they are copied based on component priority, but the copy is time-stamp based. The copy command always uses verbose output for these files so you can see exactly what files are being copied.

Complete the following steps to include files:

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 with any folder structure.
3. When you run the **`client`** build target, these files are copied directly to the *`<client-dir>/WebContent`*, which represents the root of the web application. The folder structure is maintained during the copy.

3 Application configuration

An application is a collection of user interface elements, based on UIM pages or Carbon components, that are combined to create content for a specific user or role. You create web client applications by configuring application configuration files.

An application typically consists 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. You can configure an application by using the relevant XML configuration files.

Related concepts

Cúram applications on page 31
When a user logs in to Cúram, they are presented with a view that is specific to their role, which is an application. An application is a collection of user interface elements, mostly based on UIM pages, combined to create specific content for a particular user or role.

Application user interface overview on page 20
The application user interface contains elements that are implemented in UIM or Carbon components.

Related reference

UIM reference on page 323
User interface metadata (UIM) is an XML dialect that is used to specify the contents of the Cúram web application client pages. UIM files must be well-formed XML.

Related information

Carbon Design System

3.1 Configuration files

Configure applications, sections, tabs and related elements in XML-based configuration files.

The configuration files are in the `<server-dir>\components\<component-name>\clientapps` directory. See Application configuration files on page 63 for more information about 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. The following table provides a summary of the file extensions and related schema files.

Table 5: Configuration Files

File Extensioı	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.

File Extension	Schema File	Description
.ssp	section-shortcut-panel.xsd	Configuration file to define the contents of a section shortcut panel.
.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.

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, see the *Server Developer's 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.

8.1.3.0 Updated button functionality

Manage the button behavior in the internal caseworker application.

About this task

The current internal caseworker application includes a set of user interface components that feature links styled as buttons (LIST/ACTION_SET/ACTION_CONTROL and CLUSTER/ ACTION_SET/ACTION_CONTROLlevel). This approach has led to misunderstandings among users, particularly those who rely on assistive technologies. Screen readers and keyboard navigation interpret links and buttons differently, which can make the application less intuitive for individuals with disabilities. Therefore, improving the user experience necessitates that buttons in the application appear and function distinctly as buttons, adhering strictly to accessibility standards.

The new functionality is enabled by default. Instead of links styled as buttons, we are rendering buttons, ensuring that the buttons exhibit standard button characteristics, ultimately contributing to an improved user experience that is accessible and compliant with recognized standards.

For those who prefer to maintain the previous implementation while assessing the impact of this update, we have introduced a temporary property to disable this functionality. This property will be deprecated in a future release.

If you choose to disable the button implementation, please note that the user interface will be rendered the same as in previous versions.

Note: You must restart the server to apply your changes.

To manage the behavior of buttons within the internal caseworker application, the property curam.temp.cluster.list.buttons.enabled plays a pivotal role. By default, this property is set to true, activating the updated button functionality. If you wish to revert to the previous button behavior, simply change this property to false.

Procedure

1. Log in to the Cúram application as a system administrator.
2. Select **Application Data** > **Property Administration**.
3. Enter *curam.temp.cluster.list.buttons.enabled* in the **Name** field and click **Search**.
4. Select **Edit Value...**.
5. Change the property's value to false and **Save** your changes.
6. Click **Publish** > **Yes**.

3.2 Web client properties

Configure the title that is displayed in the browser tab in the *`CDEJResources.properties`* file. The *`CDEJResources.properties`* file contains values for properties that are used throughout the web client.

The core file is located in *`%CURAM_DIR%\CuramCDEJ\doc\defaultproperties\curam\omega3\i18n`*.

Where *`%CURAM_DIR%`* is the Cúram installation directory, by default *`C:\Merative\Curam\Development`*.

This properties file can be localized. For more information, see Locales. Images that are defined in this file can also be customized for locales.

Customizing the CDEJResources.properties file

To customize the CDEJResources.properties file, use the procedure that is outlined in the following task.

Procedure

1. Create a custom copy in the custom component, for example, webclient\components\custom.
2. Include only the properties that are being overridden.

8.1.3.0 Configuring the Context Panel tooltip

To customise the tooltip shown on the Context Panel button, configure the properties that are outlined in the following task.

Procedure

Add the properties from the following list to the custom CDEJResources.properties file.

1. **context.panel.toggle.expand**
   - Defines the Context Panel button tooltip when Context Panel is collapsed. The default value for this property is “Expand Context Panel”.
2. **context.panel.toggle.collapse**
   - Defines the Context Panel button tooltip when Context Panel is expanded. The default value for this property is “Collapse Context Panel”.

Configuring the browser title

To customize the browser title, configure the properties that are outlined in the following task.

Procedure

- Add the properties from the following list to the custom `CDEJResources.Properties` file:
  - **browser.tab.title**
    Defines the application name that is used in the browser tab title.
  - **browser.tab.title.separator**
    Defines the text that is used to separate the page title and application name strings.
  - **browser.tab.title.application.name.first**
    Controls whether the browser tab title displays the application name before the current page title.

3.3 Applications

An application is a view that is 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.
- The role of the user that this application is intended for.
- A welcome message for the user.
- An application menu, which includes links to the User Preferences dialog, application help, the about box, and to logout of the application.
- A configurable application logo is placed at the far right of the application banner. It can be customized or removed.
- A quick search facility for the application.

The application search is an optional addition to the application banner that 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 combobox, which lists the types of object that you can search for.

- A search button to trigger the actual search.
- An optional link to more search options.

Application 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 the following table. The application banner is configured by using these attributes.

Table 6: Attributes of the application element

Attribute	Description
id	*Mandatory*. 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. See Associate an application with a user on page 115 for more information.
title	*Optional*. 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.
sub-title	*Optional*. 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	*Optional*. 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, %user-full-name, which will be replaced with the users full name. The full name is determined based on the `FirstName` and `Surname` fields on the `Users` database table.
hide-tab-container	*Optional*. 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	*Optional*. 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 Application optional header on page 113 for more information.

Attribute	Description
header-source	*Optional*. A reference to the source that will be used as an additional header. The value of this depends on the value of `header-type`. 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 Application optional header on page 113 for more information.
logo	*Optional*. A reference to the path of an image, e.g. *`CDEJ/themes/v8/images/large-application-logo.svg`* or an image name, e.g. *`large-application-logo.svg`*, where the named image is stored in the application resource store. This is used to configure a custom application logo displayed at the far right of the application banner. The custom application logo is displayed only when the attribute `logo-required` is set to true, otherwise this setting is ignored.
logo-alt-text	*Optional*. The alternative text for the custom application logo specified by the attribute `logo`. It is only used when the custom application logo is displayed on the application banner. Otherwise, the setting for this attribute is ignored.
logo-required	*Optional*. When set to true, in conjunction with the `logo` attribute, the referenced custom application logo is displayed. When set to false, the application logo is not displayed on the application banner.
Context	*Optional*. The unique textual value that allows for specifying the content shaping rules for the particular application. The value matches an entry in the ApplicationContext table and is used by the context-aware page objects and widgets to determine the relevant content. For more information, see *Application Context*.

The `application` element supports the child elements detailed in Application definition on page 105.

Table 7: Supported child elements of the application element

Element	Description
section-ref	*1..n*. The `application` must contain a minimum of one `section-ref` element. Each `section-ref` element references a section to be included in the application. See Application section-ref element on page 110 for more information.

Element	Description
application-menu	*Optional*. Allows for the optional addition of links to the application banner. The links supported include the user preferences editor and application logout. See Application application-menu element on page 107 for more information.
application-search	*Optional*. Allows for the optional addition of a quick search facility on the application banner. See Application application-search element on page 107 for more information.
timeout-warning	*Optional*. Allows for the optional addition of a session timeout modal dialog. See Application timeout-warning element on page 110 for more information.

Application application-menu element

The application menu forms part of the application banner, and allows for the optional addition of up to two links; a link to log out of the application and a link to open the user preferences dialog.

Each link is defined as a child element of `application-menu` element and the supported elements are detailed in the following table.

Table 8: Supported child elements of the application-menu element

Element	Description
preferences	*Optional*. Defines a link to the user preferences dialog. This dialog allows a user to configure customizations for the application view. The title of the `preferences` link is defined using the supported `title` attribute. The value of the `title` attribute should be a reference to an entry in the associated properties file.
logout	*Optional*. Defines a link to allow a user to end their session and logout of the application. The title of the `logout` link is defined using the supported `title` attribute. The value of the `title` attribute should be a reference to an entry in the associated properties file.

Application application-search element

To define the application search, use the *application-search* element.

In its simplest form, the *application-search* element requires two attributes, which are used when only one type of search is available and no combination box is to be displayed:

Table 9: Attributes of the application-search element

Attribute	Description
default-search-page	Optional. A reference to the UIM page that is displayed when users click **Search**. When this attribute is used, it is assumed that there is only one type of search and no search type combination box is displayed.
initial-text	Optional. The text to be displayed in the field as a prompt. This text describes what type of information can be provided for the search, for example, **Enter a participant reference number**. The attribute must reference an entry in the associated properties file.

Note: *smart-navigator* is an optional element that enables Cúram Smart Navigator (quick-search="true") when added to the element *application-search* of the `*.app` files. For more information, see *Enabling or disabling Cúram Smart Navigator by changing the .app files*. You cannot use the attributes *default-search-page* and *initial-text* with *smart-navigator*.

If you want *default-search-page* enabled with *smart-navigator*, then you must add *smart-navigator* by using the *search-pages* child element within the *application-search* element.

The *application-search* element supports two child elements that are used for more complex style searches, as shown in the following table.

Table 10: Supported child elements of the application-search element

Element	Description
search-pages	Optional. Defines multiple types of search, see search-pages on page 108.
further-options-link	Optional. Defines a link to a more advanced search page, see further-options-link on page 110.

search-pages

The *search-pages* element is used when multiple search types are needed, for example, **Person**, **Case**, or **types of search**. Other search types are **Person Surname** and **Person Reference Number**. Each search type is listed in a combination box and a different prompt is displayed in the field depending on the selected entry in the combination box.

The *search-pages* element supports the child elements that are detailed in table 3.

Table 11: Supported child elements of the search-pages element

Element	Description
search-page	*1..n.* Defines a single search type. The attributes of the *search-page* element are defined in Table 12: Attributes of the search-page element on page 109.

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.

Table 12: Attributes of the search-page element

Attribute	Description
type	Mandatory. The unique identifier for the type of search, it is passed as a parameter (*searchType*) to the UIM page that is started when the application search is completed.
description	Mandatory. The text to be displayed for the search option in the combination box. The attribute must reference an entry in the associated properties file.
page-id	Mandatory. A reference to a UIM page that is displayed when a user clicks **Search**.
initial-text	Mandatory. The text to be displayed as a prompt in the field when that business object is selected in the combination box. The attribute must reference an entry in the associated properties file.
default	Optional. A Boolean indicating whether this entry is the default entry to be selected in the combination box. Only one entry can specify the default as true.

Note: Blank values are not allowed in the search type combination box. If the user requires a generic search (for example, across all business objects), they must provide configuration data for this search. For example, a business object of "All" linked to a page that searches across all of the business objects that are defined.

Search pages are linked by using a reference to the UIM page to be opened when a user clicks **Search** is clicked. The UIM pages that are 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 a user enters in the field.
- *searchType*

The selected search type. *searchType* is applicable only where multiple search types are defined.

For more information about creating UIM pages, see 13 UIM reference on page 323.

further-options-link

In addition to multiple search types, the application search also supports a link to a more advanced search page. This search is specified by using the *further-options-link* element, which requires the attributes that are listed in table 5.

Table 13: Attributes of the further-options element

Attribute	Description
description	Mandatory. The text of the link. The attribute must reference an entry in the associated properties file.
page-id	Mandatory. A reference to a UIM page that is displayed when the link is clicked. This UIM page requires no page parameters.

Related reference

Enabling or disabling Cúram Smart Navigator by changing the .app files on page 116
You can enable or disable Smart Navigator for all user roles by changing the *`*.app`* files at build time.

Application section-ref element

An application must reference a minimum of one section, and up to a maximum of five sections, by using the `section-ref` element.

See 3.5 Sections on page 137 for more information.

Table 14: Attributes of the section-ref element

Attribute	Description
id	*Mandatory*. The id of a section configuration file (*`.sec`*).

Application timeout-warning element

Define the session timeout warning for an application by using the `timeout-warning` element.

In its simplest form, the `timeout-warning` element does not require any mandatory attributes. If attributes are omitted the default values are used.

A browser session is timed from when data was most recently sent to or received from the server. In some cases, a user can enter data into the application without realizing that their current session has timed out. When the user does initiate a server call, for example to submit the entered data, the browser prompts the user to reauthenticate to the application. Therefore, the user loses all the data that they had entered. To prevent users losing data when their session times out, you can configure a session timeout warning.

Table 15: Attributes of the timeout-warning element

Attribute	Description
title	*Optional*. Configures the title on the session timeout warning dialog. A property within the associated properties file. This value is used to display the title on the timeout warning dialog.
user-message	*Optional*. Configures the main user message on the session timeout warning dialog. A property within the associated properties file. This value is used to display the main user message within the timeout warning dialog.
quit-button	*Optional*. Configures the text on the quit button of the session timeout warning dialog. A property within the associated properties file. This value is used to display the text on the quit button within the timeout warning dialog.
continue-button	*Optional*. Configures the text on the continue button of the session timeout warning dialog. A property within the associated properties file. This value is used to display the text on the continue button within the timeout warning dialog.
width	*Optional*. Configures the width of the session timeout warning dialog. The width of the timeout warning dialog in pixels.
height	*Optional*. Configures the height of the session timeout warning dialog. The height of the timeout warning dialog in pixels.
timeout	*Optional*. Configures the period of time in seconds that the user has to take action within the timeout warning dialog. The period of time in seconds that the user has to take action in the dialog before the session expires. The countdown timer displayed within the modal starts at this value and counts down to 0:0 until the session times out.

Application context

The application context parameter is specified and configured at the user application level and used by infrastructure, context-aware tags, and renderers to shape the final output (content) according to the application specifics. Application Context ensures batch reusability of UIM pages where most of the page flow and business logic can be shared by separate applications with only content variations across them.

Configuring application context and code tables

The only part of the infrastructure that is context-aware at the moment is the code table infrastructure. The following text describes how to configure the application context and code table infrastructure to achieve the code table content appropriate for the current application context.

Application context is added as an attribute in the required application view, `*.app` file. An example of root element is as follows:

```
<ac:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xmlns:ac="http://www.curamsoftware.com/curam/util/client/
application-config"
                          id="SampleApp"
                          logo="SampleApp.logo"
                          logo-alt-text="SampleApp.logoAltText"
                          curam-logo="SampleApp.curamLogo"
                          title="SampleApp.title"
                          subtitle="SampleApp.subtitle"
                          user-message="SampleApp.UserMessage"
                         context="AppCTX1">
```

The parameter has the following characteristics:

- Optional.
- Is code table code from the ApplicationContext code table.
- Must be made known to the infrastructure as described in the sample XML.

To set up the new application context value into the system, it must be declared in the ApplicationContext code table by adding or editing the appropriate `ApplicationContext.ctx` file. The table fragment with the new context value can be declared in the component under design, as this code table is merged by using the same rules as other code tables in the Cúram application.

The sample XML for the ApplicationContext code table addition.

```
<?xml version="1.0" encoding="UTF-8"?>
<codetables package="curam.util.codetable">
  <codetable java_identifier="ApplicationContext" name="ApplicationContext">
    <code default="true" java_identifier="Sample1" status="ENABLED" value="AppCTX1">
      <locale language="en" sort_order="0">
        <description>Sample Application Context 1</description>
        <annotation></annotation>
      </locale>
    </code>
    <code default="true" java_identifier="Sample2" status="ENABLED" value="AppCTX2">
      <locale language="en" sort_order="0">
        <description>Sample Application Context 2</description>
        <annotation></annotation>
      </locale>
    </code>
  </codetable>
</codetables>
```

The description part is used for display and explanatory purposes. The code must match both the setting in the related application view (the 'context' attribute in the sample .app) and the code tables views that support this application context.

Code table Views that use the Application Context

The context parameter is supported by the code table infrastructure and displays the different set of codes relevant in the active context.

A view is created for a code table that contains the code table codes and values specific for a particular application context. The example context-aware code table describes two such views with different sets of code table codes, one for an application context of *"AppCTX1"* and the other with an application context of *"AppCTX2"*.

If an application has a specified context (for example, *"AppCTX1"*) and the example context-aware code table is accessed, the infrastructure ensures that only the code table codes for that particular context (for example, *"cval1"*, *"cval2"* and *"cval5"*) are returned when that code table is accessed within that application. If no context is specified for that application, all of the codes for that code table are returned.

Sample XML of the context-aware code table.

```
<?xml version="1.0" encoding="UTF-8"?>
<codetables package="sample.package">
  <codetable java_identifier="SAMPLE" name="SampleCodes">
    <code default="true" java_identifier="" status="ENABLED" value="cval1">
      <locale language="en" sort_order="0">
        <description>Description 1</description>
        <annotation/>
      </locale>
    …
    </code>
   …
    <code default="false" java_identifier="" status="ENABLED" value="cval7">
      <locale language="en" sort_order="0">
        <description>Description 7</description>
        <annotation/>
      </locale>
    </code>

    <views>
      <view context="AppCTX1" default_code="cval5">
        <code value="cval1"/>
        <code value="cval2"/>
        <code value="cval5"/>
      </view>
      <view context="AppCTX2" default_code="cval3">
        <code value="cval2"/>
        <code value="cval3"/>
      </view>
    </views>

    </codetable>
</codetables>
```

A code table might specify as many views for different contexts provided the contexts are properly introduced in the ApplicationContext code table.

For more information about code table views and the metadata elements and attributes, see the *Server Developer's Guide*.

Application optional header

You can specify a custom header in addition to, or instead of, the application banner. Define the optional header by using the `header-type` and `header-source` attributes on the application element. Define the optional header 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 Application definition on page 105, 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 about creating and referencing custom widgets, see the *Custom Widget Development Guide*.

Whether a custom widget or HTML fragment is used it must always start with a <div> element.

Application example

This example shows an application that is stored in a file called *SimpleApp.app*.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

Note: In the previous example, a namespace, ac has been declared and all elements are prefixed with the namespace. This is recommended practice. For more information, see Application configuration files on page 63.

The *SimpleApp.app* must have a corresponding *SimpleApp.properties* file, which details the localizable content. For example:

```
SimpleApp.logo=CDEJ/themes/v6/images/application-logo.png
SimpleApp.title=Cúram
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 previous example, the logo image is referencing the default logo image shipped with the 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*.

Associate an application with a user

Map a user to the application and the home page that will be displayed when the user initially logs on. The home page is the initial page, which is displayed in its associated tab.

To map a user to an application and to a home page, configure 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 *CT_APPLICATIONCODE.ctx* 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>
```

Note: For more information about code tables, see the *Server Developer's 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 information about how to associate pages with an application, see 3.10 Opening tabs and sections on page 164.

3.4 Customizing Cúram Smart Navigator

Customize your own search targets and keywords for Smart Navigator. When creating a new search target you must bind the keywords to the search target implementations using Guice injection.

A keyword is a group of one or more words that users use to search the application. Examples of keywords are *integrated*, *integrated case*, or *person search*. Search targets are predefined application pages or tabs, for example *Integrated Case*.

Enabling or disabling Cúram Smart Navigator

You can enable or disable Smart Navigator for each Cúram application role. Each role defines an application in a *`.app`* file, for example the caseworker role is defined in *`DefaultApp.app`*. You can enable or disable enable Smart Navigator for a given role by editing the corresponding application. You can edit the application by editing the *`.app`* file when you are building the application.

Enabling or disabling Cúram Smart Navigator by changing the .app files

You can enable or disable Smart Navigator for all user roles by changing the *`*.app`* files at build time.

Enabling Smart Navigator

Before you build the application, edit the *`*.app`* files directly. For example, edit *`DefaultApp.app`* and replace:

```
<ac:application-search default-search-page="Organization_resolveApplicationSearch"
initial-text="Application.Search.InitialText"/>
```

with:

```
<ac:application-search> <ac:smart-navigator initial-
text="Application.Search.IntSearch.InitialText"
description="Application.Search.IntSearch.Description" default="true" /> </
ac:application-search>
```

You can use any value for the *initial-text* and *description* parameters. In the example the parameters are referenced to properties contained in *`DefaultApp.properties`*.

Disabling Smart Navigator

To disable, Smart Navigator, reverse the changes you made to the *`*.app`* files and rebuild the application.

Related reference

Application application-search element on page 107
To define the application search, use the *application-search* element.

Guide to creating a new search target

Create a search target so that users can navigate to application pages or tabs by using Smart Navigator. A search target is a predefined application page or tab.

1. Create a search target definition in the parent code table *SearchTarget*. Creating search keywords by using code tables on page 119 shows how to create the search target definition "Case".
2. Link a keyword to a search target definition. Binding keywords to search target implementations on page 123 shows an example search target code table entry for the "Case" target.
3. Create a search target Java implementation. Creating search targets on page 121 shows how to implement a search target that links to the **Evidence Page** of a person.
4. Bind the keyword to the search targets. Binding keywords to search target implementations on page 123 shows how to bind keywords to search target implementations by using Guice injection.

Creating search keywords

Create keywords that represent your search target. Your search target is triggered only when the user types one of the keywords that are linked to your search target. By using one or more keywords, users can search the application. Examples of keywords are *integrated* or *integrated case*. You can either create search keywords by using the administration system or by using code tables. Using the administration system is the simpler method.

Linking keywords to search targets

You can combine search targets with the person search by adding keywords. Whenever the keyword is identified, Smart Navigator also searches for its related search target. Search targets can return any object type or any page in the application. The destination page can be a modal dialog or a tab. The standard product comes with a set of initial search targets. You can add more search targets to customize your search.

Keywords are defined as a hierarchy of two code tables:

- Parent Code Table: *SearchTarget* – the Search Target Definition. *SearchTarget* links the keywords to a search target implementation. Every Search Target must have only one item in this code table.
- Child Code Table: *SearchTargetKeyword* – keywords that users type in the search to trigger a specific search target. Each search target can have multiple keywords.

Creating search keywords by using the administration system

As an alternative to creating search keywords in *SearchTargetKeyword*, you can also add keywords in the administration system. You can add multiple keywords to the same search target.

About this task

Add keywords by using the Cúram administration system.

Procedure

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Code Tables**.
3. Enter *Smart Navigator* in the **Name** field and click **Search**.
4. Select **New Item** from the list actions menu **...** of the *Smart Navigator Search Target Keyword* search result.
5. Type in the *item name* and *Technical ID*. *Item name* is the keyword that a user enters in the search, for example *investigation case*. Technical ID can be set to any unique string.
6. Click **Publish** > **Yes**.

   Note: The next step links your keyword code tables to the *Smart Navigator Search Target* code table. If you are creating a new Search target follow steps 7 to 11. Otherwise if you want to add new keywords to an existing search target, go to step 12.

7. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Code Tables**.
8. Enter *Smart Navigator Search Target* in the **Name** field and click **Search**.
9. Select **New Item** from the list actions menu **...** of the *Smart Navigator Search Target* search result.
10. Type in the *item name* and *Technical ID*. *item name* is the Search Target Definition Name that should be referenced in the Search Target Java Implementation. *Technical ID* can be set to any unique string.
11. Click **Publish** > **Yes**.

    Note: The next step defines the hierarchy of the *Smart Navigator Search Target* and *Smart Navigator Search Target Keyword*.

12. Go to **Shortcuts** and browse to **Application Data** > **Code Table Hierarchies**, and select the **SearchKeywords** hierarchy link.
13. Under **Codetables**, expand *Search Target Keyword*.
14. Find the newly added keyword in the list and select **Change parent code** from its list actions menu, this action opens a dialog with all of the parent codes available in a drop-down list. These codes are the items in the parent *SearchTarget* code table.

    Note: Ensure that the locale of both the parent and the keyword code table items match the default server locale. Otherwise, Cúram Smart Navigator does not recognize the keywords

15. Select the category that you want to link the new keyword to and click **Save**. For example, selecting **Investigation case** links the new keyword to the investigation case search target by using the *Investigation Case* entry in *SearchTarget* code table.
16. Click **Publish** > **Yes**.

Results

Smart Navigator recognizes the new keyword.

Creating search keywords by using code tables

If the keywords defined in the *SearchTargetKeyword* code table are not sufficient for your search implementation, you can add keywords to a new or existing search target category code table.

Code tables

Use the following code tables to work with search target keywords:

- *SearchTarget*: the parent code table that defines the search target category.
- *SearchTargetKeyword*: defines the keywords. *SearchTargetKeyword* is the child of *SearchTarget*.

Code tables are defined as a hierarchy. As a limitation of code tables hierarchies, your file should be named `CT_SearchTarget.ctx` so that it can be merged with the existing Search Targets provided in the standard product.

If you are implementing a new search target that uses its own set of keywords, you must add a corresponding entry to *SearchTarget*. *SearchTarget* defines search targets categories and links search target implementations to specific keywords. Each entry in *SearchTarget* has corresponding keyword entries in *SearchTargetKeyword* code table.

Example code tables

The following code shows an entry in *SearchTarget* and another in *SearchTargetKeyword*, and creates a hierarchy linking both items:

```
<codetables package="curam.codetable" hierarchy_name="SearchKeywords">
  <description>Search Targets and Keywords</description>

  <!--ADD THE SEARCH TARGET DEFINITION -->
  <codetable java_identifier="SEARCHTARGET" name="SearchTarget">

     <code
       default="false"
       java_identifier="CASE"
       status="ENABLED"
       value="T_CASE">

       <locale language="en" sort_order="1">
          <description>Case</description>
          <annotation/>
       </locale>
      </code>
  </codetable>

  <!-- ADD THE SEARCH TARGET KEYWORDS -->
  <codetable
    java_identifier="SEARCHTARGETKEYWORD"
    name="SearchTargetKeyword"
    parent_codetable="SearchTarget">

    <displaynames>
      <locale language="en">Search Target Keyword</locale>
    </displaynames>

    <code
      default="false"
      java_identifier="CASE"
      status="ENABLED"
      value="AK_CASE"
      parent_code="T_CASE">
      <locale
        language="en"
        sort_order="1">
        <description>Case</description>
        <annotation/>
      </locale>
    </code>
  </codetable>


</codetables>
```

The following fields bind keywords to search target implementations:

- *java_identifier="CASE"* - the name that the generated Java™ accessor variable has in the source code. This sample entry corresponds to *SEARCHTARGET.CASE* variable in Java™.
- *value="T_CASE"* - the value that links the search target category to the keywords in the *SearchTargetKeyword* child code table.

The field *"parent_code"* must match the field *"value"* in the *SearchTarget* code table to link a keyword entry to a search target category and then, in turn, to a search target implementation.

"java_identifier field" is not used in the application by default, but it must be unique within the code table. *"java_identifier field"* has no link to entries in *SearchTarget*.

The *"description"* field contains the actual text keyword that a user enters during the search, *"description"* must be locale-specific.

In the example, keyword "Case" is linked to the "CASE" search target category by matching *"parent_code"* value *"T_CASE"* in *SearchTargetKeyword* to the "value" field value in *SearchTarget*.

You can link multiple keywords in *SearchTargetKeyword* to a single search target category in *SearchTarget* by having the same *"parent_code"* values. For example, you can link keywords "outcome", "plan", and "outcome plan" to the same search target that searches for outcome plans.

After you create both code table entries, run the **`build server database`** command to generate the new variables in Java™. You can then bind search target category and its corresponding keywords to a Java™ search target implementation.

Result

The new keyword is now recognized by Cúram Smart Navigator.

Creating search targets

Create search targets so that users can navigate to application pages or tabs by using Cúram Smart Navigator. A search target is a predefined application page or tab, for example *Integrated Case*.

Create the search target Java™ implementation

After you create the new search target keywords, you must create the search target Java™ implementation.

All search targets must implement the *curam.smartnavigator.target.TargetObjectSearch* interface by using the following methods:

- *List<TargetObjectDtls> searchTargetObject(final SearchPersonDtls person, String queryText)* performs the main search operation and returns search results in a list of *TargetObjectDtls* objects.
- *TargetType getSearchTargetType()* returns the search target type, which can have two states: *REQUIRES_PERSON* or *NO_REQUIREMENT*.
  - *REQUIRES_PERSON* : the search target is called if a person is found in the search only. Otherwise the search target is not run. *REQUIRES_PERSON* searches for objects related to a person such as cases, evidence, and eligibility.
  - *NO_REQUIREMENT*: the search is always executed regardless of whether a person is found.
- *String getSIDName();* returns a Security Identifier (SID) that checks whether the user has access to the search results. *String getSIDName();* is set to the display facade method signature that is used on the page that the search results link to. For example, this method in a search target that searches integrated cases returns *"IntegratedCase.readCaseDetails1"*, which is the facade display-phase method that is used on the home page of an integrated case.
- *String getIcon();* returns a URL to an icon that the search results display. *getIcon();* is optional, if no icon is configured, a default icon is displayed.

Example search target

Search targets must return a list of Target Objects. Each item returned is displayed in Smart Navigator. For search targets that link to a single page, for example the **Evidence Page Search Target**, a single item is returned from the implementation.

For search targets that perform searches, for example the **Case Search Target**, multiple items can be returned from the implementation. Every item is displayed in Smart Navigator.

The following example shows how to implement a simple Search Target that links the user to the **Evidence Page** of a person.

```
public List<TargetObjectDtls>
  searchTargetObject(final SearchPersonDtls person, final String queryText)
    throws AppException, InformationalException {

  final List<TargetObjectDtls> targetObjects =
    new ArrayList<TargetObjectDtls>();

  final TargetObjectDtls targetObject = new TargetObjectDtls();
  targetObject.url =
    "PDCEvidence_listEvidencePage.do?concernRoleID=" + person.concernRoleId;
  targetObject.objectDescription = "Evidence Page";
  targetObject.openOnModalDialog = false;

  targetObjects.add(targetObject);

  return targetObjects;
}

@Override
public TargetType getSearchTargetType() {

  return TargetType.REQUIRES_PERSON;
}

@Override
public String getSIDName() {
  return "PDCPerson.listEvidenceForParticipant";
}

@Override
public String getIcon() {
  return null;
}
```

Binding keywords to the search target Java™ implementation

After you create the keywords and the search target implementation, you then bind the keyword to the search target. For more information, see *Binding keywords to search target implementations*.

Result

The search target is now recognized by Cúram Smart Navigator.

Related concepts

Creating search keywords on page 117
Create keywords that represent your search target. Your search target is triggered only when the user types one of the keywords that are linked to your search target. By using one or more keywords, users can search the application. Examples of keywords are *integrated* or *integrated case*. You can either create search keywords by using the administration system or by using code tables. Using the administration system is the simpler method.

Related reference

Binding keywords to search target implementations on page 123
Bind keywords to search target implementations by using Guice injection. For more information on Guice, see https://github.com/google/guice.

Binding keywords to search target implementations

Bind keywords to search target implementations by using Guice injection. For more information on Guice, see https://github.com/google/guice.

SmartNavigatorModule

To bind a target search type category from *SearchTarget* code table to a Java™ search target implementation you must implement a new module class. This class extends *AbstractModule* class.

Example *ApplicationSearchModule*:

```
public class SmartNavigatorModule extends AbstractModule {

  @Override
  protected void configure() {

    final MapBinder<String, TargetObjectSearch> mapBinder =
      MapBinder
        .newMapBinder(binder(), String.class, TargetObjectSearch.class);

    mapBinder.addBinding(APPSEARCHTARGET.CASE).to(PersonCaseSearch.class);

  }

}
```

This class creates a Guice binding between the *SearchTarget* code table entry and the search target Java™ implementation.

An example is as follows:

- A MapBinder object of the type *<String, TargetObjectSearch>* is created in the *configure()* method.
- The *mapBinder.addBinding(SEARCHTARGET.CASE).to(PersonCaseSearch.class)* method is called, where:
  - *SEARCHTARGET.CASE* argument is the *java_identifier* value from `CT_SearchTarget.ctx` code table entry, *SEARCHTARGET* is the code table identifier.
  - *PersonCaseSearch.class* is the search target Java™ implementation.

A Guice binding is added between *SEARCHTARGET.CASE* search target category and the *PersonCaseSearch.class* Java™ search target implementation.

Example *SearchTarget* code table entry

The *SearchTarget* code table entry that is being bound in the example *ApplicationSearchModule*:

```
<code
      default="false"
      java_identifier="CASE"
      status="ENABLED"
      value="T_CASE">
      <locale
        language="en"
        sort_order="1">
        <description>Case</description>
        <annotation/>
      </locale>
 </code>
```

Add new module classes to `MODULECLASSNAME.dmx`

You must add new module classes to a *`MODULECLASSNAME.dmx`* file in the corresponding project *`data`* directory so that the Cúram application can use its bindings.

An example entry in *`MODULECLASSNAME.dmx`*:

```

<row>
    <attribute name="moduleClassName">
      <value>curam.SmartNavigator.target.SmartNavigatorModule</value>
    </attribute>
</row>
```

The *<value>* field reflects both the package and the class name.

Note: When you complete this process, you must rebuild your database.

For more information about build targets, see the *Development Environment Installation Guide*.

Result

Now every time a keyword linked to *T_CASE* in the *SearchTargetKeyword* code table is entered in the search, the logic calls the *PersonCaseSearch.class* search target implementation.

Related reference

Creating search targets on page 121
Create search targets so that users can navigate to application pages or tabs by using Cúram Smart Navigator. A search target is a predefined application page or tab, for example *Integrated Case*.

Deleting or disabling keywords

You can delete or disable search keywords that you no longer need. Disabled keywords are removed dynamically from the **KEYWORD** list, deleted keywords are removed from the

KEYWORD list after you rebuild the database. You can also disable a search target by removing all of its associated keywords.

Deleting keywords

Delete keyword entries in the *Search Target* code table by removing their corresponding XML entries from `CT_SearchTarget.ctx`.

Use any text editor to search for and remove keyword entries.

Note: When you complete this process, you must rebuild your database.

For more information about build targets, see the *Development Environment Installation Guide*.

Disabling keywords

Note: This procedure disables keywords. However, if you rebuild the database, the keywords' state reverts to default setting and Smart Navigator does not ignore the keywords' state. To permanently delete a keyword, you must delete it from `CT_SearchTarget.ctx`.

To disable keywords, take the following steps:

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Code Tables**.
3. Enter *Search Target Keyword* in the **Name** field and click **Search**.
4. Expand the *Search Target Keyword* item to display all of the keywords.
5. Locate the keyword that you want to delete, expand its actions menu, and select **Hide**. The **Shown** field changes to **No** on this item.
6. Click **Publish** > **Yes**.

Smart Navigator now ignores the keyword.

Disabling a search target

Disable a search target by removing all of its associated keywords.

If you delete or disable all of the keywords that are linked to a search target, the search target is longer used. To avoid error situations, edit the `CT_SearchTarget.ctx` file, and remove all of the entries that have *parent_code* as a value of the search target that is no longer needed.

You can also disable keyword entries as described in the *Disabling keywords* section. To view the associated *parent_code*, use either `CT_SearchTarget.ctx file` or the *SearchKeywords* code table hierarchy as a reference.

Modifying keywords

You can modify keyword entries in the *Smart Navigator Search Target* code table either by editing the code table using the administration system, or by editing the corresponding XML entries in `CT_SearchTarget.ctx` file. Using the administration system is the simpler method.

Modifying keywords by using the administration system

You can modify keyword entries in the *Smart Navigator Search Target* code table by editing the *Smart Navigator Search Target Keyword* code table using the administration system. Any modifications to keywords are dynamically updated in the keyword prompt box.

About this task

This procedure modifies keywords. However, if you rebuild the database, the keywords' state reverts to their default setting. To permanently modify a keyword, you must edit `CT_SearchTarget.ctx`.

Procedure

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Code Tables**
3. Enter *Smart Navigator Search Target Keyword* in the **Name** field and click **Search**.
4. Expand the *Smart Navigator Search Target Keyword* item to display all of the keywords that are defined.
5. Locate the keyword that you want to modify, expand its actions menu, and select **Edit**. *Item name* is the keyword text that is expected to be used in the application search.
6. Click **Publish** > **Yes**.

Results

The keyword is modified with the changes you made. Log on as caseworker and select **Search** > **KEYWORDS**. Note that changes you made to the keyword are reflected in the **KEYWORD** list.

Modifying keywords in `CT_SearchTarget.ctx`

You can modify keyword entries in the *Smart Navigator Search Target* code table by editing their corresponding XML entries in `CT_SearchTarget.ctx`.

You can edit `CT_SearchTarget.ctx` with a text editor to modify the keyword entries. The field *description* is the actual keyword text that the user is expected to input in the application search. You can change this field to any text string. When you finish your edits, you must rebuild your database.

Keyword example:

```
<code
   default="false"
   java_identifier="CASE"
   status="ENABLED"
   value="AK_CASE"
   parent_code="T_CASE">
   <locale
     language="en"
     sort_order="1">
     <description>Case</description>
     <annotation/>
   </locale>
</code>
```

Result

The keyword is modified with the changes you made. Log on as caseworker and select **Search** > **KEYWORDS**. Note that changes you made to the keyword are reflected in the KEYWORD list.

Translating search targets and keywords

If your implementation of Cúram supports more than one language, you can translate search target descriptions and keywords to support users who use that language. The correct language is used at runtime based on the user's locale. If Cúram supports a single language, but that language is not English, you can edit the keywords in that language. For more information, see *Modifying keywords*.

Related reference

Modifying keywords on page 125

You can modify keyword entries in the *Smart Navigator Search Target* code table either by editing the code table using the administration system, or by editing the corresponding XML entries in `CT_SearchTarget.ctx` file. Using the administration system is the simpler method.

Translating search target descriptions

Procedure

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Code Tables**.
3. Enter *Smart Navigator Search Target* in the **Name** field and click **Search**.
4. Expand *Smart Navigator Search Target* to display all of the targets that are defined.
5. Locate the search target item that you want to translate, select **...** to expand its actions menu, and select **Translate......**.
6. Select **Add Translation...**
7. Select the language and provide the translated text.
8. Click **Save** > **Close**.
9. Go to **Shortcuts** and browse to **Application Data** > **Code Table Hierarchies**, and select the **SearchKeywords** hierarchy link.
10. Under **Codetables**, expand **Smart Navigator Search Target**.
11. Find the correct code table in the list and select **...** > **Edit child codes......** from the list actions menu.
12. Click **Save**.
13. Click **Publish...** > **Yes**.

Results

The target is translated with the changes you made. Log on as caseworker and select **Search** > **KEYWORDS**. Note that changes you made to the target are reflected in the **KEYWORD** list.

Translating search target keywords

Procedure

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Code Tables**.
3. Enter *Smart Navigator Search Target Keyword* in the **Name** field and click **Search**.
4. Expand *Smart Navigator Search Target Keyword* to display all of the keywords that are defined.
5. Locate the keyword that you want to translate, expand its actions menu, and select **Translate......**.
6. Select **Add Translation...**
7. Select the language and enter the translated text.
8. Click **Save** > **Close**.
9. Go to **Shortcuts** and browse to **Application Data** > **Code Table Hierarchies**, and select the **SearchKeywords** hierarchy link.
10. Under **Codetables**, expand **Smart Navigator Search Target Keyword**.
11. Find the correct code table in the list and select **Change parent code......** from the list actions menu.
12. Click **Save**.
13. Click **Publish...** > **Yes**.

Results

The keyword is translated with the changes you made. Log on as caseworker and select **Search** > **KEYWORDS**. Note that changes you made are reflected in the **KEYWORD** list.

Overriding the person search

Overriding the person search.

Extending the *curam.smartnavigator.target.impl.PersonSearchImpl* class

To override the person search, you must first create a class that extends the *curam.smartnavigator.target.impl.PersonSearchImpl* class. The new class contains the following methods that can be overridden:

- *public List<SearchPersonDtls> searchPersonByTermsAndDateOfBirth(final String[] termsToSearch, final String[] allTerms, final Date date);* method searches for a person by a list of terms and a date of birth. The default implementation of this method uses *termsToSearch* field as a list of names that are further broken into *firstname* and *surname*. The *allTerms* field enables the extension of this method so that flags or other information that might be required for the search are available to the implementer. For example, you might want to extend this class and use gender in the search. By default, date of birth parameter is optional and is ignored if set to *null*.

- *public List<SearchPersonDtls> searchPersonByIDAndDateOfBirth(final String alternateID, final Date date);* method searches for people by their ID and a date of birth. By default, date of birth parameter is optional and is ignored if set to *null*.
- *public String getSIDName();* method returns the SID name string in the same way as in a normal *TargetObjectSearch* implementation.

Extending rather than implementing the class enables default methods to be used when you want to override all of the methods that are not required. For example, you can have a custom method to search people by ID, and still be able to use the default *searchPersonByNameAndDateOfBirth* method. Use the new person search implementation only for methods that you want to customize.

Adding a new Guice binding

After you create the person search implementation, you must add a Guice binding. For more information, see *Binding keywords to search target implementations*. However, instead of using the standard binding to a code table entry, you must bind the new search implementation to a specific key string value - *custPerson* (or *SmartNavigatorConstants.kPersonSearchKeyCustom*). The following example shows binding a custom person search for a Case search:

```
public class CustomNavigatorModule extends AbstractModule {

  @Override
  protected void configure() {

    final MapBinder<String, SmartNavigatorPersonSearch> mapBinder = MapBinder
      .newMapBinder(binder(), String.class, SmartNavigatorPersonSearch.class);

    mapBinder.addBinding("custPerson").to(CustomPersonSearch.class);

  }
```

Adding a new module class to *`MODULCLASSNAME.dmx`*

You must add any new module class to a *`MODULCLASSNAME.dmx`* file in the corresponding project *`"data"`* directory so that Cúram applications can use its bindings.

This is an example entry in *`MODULECLASSNAME.dmx`*:

```
<row>
 <attribute name="moduleClassName">
  <value>com.ibm.curam.extension.CustomNavigatorModule</value>
 </attribute>
</row>
```

The *<value>* field reflects both the package and the class name. When you finish adding a module class, you must rebuild the database.

For more information about build targets, see the *Development Environment Installation Guide*.

Related reference

Binding keywords to search target implementations on page 123
Bind keywords to search target implementations by using Guice injection. For more information on Guice, see https://github.com/google/guice.

Customizing case search results

When caseworkers search by a reference number or by using the *case* keyword, you can apply customizations that determine the results that are returned and how they are displayed.

Overriding default filtering of case search results

When a caseworker searches by a reference number or by using the *case* keyword, the search results are filtered and results that the user is not authorized to view are removed. To modify this behavior, you can either override or extend the default filter.

When searching by reference number or by using the *case* keyword, the results are passed through the bound implementation of *curam.smartnavigator.core.util.SearchResultFilter*. The default implementation *curam.smartnavigator.core.util.SearchResultFilterImpl* removes any results that are of type Participant Data Case, CT2001, or that the user is not authorized to view.

The authorization check maps a security identifier (SID) name to the case type code of the result and checks the logged in user against that SID. If the user does not pass the authorization check or if there is no mapping for the case type code, the result is removed from the list before the results are returned.

The default implementation provides the following case type code to SID name mapping:

- CT5 (Integrated Case): IntegratedCase.readCaseDetails1
- CT2 (Product Delivery): ProductDelivery.readHomePageDetails1
- CT2000 (Investigation): InvestigationDelivery.readHomePageDetails1
- CT4 (Liability): ProductDelivery.readHomePageDetails1
- CT1 (Service Plan): ServicePlanDelivery.readHomePageDetails1
- CT10201 (Application Case): ApplicationCase.viewApplicationHomeDetails

Overriding the default case search results implementation

You can override the default case search results implementation by providing a custom implementation of *curam.smartnavigator.core.util.SearchResultFilter*. This approach is advised when the default filtering approach based on mapping case type codes to SIDs is not sufficient.

Implementing the *curam.smartnavigator.core.util.SearchResultFilter* class

To override the default *SearchResultsFilter* implementation, you must first create a class that implements the interface *curam.smartnavigator.core.util.SearchResultFilter*. The new class contains the following method that you must implement:

- *boolean excludeResult(CaseHeaderDetails caseHeaderDetailsResult) throws AppException, InformationalException;* method is used to filter case search results that should not be returned to the user. Return *true* if the case should be excluded from the results that are returned, otherwise *false*.

Adding a new Guice binding

Guice bindings are used to register the custom implementation.

```
public class CustomNavigatorModule extends AbstractModule {

  @Override
  protected void configure() {

       bind(SearchResultFilterImpl.class)
          .to(CustomSearchResultFilterImpl.class);

  }
}
```

Extending the default search results implementation

You can extend the default implementation *curam.smartnavigator.core.util.SearchResultFilterImpl* to modify the SID name that is returned for a case type code. To do this, you must override the *getSidName* method to return results for case type codes.

Extending *curam.smartnavigator.core.util.SearchResultFilterImpl*

To extend the default *SearchResultFilter* implementation,*SearchResultFilterImpl*, you must first create a class that extends *curam.smartnavigator.core.util.SearchResultFilterImpl*. You must override the following method:

- *protected String getSIDName(final CaseHeaderDetails caseHeaderDetailsResult);* method returns the SID Name that is mapped to the case type code or *null* if not mapped. This result is filtered (removed) from the results if the SID name returned is *null* (not mapped) or the logged in user is not authorized for the SID name returned.

Adding a new Guice binding

Guice bindings are used to register the custom implementation.

```

Adding a new Guice binding

public class CustomNavigatorModule extends AbstractModule {

  @Override
  protected void configure() {

       bind(SearchResultFilterImpl.class)
          .to(CustomSearchResultFilterImpl.class);

  }
}
```

Adding a new module class to `MODULCLASSNAME.dmx`

You must add any new module class to a `MODULCLASSNAME.dmx` file in the corresponding project `"data"` directory so that Cúram applications can use its bindings.

This is an example entry in *`MODULECLASSNAME.dmx`*:

```
<row>
 <attribute name="moduleClassName">
  <value>com.ibm.curam.extension.CustomNavigatorModule</value>
 </attribute>
</row>
```

The *<value>* field reflects both the package and the class name. When you finish adding a module class, you must rebuild the database.

For more information about build targets, see the *Development Environment Installation Guide*.

Overriding default descriptions for case search results

When a caseworker searches by reference number or by using the *case* keyword, the case name that is displayed next to the number in the results is provided by an implementation of *curam.smartnavigator.core.util.CaseTypeDescriptionResolver*. You can provide a custom implementation to add descriptions for case types that are not catered for by the default implementation or to change the case name that is displayed by the default implementation.

Implementing the *curam.smartnavigator.core.util.CaseTypeDescriptionResolver* class

To override the default *CaseTypeDescriptionResolver* implementation, create a class that implements the interface *curam.smartnavigator.core.util.CaseTypeDescriptionResolver*.

The new class contains the following method that you must implement:

- *ProductTypeDescription determineProductTypeDescription(CaseIDAndTypeKey key) throws AppException, Informational Exception;* method is used to return the name to use with a case search when displaying the result.

The result that the Smart Navigator displays is a combination of the description and the reference number in the format: *ProductTypeDescription.productTypeDescription, referenceNumber*. For example, Application, 257.

Adding a new Guice binding

Guice bindings are used to register to the custom implementation.

```
public class CustomNavigatorModule extends AbstractModule {

  @Override
  protected void configure() {

    final MapBinder<String, CaseTypeDescriptionResolver> mapBinder =
      MapBinder.newMapBinder(binder(), String.class,
        CaseTypeDescriptionResolver.class);

    mapBinder.addBinding("custRes")
      .to(CustomCaseTypeDescriptionResolverImpl.class);
  }
```

Adding a new module class to *`MODULECLASSNAME.dmx`*

You must add any new module class to a *`MODULCLASSNAME.dmx`* file in the corresponding project *`"data"`* directory so that Cúram applications can use its bindings.

For example:

```
<row>
 <attribute name="moduleClassName">
  <value>com.ibm.curam.extension.CustomNavigatorModule</value>
 </attribute>
</row>
```

The *<value>* field reflects both the package and the class name. When you finish adding a module class, you must rebuild the database.

For more information about build targets, see the *Development Environment Installation Guide*.

Modifying search targets redirect URLs

Modify the search target redirect URLs if you want to use a different URL to the search target pages that are used in Smart Navigator.

About this task

Modify the search target redirect URLs if you want to use a different URL to the search target pages. For example you could change the URL that points to a person's **eligibility page**.

Procedure

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Property Administration**.
3. Search for **smart**.
4. Select the select the actions menu **...** of the property whose the URL you want to change, for example **curam.smartnavigator.person.evidence.url**.
5. Select **Edit Value...**.
6. Change the property's value and **Save** your changes.
7. Select **Publish**.

Results

Log in as caseworker, search for the target URL you changed and observe that the URL is redirected.

Setting the preferred tabs by populating the attribute *preferredTabs*

To set the preferred tabs that are used by Smart Navigator, populate the *preferredTabs* attribute of the object *TargetObjectSearch*.

Before you begin

When you are adding a new search target, you might find that the page that you are targeting could be displayed on multiple tabs. If the page you want to redirect to is defined in more than one tab, you must specify the tab or tabs to target. Use the preferred tabs option to specify the targeted tabs. For more information on specifying tabs, see *Page to tab and tab to section associations*. Use preferred tabs only when there is more than one tab configured for a page.

Associating tabs to a search target

Decide which tab, or the ordered list of preferred tabs, you need to associate to your search target. Then, populate the attribute *preferredTabs* of the object *TargetObjectSearch* in your search target implementation. The following code snippet shows an example of setting *"ProspectPersonHome"* as the preferred tab of the target *final SearchPersonDtls person*:

```
    public List<TargetObjectDtls> searchTargetObject(final SearchPersonDtls person,
     String queryText)
      throws AppException, InformationalException {


     final List<TargetObjectDtls> references =
       new ArrayList<TargetObjectDtls>();


     final TargetObjectDtls objReference = new TargetObjectDtls();

     objReference.preferredTabs = "ProspectPersonHome";

     objReference.url = " ProspectPerson_resolveHomePagePage.do?concernRoleID=
  "+person.concernRoleId;

     // more code to populate objReference .........

     references.add(objReference);

     return references;

 }
```

Result

Log in as caseworker and search for a target that you have changed. Note that the search returns content in the updated tabs.

Related reference

Page to tab and tab to section associations on page 166
A page is associated with a tab based on the navigation configuration for the tab. A tab is associated with a section through the section configuration file.

Enabling or disabling recent searches

To see recent searches, users can click inside the Smart Navigator **Search** box before inserting text. By default, the last six search items are displayed.

About this task

Change the *curam.smartnavigator.search.history.threshold* property to enable or disable recent searches. You can also increase or decrease the number of recent searches that are saved.

Procedure

1. Log in to the Cúram application as a system administrator.

2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Property Administration**.
3. Search for *smart*.
4. For the property *property curam.smartnavigator.search.history.threshold*, select the actions menu **...** > **Edit Value**.
5. Set the property to the desired value and **Save** your changes.

```
 0 Do not save recent searches
 X Save X number of recent searches per person
-1 Save all recent searches per person
```

6. Click **Publish** to apply your changes in the application.

Results

Log in as caseworker and run some searches. If you have set *property curam.smartnavigator.search.history.threshold* to *-1*, you will see your searches in the **Search** box.

Setting the debounce timeout

As a user types in the input field, Cúram Smart Navigator searches for results. However, the search is not performed on every keystroke. The debounce technique groups every keystroke into a single event until the user stops typing for a specified time.

About this task

By default, the debounce timeout is set to 500 milliseconds (ms), so the search is triggered only when the user stops typing for 500 ms. You can also disable *search as you type*, in this case the search triggers only if the user presses **Enter** or clicks 🔍 .

To set the debounce timeout, take the following steps:

Procedure

1. Log in to the Cúram application as a system administrator.
2. Browse to **System Configurations** > **Shortcuts** > **Application Data** > **Property Administration**.
3. Search for the property *curam.smartnavigator.search.debounce.timeout*.
4. Select the actions menu **...**.
5. Edit the value of *curam.smartnavigator.search.debounce.timeout* as required. The value is defined in milliseconds. To change the default debounce to 750 milliseconds for example, set the value of *curam.smartnavigator.search.debounce.timeout* to *750*. To disable the debounce feature, set the value to *-1*.

Results

The new debounce timeout is defined. If the value is set to -1 searches run only when the user presses **Enter** or clicks 🔍 .

Implementing a navigation hook

Smart Navigator provides a hook where customized business logic can be added between the click of a result and navigation to the related page.

About this task

Smart Navigator enables a user to directly navigate from a search result to the intended page without requiring the user to traverse intermediate pages. However, there might be scenarios where the direct navigation from a search result to a page is not the intended behavior and therefore the direct navigation needs to be intercepted. The proceeding JavaScript hook addresses the problem by providing a mechanism to insert custom logic between when the user clicks the result and when the user navigates to the related page. For example, a custom implementation might open a modal when the user clicks a result to display information that might be considered important but would otherwise be bypassed. To provide an implementation of the hook, apply the proceeding steps.

Procedure

Use the proceeding sample to create the file *`SearchMultipleTextBoxHookPoints.js`* in `webclient/components/custom/WebContent/CDEJ/jscript/curam/widget`. Where it is indicated, use custom implementation.

```
/**
 * @name curam.widget.SearchMultipleTextBoxHookPoints
 *
 * API for implementing hook points exposed by SearchMultipleTextBox (Smart Navigator).
 *
 */
define([
        "dojo/topic"], function(topic) {
  curam.define.singleton("curam.widget.SearchMultipleTextBoxHookPoints", {
  /**
    * Implement this function in order to add custom processing between
    * click of a Smart Navigator result and the rendering of the page.
    *
    * This hook cannot alter navigation,  Smart Navigator is responsible
    * for completing navigation, that is, saving to history and rendering
    * the URL.
    *
    * If this hook is used to interact with the user then a modal approach
    * should be used to prevent page navigation given when this hook
    * completes the URL will be rendered by Smart Navigator.
    *
    * @param data Object holding:
    * url being navigated to
    * concernRoleId: concernRoleID of individual if result is a person or keyword of a
 person.
    *
    * @return publish message '/smartnavigator/prenavigationhook/completed' to hand
 back control.
    */
    preNavigationHook: function(data){

      // Custom implementation goes here

      // On completion, publish message to hand control back to smart navigator.
      topic.publish('/smartnavigator/prenavigationhook/completed');
    }
   });
  return curam.widget.SearchMultipleTextBoxHookPoints;
});
```

The final step in the hook implementation must be to assign control back to Smart Navigator so that navigation can be completed.

As shown in the preceding sample, control is assigned back to Smart Navigator by using the following code:

```
topic.publish('/smartnavigator/prenavigationhook/completed');
```

For more information, see the *Participant Guide*.

3.5 Sections

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 user activities. The following types of sections are recommended:

- **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 provides a summary of significant information and quick links to common activities.
- **Workspace**
  The Workspace section contains most user tasks for the user role.
- **Inbox**
  The Inbox section is where the user can access their currently allocated work.
- **Calendar**
  The Calendar section contains a calendar of the user's activities and schedules.
- **Reports**
  The Reports section contains reports that are relevant for the user.

Section 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 the following table.

Table 16: Attributes of the section Element

Attribute	Description
id	*Mandatory*. The unique identifier for the section, which must match the name of the file. This is used when referenced from an application (*`.app`*) configuration file.
title	*Mandatory*. The text for the title that will be displayed on the section tab. The attribute must reference an entry in the associated properties file.
hide-tab-container	*Optional*. 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.
default-page-id	*Optional*. 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, see 3.7 Tabs on page 143. 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.

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 Associate an application with a user on page 115 for more information.

The `section` element supports the child elements detailed in the following table.

Table 17: Supported Child Elements of the section Element

Element	Description
tab	*1..n*. A reference to a tab to be included in this section. See Section tab element on page 139 for more information.
shortcut-panel-ref	*Optional*. A reference to the section shortcut panel to be included in this section. See Section shortcut-panel-ref element on page 139 for more information.

Section tab element

A section is a collection of tabs. To associate a tab with a section, use the `tab` element. A `section` must define at least one `tab` element and tabs must only ever be referenced by one section in any application. Therefor 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 the following table.

Table 18: Attributes of the tab element

Attribute	Description
id	*Mandatory*. The id of a tab configuration file (`.tab`). See Section tab element on page 139 for more information.

Section shortcut-panel-ref element

Use the `shortcut-panel-ref` element to define the section shortcut panel to add to a section.

Specify only one `shortcut-panel-ref` per section. See 3.6 Section shortcut panel on page 140 for more information.

The attributes of the `shortcut-panel-ref` element are detailed in the following table.

Table 19: Attributes of the shortcut-panel-ref element

Attribute	Description
id	*Mandatory*. The id of a section shortcut panel (`.sec`). See 3.6 Section shortcut panel on page 140 for more information.

Section example

An example shows a section that is stored in a file called *`SimpleWorkspaceSection.sec`*.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

The *`SimpleWorkspaceSection.sec`* must have a corresponding *`SimpleWorkspaceSection.properties`* file, which details the localizable content. For example:

```
SimpleWorkspaceSection.title=Workspace
```

3.6 Section shortcut panel

Each section can optionally contain a section shortcut panel that provides quick links to open content and complete actions within the section. The menu items in the shortcut panel can be divided into categories.

When a section is first opened, the section shortcut panel is collapsed by default, but it can be expanded or collapsed as needed.

Menu items in a shortcut panel that open modal dialogs are identified by an ellipsis (...), which indicates that further actions are needed.

Section shortcut panel 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 the following table.

Table 20: Attributes of the section-shortcut-panel Element

Attribute	Description
id	*Mandatory*. The unique identifier for the section shortcut panel, which must match the name of the file. This is used when referenced from a section (*`.sec`*) configuration file.

Attribute	Description
title	*Mandatory*. 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.

The `section-shortcut-panel` element supports the child elements detailed in the following table.

Table 21: Supported Child Elements of the section-shortcut-panel Element

Element	Description
nodes	*Mandatory*. Groups together multiple child `node` elements. See Section shortcut panel node element on page 141 for more information.

Section shortcut panel node element

Use the `node` element to represent menu items and categories that are 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 3.6 Section shortcut panel on page 140. “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. A modal dialog is a UIM page that opens in a new window, where the parent window cannot be accessed while it is open. For more information, see Modal dialogs on page 366. Where a menu item opens a modal dialog, an ellipsis is appended to the menu item text to indicate that more information is required.
- **separator**

  A separator can be used to add extra space between menu items within a node of type group, that is, a category.

The attributes supported by the `node` element are detailed in the following table.

Table 22: Attributes of the node element

Attribute	Description
id	*Mandatory*. The identifier for the node. This must be unique within the *`.ssp`* file.

Attribute	Description
type	*Mandatory*. The type of node, where three types are supported: • group • leaf • separator
title	*Mandatory*. 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	*Optional*. 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	*Optional*. 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 *modal*. This is only applicable for `node` elements with a type of leaf.
append-ellipsis	*Optional*. 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 to the menu item.

Section shortcut panel example

An example shows a section shortcut panel that is stored in a file called *SimpleShortcutPanel.ssp*.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

The *SimpleShortcutPanel.ssp* must 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
```

3.7 Tabs

A tab typically represents a business object, for example, a Case or a Participant, though it can also be used to represent a logical grouping of information.

The following elements relate to tabs.

- **Tab Title Bar**
  The title bar contains text to identify the current tab.
- **Tab inline actions and the tab actions overflow menu**
  The actions menu provides actions that are associated with the business object that is represented by the tab. The actions can be a mix of menu items and other menus, each of which links to a page that is displayed in the tab content area or a modal dialog.

Standardized behavior for inline menu items on actions menus was introduced in 8.0.3 as optional and became the default behavior in 8.1. For more information, see 3.11 Tab, page, and list actions menus on page 168.

- **Tab Context Panel**
  The context panel is typically used to present summary information about the business object. The summary information is available for every page that is displayed in the content area. The context panel can be collapsed and expanded to provide more space for the tab content area.
- **Tab Content Area**
  A tab consists of one or more pages of information. The pages are displayed in the content area and can be navigated by using the navigation bar.

  - **Navigation Bar**
    The navigation bar contains a number of navigation 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.
  - **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.
  - **Page Content**
    Selecting a navigation tab or page group entry displays the corresponding UIM page content in the content area.
- **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.

A tab supports the ability to dynamically enable or disable, and hide or show, entries in the tab actions menu, the tab navigation bar, and the page group navigation bar. The dynamic content is updated based on configured refresh events. A refresh event updates the specified part of the tab based on the submission of a modal dialog page or when a specific UIM page is loaded in the content area.

Related reference

Tab tab-refresh element on page 148
The `tab-refresh` element allows the tab actions menu, tab navigation and context panel to be refreshed based on different events.

Tab 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 following table shows the required attributes.

Table 23: Attributes of the tab-config Element

Attribute	Description
id	*Mandatory*. The identifier for the tab, which must match the name of the file. The `id` attribute is used to reference the tab configuration from section configuration files (*`.sec`*). See Section tab element on page 139 for more information.

The `tab-config` element supports the child elements that are shown in the following table. See the child topics for more information.

Table 24: Supported Child Elements of the tab-config Element

Element	Description
page-param	*0..n*. Defines a parameter required when opening a tab.
menu	*Optional*. A reference to the actions menu configuration.
context	*Mandatory*. A reference to the UIM page to be used as the tab context panel, or alternatively details of the tab name and title.
navigation	*Mandatory*. A reference to the tab navigation configuration, or alternatively the name of the UIM page that will be opened in this tab.
smart-panel	*Optional*. A reference to the UIM page to be used for the smart panel.
tab-refresh	*Optional*. Defines what part of a tab should refresh under what circumstances.

Tab `page-param` element

The `page-param` element allows for multiple page parameters to be defined for a tab. Each page parameter that is defined maps to the name of a name-value pair. The name-value pair is 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. The id parameter maps to the unique id for the person and is different for both James Smith and Linda Smith.

Table 25: Attributes of the `page-param` element

Attribute	Description
name	*Mandatory*. A unique identifier for the page parameter.

Related reference

Opening tabs and sections on page 164
You can open new sections and tabs by using several methods.

Tab actions `menu` element

The `menu` element contains a reference to the tab actions menu configuration which is maintained in a separate *`.mnu`* configuration file.

The following table shows the attributes of the `menu` element.

Table 26: Attributes of the `menu` element

Attribute	Description
id	*Mandatory*. A reference to the id of a tab actions menu configuration file (*`.mnu`*).

Related reference

Tab actions menus on page 152
You can associate tab-specific actions with a tab to make them available from the tab title bar.

Tab `context` element

The `context` element defines a context panel by referencing a UIM page which forms the content of the context panel.

The `context` element is mandatory. 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, set the height attribute to zero.

Table 27: Attributes of the `context` element

Attribute	Description
page-id	*Optional*. A reference to the UIM page to be used for the content of the context panel. If this is not specified, the `tab-name` and `tab-title` attributes *must* be specified.
tab-name	*Optional*. The text to display in the tab bar. The attribute must reference an entry in the associated properties file.
tab-title	*Optional*. The text to display in the tab title bar. The attribute must reference an entry in the associated properties file.
height	*Optional*. The pixel height of the context panel. This is only relevant if a `page-id` attribute has been specified to define a context panel. The default value if not specified is 150 pixels.

Related reference

[Context panel UIM on page 150](#)
A context panel is a specific type of UIM page identified by the `PAGE` element that contains an attribute of `TYPE="DETAILS"`.

Tab `navigation` element

The `navigation` element defines what pages are opened within a 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`).

Note: The `navigation` element is mandatory and one of either `page-id` or `id` must be specified.

Table 28: Attributes of the `navigation` element

Attribute	Description
page-id	*Optional*. A reference to the UIM page to be opened in the tab. When a link to this UIM page is selected, it automatically triggers the page to be opened in a new tab.
id	*Optional*. A reference to a tab navigation configuration file (`.nav`).

Related reference

[Tab navigation on page 159](#)
Within a tab, you can navigate to the UIM pages that are grouped as part of the tab. Tab navigation includes the Content Area Navigation Bar and the Page Group Navigation Bar components.

Tab `smart-panel` element

The content of the smart panel is defined by a UIM page, referenced by the `page-id` attribute.

Similar to the context panel, the UIM elements that can be used are limited.

Table 29: Attributes of the `smart-panel` element

Attribute	Description
page-id	*Mandatory*. A reference to the UIM page that will be displayed in the smart panel of the tab.
title	*Mandatory*. 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	*Optional*. The initial width of the smart panel when it is expanded. The default value if this attribute is not set is 250 pixels.

Attribute	Description
collapsed	*Optional*. Boolean indicating if the smart panel should be expanded or collapsed by default. The default value if this attribute is not set is true.

Related reference

Context panel UIM on page 150
A context panel is a specific type of UIM page identified by the `PAGE` element that contains an attribute of `TYPE="DETAILS"`.

Tab `tab-refresh` element

The `tab-refresh` element allows the tab actions menu, tab navigation and context panel to be refreshed based on different events.

By default, only the content area of a tab is refreshed when a modal dialog is submitted. When a modal dialog is either closed or canceled without an action being performed, the content area is not refreshed.

The tab actions menu, tab navigation and context panel can all be refreshed based on two events.

- When a specific UIM page is loaded in the content area.
- When a UIM page is submitted from a modal or the content area.

Each element of a tab is refreshed as follows:

- **Tab Actions Menu**
  Refreshing the tab actions menu results in updating the entries in the menu that can be dynamically disabled or hidden. See the related link for more information about 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. See the related link for more information about dynamic support.
- **Context Panel**
  Refreshing the context panel reloads the UIM page that is displayed in the context panel.
- **Content Area**
  Refreshing the content area reloads the UIM page that is displayed in the content area. This refresh option is available for use only where a modal dialog has been opened from the list drop-down panel of a nested expandable list.

  By default only the parent of a list drop-down panel is updated when the modal dialog is submitted. Where the list drop-down panel exists in a nested expandable list, this will result in the parent list reloading and not the entire content area.

The two different type of refresh events can be configured by using the child elements in the following table.

Table 30: Supported child elements of the `tab-refresh` element

Element	Description
onload	*1..n*. Defines a refresh event, where when the specified page is loaded in the content area, the defined parts of the tab are updated.
onsubmit	*1..n*. 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.

onsubmit/onload

The `onsubmit` and `onload` elements both require the same set of attributes, as described in the following table.

Table 31: Attributes of the onload/onsubmit Elements

Attribute	Description
page-id	*Mandatory*. A reference to the UIM page to associate with the refresh event.
context	*Optional*. Boolean indicating if the context panel should be update when the specified page is loaded or submitted.
menu-bar	*Optional*. Boolean indicating if the tab actions menu should be updated when the specified page is loaded or submitted. See the related link for more information about dynamic support.
navigation	*Optional*. Boolean indicating if the tab navigation should be updated when the specified page is loaded or submitted. See the related link for more information about dynamic support.
main-content	*Optional*. Boolean indicating if the main content area should be updated when the specified page is loaded or submitted. 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.

Related reference

Tab actions menu dynamic support on page 157
You can dynamically enable or disable tab actions, or hide or show tab actions. This feature is supported by using a combination of the `dynamic` attribute of the `menu-item` element, the `loader-registry` element and a Java™ loader implementation.

Context panel refresh example

You can configure refresh options for the tab actions menu, tab navigation, and context panel in XML files or in the Administration application. This example outlines how to configure a refresh of the context panel in Administation when a specified page is submitted from a modal.

About this task

This example makes the following assumptions:

- In the caseworker workspace, a number of outstanding verifications exist for evidence in a case.
- You want to configure the context panel to refresh and display the updated number of outstanding verifications when a caseworker creates new evidence that requires verification.

Procedure

1. Log in to Cúram as an administrator.
2. Click **Administration Workspace** and in the Shortcuts panel, click **User Interface** > **Tabs**.
3. Find the tab from where you want to configure a refresh of the context panel. For example, Application.
4. Click the tab ID to open its configuration details and select **Refresh Options**. A list of page IDs for which you can configure refresh options is displayed.
5. Choose an option.
    - Find the ID for the evidence page that you want to configure the refresh options for and click **Edit**.
    - If the evidence page does not exist, click **New** to create a new page ID for the evidence page.
6. In the On Submit section, select the **Enabled** and **Refresh Context** checkboxes and click **Save**.

    Result: When the new evidence is created for the case, the context panel automatically refreshes with the updated number of outstanding verifications. The caseworker does not need to manually refresh the tab to see the updated number of outstanding verifications in the context panel.

Context panel UIM

A context panel is a specific type of UIM page identified by the `PAGE` element that contains an attribute of `TYPE="DETAILS"`.

This type of UIM page can use only the following subset of existing UIM elements:

- `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. Additionally, a mandatory `PAGE_TITLE` element is required to add information to the tab title bar.

Related reference

TAB_NAME element on page 386

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.

PAGE_TITLE element on page 382

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.

Tab example configuration file

An example is provided of a tab configuration file.

The following example shows a tab configuration file called *`SimpleTab.tab`*.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

The *SimpleTab.tab* file should have a corresponding *SimpleTab.properties* file, which details the localizable content, for example:

```
simple.tab.name=Simple Tab
smart.panel.title=Smart Panel
```

3.8 Tab actions menus

You can associate tab-specific actions with a tab to make them available from the tab title bar.

Standardized behavior for inline menu items on actions menus was introduced in 8.0.3 as optional and became the default behavior in 8.1. For more information, see 3.11 Tab, page, and list actions menus on page 168.

The menu items support opening UIM pages as follows:

- In the content area of a tab.
- Opening a modal dialog to compete an action. Menu items that open a modal dialog are identified by an ellipsis (...).

You can download a file directly from a menu item.

The tab actions menu can dynamically hide or show items.

Tab actions menu items that are disabled are shown as unavailable (grayed out).

Tab actions menu definition and `menu-bar` element

Define a tab actions menu 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 .

The attributes allowed on the `menu-bar` element are defined in the following table.

Table 32: Attributes of the `menu-bar` element

Attribute	Description
id	*Mandatory*. 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 by using the `menu` element.

Attribute	Description
max-inline-items	*Optional*. Specifies the number of tab actions menu items to display inline in the tab title bar. The first few items in the menu are displayed inline and the rest are available from the tab actions overflow menu. Two tab actions are displayed inline by default. This attribute is applicable only when the `curam.actionmenus.display-inline.enabled.tab` application property is enabled. The `curam.actionmenus.display-inline.enabled.tab` application property is enabled by default. The value that you set overrides both the default value or any value that is set for the application by the `curam.actionmenus.max-inline-items.tab` application property. For more information about configuring the display of actions menus, see 3.11 Tab, page, and list actions menus on page 168.

A menu definition can be reused and referenced by multiple tab configurations. A menu can consist of menu items and submenus, which you can use to group menu items. The following child elements define the structure of the menu.

Table 33: Supported child elements of the `menu-bar` *element*

Element	Description
menu-item	*0..n*. 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.
submenu	*0..n*. Defines a group of menu items, which form a sub menu.
menu-separator	*0..n*. Defines a separator line between entries in the menu. For inline entries, the `menu-separator` element has no affect.
loader-registry	*Optional*. Defines the server interfaces that can be called to dynamically change the state of menu items.

Tab actions menu menu-item element

An action entry in the tab actions menu is defined by the `menu-item` element.

The attributes of the `menu-item` element are defined in the following table.

A `menu-item` can do the following actions:

- Open a UIM page in the content area of a tab.
- Open a UIM page in a modal dialog.
- Download a file.

Menu items that open modal dialogs are identified by an ellipsis (...), which indicates that further actions are required.

Table 34: Attributes of the `menu-item` element

Attribute	Description
id	*Mandatory*. The unique identifier for the `menu-item`, which must be unique within the configuration file.
page-id	*Mandatory*. A reference to the UIM page to open when the `menu-item` is selected.
title	*Mandatory*. The text that is displayed for the `menu-item`. The attribute must reference an entry in the associated properties file.
open-as	*Optional*. Where set, this attribute indicates that the UIM page to be displayed is opened as a modal dialog. The only value supported is *modal*.
append-ellipsis	*Optional*. A Boolean attribute that indicates whether to display the ellipsis for menu items that open in a modal dialog. The default is true. The attribute is applicable only where the `open-as` attribute has been set. **Note:** Setting this attribute to true where the `open-as` attribute is not set does not add the ellipsis to the `menu-item`.

window-options	*Optional*.
	
	Defines the height and width of a modal dialog opened from the `menu-item`. This is applicable only where the `open-as` attribute is set to `modal`.
	
	The format for the attribute is:
	
	`width=<pixel value>,height=<pixel value>`
	
	For example:
	
	`window-options="width=500,height=300"`
	
	The `height` portion of the `window-options` is optional. If not specified, the height of the dialog is automatically calculated.

dynamic	*Optional*. Boolean indicating that the `menu-item` can be dynamically disabled or hidden. For more information see the related link.
visible	*Optional*. Boolean indicating if the `menu-item` is hidden or visible. The default is true.
type	*Optional*. Defines a `menu-item` that downloads a file when selected. The only value supported is `FILE_DOWNLOAD`. For more information see the related link.

Attribute	Description
description	*Optional*. Defines text which forms a description for the `menu-item`. This is used for administration purposes only. The attribute must reference an entry in the associated properties file.

Related reference

[Tab actions menu dynamic support on page 157](#)
You can dynamically enable or disable tab actions, or hide or show tab actions. This feature is supported by using a combination of the `dynamic` attribute of the `menu-item` element, the `loader-registry` element and a Java™ loader implementation.

[File download menu item on page 157](#)
A `menu-item` can reference a FILE_DOWNLOAD configuration by using the `type="FILE_DOWNLOAD"` attribute.

Tab actions menu `submenu` element

A submenu is a group of menu items and is defined by using the `submenu` element.

The attributes of the `submenu` element are defined in the following table.

Table 35: Attributes of the `submenu` element

Attribute	Description
id	*Mandatory*. The unique identifier for the `submenu`, which must be unique within the configuration file.
title	*Mandatory*. The text to display for the `submenu`. The attribute must reference an entry in the associated properties file.
description	*Optional*. Defines a text description for the `submenu`. This description is used for administration purposes only. The attribute must reference an entry in the associated properties file.

The `submenu` element allows for further submenus to be defined, in addition to including menu items and menu separators. Use the supported child attributes that are defined in the following table:

Table 36: Supported child elements of the `submenu` element

Element	Description
menu-item	*0..n*. 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.
submenu	*0..n*. Defines a further sub grouping of menu items.

Element	Description
menu-separator	*0..n*. Defines a separator between entries in the submenu.

Tab actions menu `menu-separator` element

A tab actions menu, including associated submenus, can include a line separator to divide the entries in the menu.

Define a line separator by using a `menu-separator` element. The attributes of the `menu-separator` are outlined in the following table.

Table 37: Attributes of the `menu-separator` element

Attribute	Description
id	*Mandatory*. The unique identifier for the `menu-separator`.

Tab actions menu `loader-registry` element

The `loader-registry` element defines a list of loader implementations that are used to dynamically enable or disable, and to show or hide, menu items in the tab actions menu.

The following table shows the supported child elements of the `loader-registry` element.

Table 38: Supported child elements of the `loader-registry` element

Element	Description
loader	*1..n*. Defines one or more loader implementations that dynamically set the visibility and enabled state of the menu items.

Related reference

Tab actions menu dynamic support on page 157
You can dynamically enable or disable tab actions, or hide or show tab actions. This feature is supported by using a combination of the `dynamic` attribute of the `menu-item` element, the `loader-registry` element and a Java™ loader implementation.

Tab actions menu `loader` element

The `loader` element defines a single loader implementation that dynamically sets the state of the menu items in a tab actions menu.

The following table shows the attributes of the `loader` element.

Table 39: Attributes of the `loader` element

Attribute	Description
class	*Mandatory*. The fully qualified class name of an implementation of the `curam.util.tab.impl.DynamicMenuStateLoader` interface.

Related reference

Tab actions menu dynamic support on page 157

You can dynamically enable or disable tab actions, or hide or show tab actions. This feature is supported by using a combination of the `dynamic` attribute of the `menu-item` element, the `loader-registry` element and a Java™ loader implementation.

Tab actions menu dynamic support

You can dynamically enable or disable tab actions, or hide or show tab actions. This feature is supported by 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 is 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`*).

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 to define the fully qualified classname that 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:

- The list of menu item identifiers that are identified as dynamic by the `dynamic` attribute on the `menu-item` element.
- A set of name-value page parameters pairs.

The loader implementation determines 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 Javadoc.

Related reference

Tab tab-refresh element on page 148

The `tab-refresh` element allows the tab actions menu, tab navigation and context panel to be refreshed based on different events.

File download menu item

A `menu-item` can reference a FILE_DOWNLOAD configuration by using the `type="FILE_DOWNLOAD"` attribute.

The following sample code shows an example of using the `FILE_DOWNLOAD` element in the *`curam-config.xml`* file:

```
<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 that is specified for the `FILE_DOWNLOAD` element.

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.

The following example 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 previous example.

```
<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>
```

Related reference
ACTION_CONTROL element on page 326
The `ACTION_CONTROL` element defines a link (text based), button or file download link that the user can activate on a page.

Tab actions menu example configuration file

An example is provided of a tab actions menu configuration file.

The following example shows an example tab actions menu configuration file called ***SimpleMenu.mnu***.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

The *SimpleMenu.mnu* must 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
```

3.9 Tab navigation

Within a tab, you can navigate to the UIM pages that are grouped as part of the tab. Tab navigation includes the Content Area Navigation Bar and the Page Group Navigation Bar components.

The following list describes the tab navigation components:

- **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.
- **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.

Selecting a navigation tab or navigation page displays 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 defines when new tabs are opened and determines what UIM page is associated with what tab.

Tab navigation 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 the element are defined in the following table.

Table 40: Attributes of the navigation element

Attribute	Description
id	*Mandatory*. 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 `navigation` element.

The child elements outlined in the following table are used to define the structure of the navigation. For more information, see the child topics.

Table 41: Supported child elements of the navigation element

Element	Description
nodes	*Mandatory*. Groups navigation pages and navigation tabs together.
loader-registry	*Optional*. Defines the server interfaces that can be called to dynamically change the state of the navigation tabs and navigation pages.

Tab navigation nodes element

The `nodes` element groups together the elements that represent navigation tabs and navigation pages.

The elements are outlined in the following table.

Table 42: Supported child elements of the nodes element

Element	Description
navigation-page	*1..n*. Defines a navigation tab that has no page group navigation bar.
navigation-group	*1..n*. Defines a navigation tab which contains a page group navigation bar. This element groups together `navigation-page` elements that form the page group navigation bar.

Tab navigation navigation-group element

The `navigation-group` element defines a navigation tab that contains a page group navigation bar.

The attributes of the element are outlined in the following table.

Table 43: Attributes of the navigation-group element

Attribute	Description
id	*Mandatory*. The unique identifier for the `navigation-group`, which must be unique within the configuration file.
title	*Mandatory*. The text that will be displayed for the navigation tab in the navigation bar. The attribute must reference an entry in the associated properties file.
dynamic	*Optional*. Boolean indicating that the navigation tab can be dynamically disabled or hidden.

Attribute	Description
visible	*Optional*. Boolean indicating if the navigation tab is hidden or visible. The default is true.
description	*Optional*. 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.

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.

Table 44: Supported child elements of the navigation-group element

Element	Description
navigation-page	*1..n*. Defines the set of navigation pages that are grouped together to form the page group navigation bar.

Related reference

Tab navigation dynamic support on page 163
The tab navigation bar and page group navigation bar support the ability to dynamically enable or disable, and hide or show, navigation tabs and navigation pages.

Tab navigation navigation-page element

A `navigation-page` element can represent both a navigation tab and navigation page.

If the `navigation-page` element is defined as a child element of the `nodes` element, it represent a navigation tab which is part of the navigation bar. If the `navigation-page` element is defined as 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 the following table.

Table 45: Attributes of the navigation-page element

Attribute	Description
id	*Mandatory*. The unique identifier for the `navigation-page`, which must be unique within the configuration file.
page-id	*Mandatory*. A reference to the UIM page to open when the navigation tab or navigation page is selected.

Attribute	Description
title	*Mandatory*. The text that will be displayed for the navigation tab or navigation page. The attribute must reference an entry in the associated properties file.
dynamic	*Optional*. Boolean indicating that the navigation tab or navigation page can be dynamically disabled or hidden.
visible	*Optional*. Boolean indicating if the navigation tab or navigation page is hidden or visible. The default is true.
description	*Optional*. 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.

Related reference

Tab navigation dynamic support on page 163
The tab navigation bar and page group navigation bar support the ability to dynamically enable or disable, and hide or show, navigation tabs and navigation pages.

Tab navigation loader-registry element

The `loader-registry` element defines a list of loader implementations that are used to dynamically enable or disable, and hide or show both the navigation pages and navigation tabs.

The following table shows the supported child elements of the `loader-registry` element.

Table 46: Supported child elements of the loader-registry element

Element	Description
loader	*1..n*. 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.

Related reference

Tab navigation dynamic support on page 163
The tab navigation bar and page group navigation bar support the ability to dynamically enable or disable, and hide or show, navigation tabs and navigation pages.

Tab navigation loader element

The `loader` element defines a single loader implementation that will dynamically set the state of the navigation pages and navigation tabs.

The following table shows the attributes of the `loader` element.

Table 47: Attributes of the loader element

Attribute	Description
class	*Mandatory*. The fully qualified class name of an implementation of the `curam.util.tab.impl.DynamicNavStateLoader` interface.

Related reference

[Tab navigation dynamic support on page 163]
The tab navigation bar and page group navigation bar support the ability to dynamically enable or disable, and hide or show, navigation tabs and navigation pages.

Tab navigation dynamic support

The tab navigation bar and page group navigation bar support the ability to dynamically enable or disable, and hide or show, navigation tabs and navigation pages.

Dynamic support is implemented through 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`*).

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 Javadoc.

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.

Related reference

Tab tab-refresh element on page 148

The `tab-refresh` element allows the tab actions menu, tab navigation and context panel to be refreshed based on different events.

Tab navigation example configuration file

An example tab navigation configuration file is provided.

The following example shows an example tab navigation configuration file named *`SimpleNavigation.nav`*.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

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
```

3.10 Opening tabs and sections

You can open new sections and tabs by using several methods.

- A section can be opened directly by clicking the relevant section tab control.
- A tab can be opened directly by clicking 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 any section except the current section.

Opening a section or tab by clicking 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 clicking 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, depending on the section configuration.

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 client automatically determines whether it is be opened in a new tab and if that tab is to be opened in a new section.

Using links to open tabs and sections

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 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, see the related link. For the purposes of this section, selecting a link will apply to any action that can open a new UIM page.

Related reference

UIM reference on page 323
User interface metadata (UIM) is an XML dialect that is used to specify the contents of the Cúram web application client pages. UIM files must be well-formed XML.

Page to tab and tab to section associations

A page is associated with a tab based on the navigation configuration for the tab. A tab is associated with a section through the section configuration file.

Page to tab associations

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`*).

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 Because of the way in which the Cúram client application handles resolve pages and opening new tabs, it is recommended not to use resolve pages in a navigation configuration. A resolve page is a specific type of UIM page that contains only a `JSP_SCRIPTLET` element.

When a link to a resolve page is selected, the client recognises that 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.

Tab to section associations

A tab is associated with a section by listing it through 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.

Related reference

Tab navigation on page 159
Within a tab, you can navigate to the UIM pages that are grouped as part of the tab. Tab navigation includes the Content Area Navigation Bar and the Page Group Navigation Bar components.

JSP SCRIPTLET on page 358
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.

Setting the preferred tabs by populating the attribute preferredTabs on page 133
To set the preferred tabs that are used by Smart Navigator, populate the *preferredTabs* attribute of the object *TargetObjectSearch*.

Tab and section page parameters

The client determines if a new tab is opened based on the page to tab to section association. In addition, existing open tabs, and values of the parameters that are passed to a tab, are also considered.

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 client application automatically determines 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 the following table.

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.

Table 48: Tab Opening Rules

Page to Tab Association	Page Parameter Values	Action
Page maps to current tab	Match	Page opens in current tab

Page to Tab Association	Page Parameter Values	Action
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

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.

Tab ordering

A default tab ordering is configured in the application that applies when you open a new tab. You can change the default tab ordering.

The default behavior when opening a new tab in the application is that the tab opens at the end of the tab list. This behavior can be changed to open new tabs next to the tab where the request was made. This is known as tab ordering.

The `curam.environment.enable.sequential.tabs` application property controls tab ordering. The default value for the tab ordering is set to false.

For more information about configuring application properties, see the *System Administration Guide*.

3.11 Tab, page, and list actions menus

You can associate user actions with tabs, pages, and lists, which users typically access from actions menus. By default, the first few actions menu items are displayed inline and the remaining actions are available from the actions overflow menus. If needed, you can configure the number of inline menu items that are displayed separately for tab, page, or list actions menus, or customize individual actions menus.

Note: Standardized behaviour and more customization options for inline menu items on actions menus was introduced in version 8.0.3 as an optional feature that needed to be enabled. From 8.1 onwards, the standardized actions menus behavior is enabled by default. For more information about enabling inline actions menus in version 8.0.3 and about the actions menus behavior in previous versions, see (Deprecated) Enabling inline menu items for actions menus on page 171.

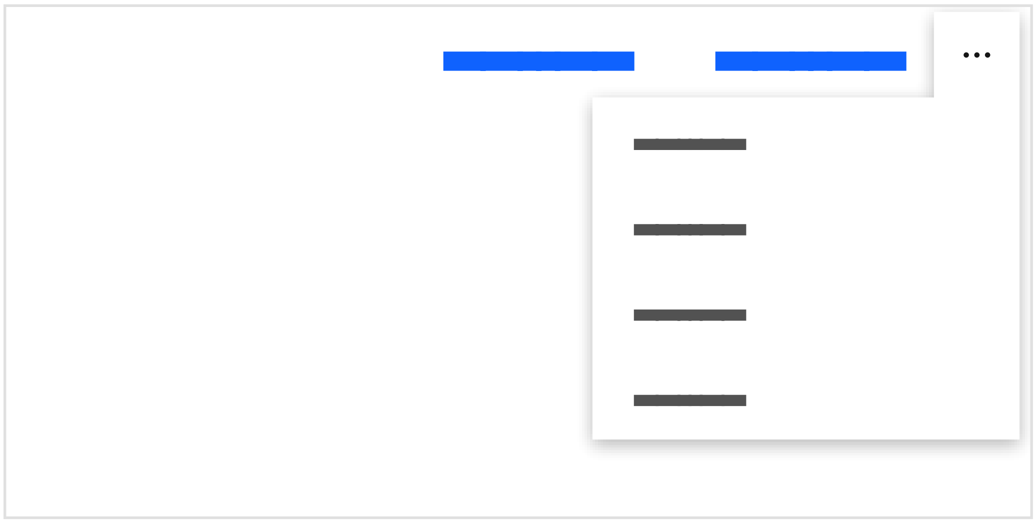
Tab, page, and list actions menus are standardardized as follows:

- **Inline actions**

  The first few menu items are displayed inline, and the rest of the menu items are available from the actions overflow menu. By default, two menu items are displayed inline for tabs, two for pages, and one for lists.

  For inline actions with long text labels, the button increases its width to fill the available space in the UI or to its maximum width. The text then wraps to the next line and the height of the button increases.
- **Actions overflow menus**
  The actions overflow menu contains all of the menu items that are not displayed inline.

For example, the default number of inline actions for tab actions menus is two, so the first two menu items are displayed inline as shown.



You can configure the number of inline actions for actions menus on tabs, pages, and lists application-wide or specifically for individual actions menus.

For example, you can configure tab actions menus to display the first three actions menu items in the menu inline as shown, either application-wide for all tabs, or for an individual tab.

Disabled actions are shown as unavailable (grayed out) for both inline menu items or actions menus items.

Best practices for designing actions

Configuring the most important actions to be available inline rather than in actions menus can significantly reduce the number of clicks that users need to complete tasks.

- **Do**
  - Sort actions in order of importance with the most important first and least important last.
  - Talk to users to understand which actions are the most important.
  - Display actions inline to make them easier to find and use. Consider 2-3 for tab and page actions, and 1-2 for list actions.
  - Consider available space when deciding how many actions to display inline. Test at different viewport widths and browser zoom levels.
  - Use the overflow menu to house less frequently-used actions.
  - Override the default number of inline actions for individual action menus. For example, configure lists to display 1 action inline by default but where a list has only two actions, configure that list to display two actions inline.
  - If the primary action for a list is a link in the first column, you can house all other actions in the overflow.
  - Use short, specific labels to describe the action. Ideally 1-2 words and no more than 3. For example, use **Add Note** instead of **Add**.
  - Organize actions in the overflow menu into logical groupings using a separator. Separators are currently supported only inside the overflow menu.
- **Don't**
  - Fill the title bars completely from left to right.
  - Display more than 5-7 actions inline.
  - Hide all actions in the overflow to make the interface seem cleaner.
  - Use ambiguous labels, such as **Add**, which can be unclear, particularly at page and tab level. Specify what is added, for example **Add note**.

(Deprecated) Enabling inline menu items for actions menus

Standardized inline menu items for actions menus is optional in 8.0.3. Enable inline actions to apply the standardized behavior to all actions menus across the application and enable more configuration options for inline actions. You can enable inline actions separately for tabs, pages, or lists by setting the appropriate application properties.

Before you begin

If you choose not to enable inline menu items for actions menus in 8.0.3, the actions menu behavior remains the same as in previous versions. If you enable inline actions menus, the behavior becomes the same as 8.1 or later. From 8.1 or later, these application properties are no longer needed.

Note: You must restart the server to apply your changes.

About this task

In previous versions, and when inline menu items for actions menus are **not enabled** in 8.0.3, actions menus behave as follows:

- **Tab actions menus**
  All tab actions are displayed in a tab actions overflow menu on the tab title bar.
- **List actions menus**
  All list row actions are displayed in a list actions overflow menu on the list item row.
- **Page actions menus**
  Where the page actions menu has up to two menu items, they display inline on the page title bar area. Where the page actions menu has three or more menu items, all menu items are placed in a page actions overflow menu.

You can use three different application properties to enable inline menu items for actions menus separately for tabs, pages, and lists. When enabled, the same menu behavior and customization options apply to tab, page or list actions menus.

- **Display tab actions menu items inline**
  The `curam.actionmenus.display-inline.enabled.tab` application property enables tab actions menu items to be displayed inline on the tab title bar. By default, the value is false. When set to `true`, the first two tab actions menu items are displayed inline by default. Menu items that are not displayed inline are available from the tab actions overflow menu.
- **Display page actions menu items inline**
  The `curam.actionmenus.display-inline.enabled.page` application property aligns the behavior of page actions menus with the other actions menus and provide the additional inline menu customization options. By default, the value is false. When set to `true`, the first two page actions menu items are displayed inline by default as in previous versions. Menu items that are not displayed inline are available in the page actions overflow menu.
- **Display list actions menu items inline**
  The `curam.actionmenus.display-inline.enabled.list` application property enables list actions to be displayed inline on the list row. By default, the value is false. When set to

`true`, the first list action menu item is displayed inline by default. Menu items that are not displayed inline are available in the list actions overflow menu.

Procedure

1. Log in as *`sysadmin`*.
2. Select **Application Data** > **Property Administration**.
3. Select **Infrastructure - Action Menu Parameters**.
4. You can enable inline menu items for actions menus separately for tabs, pages, and lists.
   - To enable inline menu items for all tab actions menus across the application, set the `curam.actionmenus.display-inline.enabled.tab` property to `true` and publish the change.
   - To enable inline menu items for all page actions menus across the application, set the `curam.actionmenus.display-inline.enabled.page` property to `true` and publish the change.
   - To enable inline menu items for all list actions menus across the application, set the `curam.actionmenus.display-inline.enabled.list` property to `true` and publish the change.
5. Restart the server to apply your changes.

Configuring inline menu items for actions menus

To modify the default number of actions menu items that are displayed inline, you can set an application-wide value in the appropriate application properties for tabs, pages, or lists.

Before you begin

Ensure that inline menu items for actions menus is enabled.

About this task

Use the following properties to modify the default number of actions menu items that are displayed inline for tabs, pages, or lists:

- **Number of tab actions to display inline**
  The `curam.actionmenus.max-inline-items.tab` application property specifies the maximum number of actions menu items to display inline on all tabs across the application. Actions that are not displayed inline are available from the tab actions overflow menu. Ideally, 2-3 tab actions are displayed inline, but our recommended maximum is 4. Set a value of 0 to place all tab actions in the tab actions overflow menu.
- **Number of page actions to display inline**
  The `curam.actionmenus.max-inline-items.page` application property specifies the maximum number of actions menu items to display inline on all pages across the application. Actions that are not displayed inline are available from the page actions overflow menu. Ideally, 2-3 page actions are displayed inline, but our recommended maximum is 4. Set a value of 0 to place all page actions in the page actions overflow menu.

- **Number of list actions to display inline**
  The `curam.actionmenus.max-inline-items.list` application property specifies the maximum number of actions menu items to display inline on the list row. Actions that are not displayed inline are available from the list actions overflow menu. Ideally, 1-2 list actions are displayed inline, but our recommended maximum is 2. Set a value of 0 to place all list actions in the list actions overflow menu.

Procedure

1. Log in as *`sysadmin`*.
2. Select **Application Data** > **Property Administration**.
3. Select **Infrastructure - Action Menu Parameters**.
4. You can modify the default number of inline actions separately for tabs, pages, and lists.
   - To modify the default number of inline actions for all tab actions menus across the application, set the `curam.actionmenus.max-inline-items.tab` property to the required value and publish the change.
   - To modify the default number of inline actions for all page actions menus across the application, set the `curam.actionmenus.max-inline-items.page` property to the required value and publish the change.
   - To modify the default number of inline actions for all list actions menus across the application, set the `curam.actionmenus.max-inline-items.list` property to the required value and publish the change.

Customizing inline menu items for individual actions menus

When inline actions menus are enabled, you can override the default number of inline actions for individual actions menus by applying UIM attributes or tab `menu-bar` attributes to the menus.

Note: Before you begin, ensure that inline menu items for actions menus is enabled.

Tab actions menu example

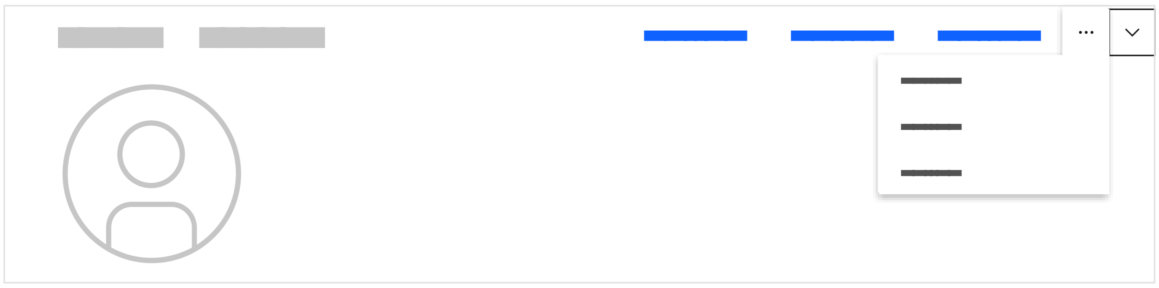
A tab actions menu is defined in a tab actions `menu-bar` element in an XML file with the extension `.mnu` in the *`clientapps`* directory.

You can override the maximum inline items application setting for specific tab actions menus by setting the `max-inline-items` attribute on the tab actions `menu-bar` element, see Tab actions menu definition and `menu-bar` element on page 152.

For example:

```
<mc:menu-item xmlns:mc=<name_space>
id="MenuName" max-inline-items="3"
```

Review the menu items, and ensure that they are ordered with the most important items first. This tab actions menu now displays three inline actions, typically the most important actions, as shown. The number of inline actions on other tab actions menus is unaffected.



Page actions menu example

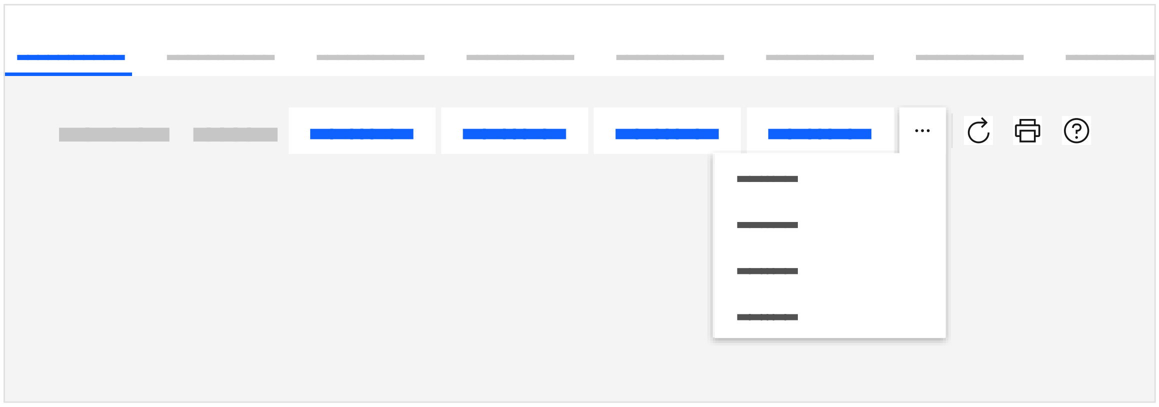
A page level action menu is defined in a UIM `ACTION_SET` element that is defined at the `PAGE` level.

Where needed, you can override the maximum inline items application setting for specific page or list actions menus by setting the `MAX_INLINE_ITEMS` attribute on the UIM `ACTION_SET` element, see `ACTION_SET` element on page 331.

For example, you can configure an `ACTION_SET` element on a UIM page as shown. Review the menu items, and ensure that they are ordered with the most important items first. For example, `Action.Label.1` should be the most important action.

```
</PAGE_TITLE>
  <ACTION_SET MAX_INLINE_ITEMS="4">
    <ACTION_CONTROL LABEL="Action.Label.1"/>
    <ACTION_CONTROL LABEL="Action.Label.2"/>
    <ACTION_CONTROL LABEL="Action.Label.3"/>
    <ACTION_CONTROL LABEL="Action.Label.4"/>
    <ACTION_CONTROL LABEL="Action.Label.5"/>
  </ACTION_SET>
```

This page actions menu now displays four inline actions, typically the most important actions, as shown. The number of inline actions on other page actions menus is unaffected.



List actions menus example

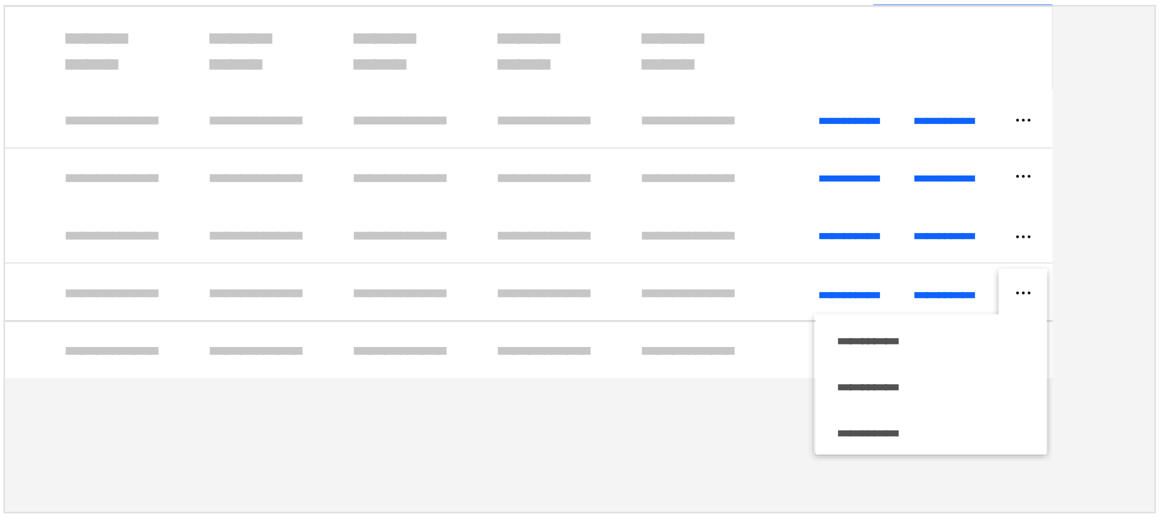
A list action menu is defined in a UIM `LIST` element that contains an `ACTION_SET` element with its `TYPE` attribute set to `LIST_ROW_MENU`.

Where needed, you can override the maximum inline items application setting for specific page or list actions menus by setting the `MAX_INLINE_ITEMS` attribute on the UIM `ACTION_SET` element, see `ACTION_SET` element on page 331.

For example, you can configure an `ACTION_SET` element in a `LIST` element as shown. Review the menu items, and ensure that they are ordered with the most important items first. For example, `Action.Label.1` should be the most important action.

```
<ACTION_SET TYPE="LIST_ROW_MENU" MAX_INLINE_ITEMS="2">
        <ACTION_CONTROL LABEL="Action.Label.1"/>
<ACTION_CONTROL LABEL="Action.Label.2"/>
<ACTION_CONTROL LABEL="Action.Label.3"/>
<ACTION_CONTROL LABEL="Action.Label.4"/>
<ACTION_CONTROL LABEL="Action.Label.5"/>
</ACTION_SET>
```

This list actions menu now displays 2 inline actions, typically the most important actions, on the row as shown. The remaining actions are available from the page actions overflow menu. The number of inline actions on other list actions menus is unaffected.



For information about building the application to apply changes, see Building an application on page 44.

4 Localization

Use this information to learn about the various files that need to be updated when translating the application into a new language.

To simplify the translation process, the language-specific parts of the application are separated out from the application code.

4.1 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.

4.2 File encoding

By default, Cúram supports the development of applications that are localized into many languages. The CDEJ generators support files encoded in the various character encodings appropriate for those languages. Definition of the encoding for a file depends on the type of file. You must set the encoding for the different types of supported files, that is, XML files, Java properties files, or other non-XML files.

XML files

Declare the encoding for XML format files explicitly within the first line of the XML file. The following example shows the format of the XML declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The previous example tells the XML parser that the file uses UTF-8 encoding. If you omit the XML declaration, the parser assumes UTF-8 encoding by default. UTF-8 encoding is based on the Unicode standard, and covers most modern languages and many other languages.

Ensure that the XML declaration matches the actual file encoding. The declaration identifies the encoding but it does not determine the encoding. If you change the declaration, the file encoding does not change automatically. If you use a specialized XML editor application, then it typically recognizes the declaration and changes the file encoding for you. However, plain-text editors do not change the file encoding, so you must ensure that you select the correct encoding in your editor before you save the file.

It is highly recommended that you use UTF-8 encoding for XML files.

Java properties files

Java properties files are used in the application, for example, to define the text strings that appear on client screens. No equivalent of the explicit XML declaration exists for Java properties files. The client generator assumes an encoding for the client properties files based on the default system encoding of the computer that the build is running on. On a Microsoft® Windows® computer in Western Europe, for example, the system encoding is probably Cp1252, the Windows variant of ISO-8859-1. This encoding handles the accented characters of Western European languages but does not cover, for example, Cyrillic or Chinese characters.

If you are building on a computer that does not share its system encoding with the files that are being processed, you must set the ENCODING environment variable. For example, to build a Chinese language web client application on an English language Microsoft Windows computer, you might choose to save your properties files in the UTF-8 encoding. 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 always assumes that properties files use the ISO-8859-1 encoding, which does not suit if you want to use the UTF-8 encoding for localization to, for example, Chinese. To overcome this limitation, the CDEJ automatically translates properties files from your preferred encoding into the encoding required by Java. Your preferred encoding might be either the system default encoding, or the encoding specified by the ENCODING environment variable. This is translation is done automatically during the build process and your original properties files are not 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 is the first property in the file. To resolve the issue, save the file should be saved in the correct encoding, ensuring the BOM character was removed.

Note: The properties files shipped by default with Cúram use ISO-8859-1 encoding, and where necessary use Unicode characters.

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 default Cúram application in a new encoding to build them in a different language environment.

4.3 Locales

The Java locale identifier is used in the Cúram application to identify locales. Most locale-specific information in the application are contained in properties files. For example, "en" (English language) or "en_US" (English language for the United States) are valid locales.

A Java locale identifier has three parts. The language code is required, but the other parts are optional. The individual parts are separated by an underscore character.

- **Language**
  A lower-case, two-letter, ISO-639 code.
- **Country**
  An upper-case, two-letter, ISO-3166 code.
- **Variant**
  A vendor-specific or browser-specific code.

Non-JavaScript property files

To localize Non-JavaScript property files in the application, you must create properties files for each locale. The files for the default locale are named *`SomeFile.properties`*. Identify the other locales by appending an "_" (underscore) character and the locale identifier to the end of the file name.

For example, *`SomeFile_es.properties`* would be the name of the Spanish language version of *`SomeFile.properties`*.

If the application does not find a property in *`SomeFile_es.properties`*, it then searches the default locale *`SomeFile.properties`* file.

After you add localized *`.properties`* files, update the LOCALE_LIST environment variable. This variable defines the list of locales for the client build. For example, set LOCALE_LIST to "en,es" for a default English language application and a Spanish language application. For more information about this setting, see Client application locales on page 41.

The merging of localized properties files from different components is the same as for default locale properties files. For more information about the merging of properties files, see 4.6 UIM externalized strings on page 183.

JavaScript property files

To localize JavaScript files in the application, you must create JavaScript property files for each locale. The files for the default locale are named *`*.js.properties`*. Identify the other locales by appending an "_" (underscore) character and the locale identifier to the *`.js`* extension.

For example, *`SomeJSFile.js_es.properties`* is the Spanish language version of *`SomeJSFile.js.properties`* file. This file is automatically processed by a client build. If a property is not found by the application in *`SomeJSFile.js_es.properties`* file, then the property from *`SomeJSFile.js.properties`* is used.

4.4 Language toggle

You can configure whether internal users can change their default locale to update the language in which field labels, tabs, and shortcut menu items are displayed.

The Cúram application language is displayed to internal users in their default locale. On request, administrators can change any internal user’s default locale to any language that is included in your installation of the application. When an administrator changes a user's default locale, all enabled locales from the Locale code table are displayed for selection. Therefore, ensure that you enable locale codes only for installed languages.

By setting an application property, administrators can enable a Language menu item to globally display in the application menu for all application views. When the property is enabled, internal users can change the language of their application view. To make the Language menu item available in the application menu so that internal users can change their own default locale, set the application property `curam.environment.app.menu.locale.toggle.enabled`. Setting the application property enables the Language menu item globally for all application views.

Note: Although the Language menu item is available for all application views, we recommend that internal users do not change the language of application views that include the following functionality. This functionality is available only in the English language.

- Cúram Income Support application module
- Cúram Child Welfare Structured Decision Making (SDM) add-on module

We also recommend that administrator users do not change the language of administration applications because these application views contain features that do not currently support localization.

4.5 Localizable text translations

Localizable text translations provide a way for organizations to record text content in multiple locales at runtime to support users with multiple locales. Localizable text is one type of text field that is displayed throughout the application. Administrators can configure translation text for fields of this type in the Administration application for localization. When you design a new text field that supports multiple locales, you must model and code the field as a Localizable Text entity.

Modelling and coding the field as a Localizable Text entity ensures that the data that is stored on the business entity represents an identifier instead of a piece of text in a single locale. The identifier relates to the localizable text, which can have multiple associated text translations of the specific text.

The attributes that you can define for the Localizable Text entity for your use case depend on whether the localizable text is short text or rich text.

Attribute	Possible value	Description
shortTextInd	true	• The text is stored in the text translation `shortText` field. • An uppercase version is stored in the text translation `upperShortText` field. • The text is under 3500 characters. • It is possible to write SQL-based searches on the contents of the text.
	false	• The text is stored in the text translation `text` field. • The text is under 5000 characters. • It is not possible to write SQL-based searches on the contents of the text. • Java-based searches on the contents of the text are possible. Note, to be performant the search must use a well filtered and indexed query and not a full table search.
richTextInd	true	• The field dispays a rich version of the text including in lists. For example, the font face, size and color can differ from the standard Cúram design. • Editing the field in the application utilizes the Rich Text Editor. • Due to the character count that the application uses to store the rich text mark-up, the field character limit is reached sooner.
	false	• The text is stored as plain text. • The text is displayed as plain text for viewing and editing.

The Text Translation table stores each translation and its locale for a specific Localizable Text entity.

Fallback for localizable text

Cúram uses a locale fallback mechanism if text translations are not available for display in a user's locale.

Displaying localized text

Cúram always displays text in a user's preferred locale where possible. However, translations might not always be available in the user's locale. Therefore, Cúram uses a locale fallback mechanism to predictably identify and display the best available alternative translation. For localizable text, the fallback mechanism is based on the JDK class `java.util.ResourceBundle`.

How does the fallback identify an alternate translation?

Cúram displays the first available translation that the fallback mechanism identifies in the following order of priority when it retrieves the display text. If no translations are available in this set, then no text is displayed.

Note: In multi-locale environments, it is an administrator's responsibility to configure a locale-specific version of each localizable text record and to prioritize the configuration of the final fallback translation.

Priority	Translation locale	Example
1	User locale	fr_CA
2	User locale language	fr
3	Default server locale	en_US
4	Default server locale language	en
5	Blank	-

The fields in the following table were migrated from legacy fields and add a layer to the standard fallback mechanism.

Entity	Field
WorkQueue	nameTextID
QuickLink	nameTextID
Announcement	announcementTextID, commentsTextID
OrganisationStructure	nameTextID
OrganizationUnit	nameTextID
Position	nameTextID
Job	nameTextID
Location	nameTextID
Slot	nameTextID

For these fields, an additional fallback exists before a blank value is returned. See Priority 5 in the following table.

Priority	Translation locale	Example
1	User locale	fr_CA
2	User locale language	fr
3	Default server locale	en_US
4	Default server locale language	en
5	Legacy text if the field is a migrated legacy field	-
6	Blank	-

Editing localizable text

Administrators can use the **Add Translations** icon to edit translations. In some instances, administrators can edit a translation from the Edit screen for the business object in the Administration application. However, this functionality is limited to the user's current locale. The Edit screen displays the best available translation as the text value. For example, if the user is French but only an English text translation is available, the text is displayed to the user in English. Editing the text translation does not overwrite the English translation. Instead, when the administrator saves the record, a new French translation is created that is displayed to the user in future.

For more information about configuring text translations, see the *System Administration Guide*.

4.6 UIM externalized strings

All string values in UIM files are externalized to `.properties` files.

For an overview of externalized strings, see Externalized strings on page 57.

If `MyPage.uim` is the UIM file, then `MyPage.properties` is the corresponding properties file. For more information about adding localized properties files, see 4.3 Locales on page 179.

The strings are stored in a properties file in the same folder as the page or view file, with the same name and a `.properties` extension. For example, for a `MyPage.uim` file, the strings are stored in the `MyPage.properties` file in the same folder. Similarly, for the `MyView.vim` file, the strings are stored in the `MyView.properties`file.

While UIM documents in the highest priority component override those documents in all other components, properties files in different components are merged in two separate steps. The component order is applied for each properties file, and then the page-view order applied to the resulting properties.

- Individual properties override those with the same property name in lower priority components.
- When a UIM page includes a UIM view (a `.vim` file), the properties for both the page and the view are merged. Where properties share a name, the page properties override the view properties, even if the view property is defined in a higher priority component.

Related reference

Externalized strings on page 57
All string values in UIM files and JavaScript must be externalized, which helps with maintenance and allows the application to be localized. JavaScript, UIM pages, and UIM views can reference externalized strings.

4.7 JavaScript externalized strings

All string values in JavaScript files are externalized to JavaScript property files (`.js.properties` files).

For an overview of externalized strings, see Externalized strings on page 57.

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

The exception 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 in a directory outside the *WebContent* directory.

For more information about adding localized JavaScript properties files, see 4.3 Locales on page 179.

JavaScript properties files with the same name across all components are merged during processing. Any property with the same name is overwritten by the highest component in the component order.

Placeholders in property values are 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 placeholders in a property value.

For example, three placeholders in a property value must be numbered in the range 1 - 3, giving `%1s`, `%2s`, `%3s`. Anything outside this range is not supported.

Related reference

Externalized strings on page 57

All string values in UIM files and JavaScript must be externalized, which helps with maintenance and allows the application to be localized. JavaScript, UIM pages, and UIM views can reference externalized strings.

Accessing properties in JavaScript

Complete these 3 steps to access a JavaScript property.

About this task

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

Procedure

1. Load the resources using `dojo.requireLocalization()`.

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

2. Create an instance of the `curam.util.ResourceBundle` object to be able to access the localized resources.

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

3. Use the `getProperty()` method to access a property on the instantiated `ResourceBundle`.

This example shows how to get a property and how to get a substituted property with two substitutions.

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

4.8 Image.properties

The `Image.properties` file is localized in the same way as other properties files. Place the localized properties file in the same directory as the `Image.properties` file.

For more information about images, see Images on page 58.

For more information about localizing properties files, see 4.3 Locales on page 179.

If the application does not find a property in a localized properties file, it checks the default locale properties file. This is generally true for all properties files but it is particularly useful in the case of `Image.properties`, where only images that contain text need to be localized. Properties for images without text can be defined once in the default locale properties file and they are picked up in all locales.

4.9 Infrastructure widget properties files

Review the list of properties files that are associated with Infrastructure widgets. For example, 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*
- *FileEdit.properties*

Note: The names of the properties files that are 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, so you must take care when you name new properties files.

To customize a widget properties file, create a new version under the *webclient/components/custom* component folder. The default content for the file can be found in the corresponding sample widget properties file in the *<cdej-dir>/doc/defaultproperties/* folder. For each entry in the default version of the file you want to change, add a corresponding entry to your custom file. For more information about localizing these properties files, see 4.3 Locales on page 179.

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 and values that are selected from an input field and two drop-downs in the widget. You can see an example frequency pattern in 8.3 Frequency Pattern Selector on page 217.

Because of grammar differences between languages, the construction of this example frequency pattern sentence can be dramatically changed in other languages. For example, the values that are selected by a user can be reordered. Therefore, placeholders represent user-selected values so that you can localize every frequency pattern as "whole" into all properties in the properties file.

The example frequency pattern contains this property entry from *FrequencyPatternSelector.properties*.

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

The `%ordinal%`, `%dayOfWeekExtended%`, and `%monthInterval%` strings in this property entry are the placeholders that map to the values to be selected from two drop-downs and one input field in the widget.

To use these placeholders properly, you must follow 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 pattern causes 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 cannot add change, add, or remove placeholders for a frequency pattern.

The following table lists a description of all the placeholders that are used in the properties file of this widget.

Table 49: Placeholders used in Frequency Pattern Selector

Placeholder Name	Description
`%dayInterval%`	A day interval. It maps to an input field where you can enter a number for a day interval for a frequency pattern.
`%weekInterval%`	A week interval. It maps to an input field where you can enter a number for a week interval for a frequency pattern.
`%dayOfWeek%`	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.
`%dayOfWeekExtended%`	An extension of the values that are represented by `%dayOfWeek%`, 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.
`%monthInterval%`	A month interval. It maps to an input field where you can enter a number for a month interval for a frequency pattern.
`%ordinal%`	An ordinal, such as first or second. It maps to a drop-down where you can select an ordinal for a frequency pattern.
`%dayIntervalOne%`, `%dayIntervalTwo%`	Two day intervals in a frequency pattern. They are to be used together and map to two input fields where you can enter numbers for two day intervals for a frequency pattern.
`%ordinalOne%`, `%ordinalTwo%`	Two ordinals in a frequency pattern. They are to be used together and map to two drop-downs where you can select two ordinals for a frequency pattern.
`%monthOfYear%`	A month in a calendar year. It maps to a drop-down where you can select a month for a frequency pattern.

The placeholders for each frequency pattern are fixed, so you must ensure that you use them properly when you localize properties in the widget properties file. Aside from the placeholders, customizing the widget properties file is the same as the other infrastructure widgets. The following table lists all the properties and the placeholders they contain for every frequency pattern sentence that is displayed on the Frequency Pattern Selector.

Table 50: Properties of the Frequency Pattern Selector

Property Name	Placeholders
`Text.daily.freq.type.one`	`%dayInterval%`
`Text.daily.freq.type.two`	None.
`Text.weekly.freq.type`	`%weekInterval%`, `%dayOfWeek%`
`Text.monthly.freq.type.one`	`%dayInterval%`, `%monthInterval%`

Property Name	Placeholders
`Text.monthly.freq.type.two`	`%ordinal%`, `%dayOfWeekExtended%`, `%monthInterval%`
`Text.bimonthly.freq.type.one`	`%dayIntervalOne%`, `%dayIntervalTwo%`
`Text.bimonthly.freq.type.two`	`%ordinalOne%`, `%ordinalTwo%`, `%dayOfWeek%`
`Text.yearly.freq.type.one`	`%monthOfYear%`, `%dayInterval%`
`Text.yearly.freq.type.two`	`%ordinal%`, `%dayOfWeekExtended%`, `%monthOfYear%`

4.10 CDEJResources.properties

This properties file can be localized. For more information, see *Locales*. Images defined in this file can also be customized per locale.

Related reference

Optimal browser support on page 205
A number of browsers and a range of browser versions are supported for use with Cúram. Users can be notified when they are not using the optimal version of a supported web browser. You can configure the range of supported versions for a browser, the message that users see, and the frequency at which the message is displayed.

Locales on page 179
The Java locale identifier is used in the Cúram application to identify locales. Most locale-specific information in the application are contained in properties files. For example, “en” (English language) or “en_US” (English language for the United States) are valid locales.

4.11 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 Application configuration properties on page 50 for details.

4.12 Application menu

To localize the application menu on the application banner, set the `LABEL` attribute of the `LINK` element to a key into the *`CDEJResources.properties`* file. Then include the key in the localized version of *`CDEJResources.properties`*.

The contents of the application menu are defined in *`curam-config.xml`*. For non-translated applications, you can put the text that displays on screen directly into this file, in the `LABEL` attribute of the `LINK` element. However, that approach is not suitable for multiple languages. The application checks if the `LABEL` attribute is a key into the *`CDEJResources.properties`* file. If it finds the key, it uses the corresponding value in the menu.

To localize the menu, include the same key in the localized version of *`CDEJResources.properties`*. For information about how to localize properties files such as *`CDEJResources.properties`*, see 4.3 Locales on page 179.

4.13 Tabbed configuration artifacts

Each tabbed configuration artifact has a corresponding properties file for any text that is localizable. To localize this text for a specific language, you must add the locale-specific properties file in the same directory as its associated tabbed configuration artifact in your *`<custom>`* component.

These properties file can be localized as per 4.3 Locales on page 179.

4.14 Runtime messages

The CDEJ runtime messages can be localized or customized by creating a *`RuntimeMessages.properties`* file within the *`<client-dir>/components/<component_name>`* folder. The default content for this file can be found in the *`<cdej-dir>/doc/defaultproperties/`* folder. Messages in this file override the corresponding messages from the *`RuntimeMessages.properties`* that is included with the CDEJ.

Use the standard file naming convention for Java properties files to add locale-specific messages. For example, to create a Spanish version, create *`RuntimeMessages_es.properties`* file.

You don't need 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 are 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 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.

5 Session management

Learn how browser sessions are handled in the Cúram application. 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 are 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 deployment guides for more information on application server configuration.
- The user can explicitly log out, using the log out 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.

5.1 Session Overview

There is a maximum limit on the number of tabs that can be opened per section of an application. The system administrator can configure this limit by updating the `curam.environment.max.open.tabs` property in the system administration application. The default value for the maximum limit of open tabs per section of an application is set to fifteen.

If a user requests to open a tab and the number of open tabs reaches the maximum limit within the current section then an informational modal dialog will be displayed immediately after the tab is initially opened (before content in the tab is displayed). As instructed in this modal dialog, existing open tabs within the current section should be closed before any new tabs can be opened in an application. If the information displayed in the informational dialog is ignored and the user attempts to open more tabs within the current section of an application, the requested tabs will not be opened and an error modal dialog will be displayed instructing that new tabs can only be opened after existing open tabs within the current section of the current application are closed. An error modal dialog can simply be dismissed by clicking on the button on the bottom of the dialog.

The message and title of both the dialog can be customized by customizing by adding the *`GenericModalError.js.properties`* file within the custom component. For more information on localizing JavaScript property files, see 4.2 File encoding on page 177.

The text on the button can be customized by changing the value of the `Text.Ok` property in *`CDEJResources.properties`*. For more information on localizing *`CDEJResources.properties`*, see 4.10 CDEJResources.properties on page 188.

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, the currently open tabs are also restored. There are two exceptions to this:

- If the system administrator has decreased the maximum limit of tabs that can be opened within a section of an application since the termination of the last session then only the new maximum number of tabs within each section will be restored. An error dialog will be displayed informing the user that the maximum limit of open tabs has been exceeded.
- If the system administrator has updated the tab configuration to remove tabs from sections via the User Interface administration screens, then the removed tabs will not be 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.

5.2 Tab Restoration

The list of currently open tabs per user 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 depends on whether it was the result of a browser refresh or the start of a new session when a user logs in.

- `Browser Refresh`

  If the browser is refreshed, tabs are restored to their current state from the web tier session data, for the current user. No tab changes will be lost.

  - The tab that was last selected for the current user in the selected section will remain the selected tab.
  - The selected tab for the current user 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 for the current user are not restored.
- `New Session`

  When a new session starts, usually requiring the user to log in, 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.

For more information about a special case of tab restoration where pages are directly accessed through the browser navigation bar, see direct browsing in 2.6 The application controller JSP and web client URL on page 34.

5.3 Session 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 period. 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.

5.4 Session timeout warning

A browser session timeout warning displays to prevent Cúram users from losing data when their session times out. System administrators can configure the session timeout warning modal text and messages in global properties for internal and external applications, and separately for specific applications.

A browser session is timed from when data was most recently sent to or received from the server. Before a browser session times out, a session timeout warning modal is displayed to users at a configured time. The modal displays a timer to indicate the remaining period before the session times out. Users can either reset the session timeout and continue working in the application, or end the session and quit the application.

In some cases, a user might enter data into the application without realizing that the current session has timed out. When the user does initiate a server call, for example to submit the entered data, the browser prompts the user to reauthenticate to the application and their data can be lost.

Session timeout warning default values

The session timeout warning uses default values that are defined in the *`ApplicationConfiguration.properties`* file and in the *`CDEJResources.properties`* file.

CDEJ resources properties

You can configure the default values of the following properties that are defined in the *`CDEJResources.properties`* file:

- **timeout.warning.modal.title**
  Configures the title that is displayed on the timeout warning modal dialog. The default value is `Timeout Warning`.
- **timeout.warning.modal.user.message**
  Configures the message that is displayed to the user before the session expires. The default value is `You are about to be logged out. Time remaining:`
- **timeout.warning.modal.expired.user.message**
  Configures the message that is displayed to the user after the session expires. The default value is `You have been automatically timed out due to a period of inactivity on your account.`
- **timeout.warning.modal.continue.button**
  Configures the text that is displayed on the **Continue** button in the modal dialog that is displayed to the user before the session expires. The default value is `Continue`.
- **timeout.warning.modal.quit.button**
  Configures the text that is displayed on the **Quit** button in the modal dialog that is displayed to the user before the session expires. The default value is `Quit`.

Application configuration properties

The following default values are defined in the `ApplicationConfiguration.properties` file:

- **Session timeout warning modal width**
  Configures the default width of the session timeout warning modal in pixels. The default value is `580`. You can override the default property value only by customizing the `timeout-warning` element in an application.
- **Session timeout warning modal height**
  Configures the default height of the session timeout warning modal in pixels. The default value is `250`. You can override the default property value only by customizing the `timeout-warning` element in an application.
- **Default buffering period**
  Configures the default buffering period in seconds to allow a server more time to respond to a client request over a slow network. The default value is `20`. You cannot override the default property value.

Customizing the session timeout warning in the caseworker application

Customize the session timeout warning in internal applications, such as the caseworker application, by configuring system application properties and CDEJ resource properties.

About this task

Settings that you customize in CDEJ properties apply to the whole of Cúram.

If the `timeout-warning` element is configured for a specific application, the application configuration takes precedence over the corresponding values that are configured in the application configuration properties and the CDEJ properties. For more information, see the Application timeout-warning element on page 110.

To customize system application properties, do the following preliminary steps:

1. Log on to Cúram as a system administrative user.
2. Click **System Configurations**.
3. In the Shortcuts panel, click **Application Data** > **Property Administration**.
4. Search for and edit each property that you want to configure.
5. To publish the property change, click **Publish**.

Procedure

Application configuration properties

- Customize the following application configuration properties for the session timeout warning as required:
  - **Enable or disable the session timeout warning**
    Edit the curam.environment.internal.enable.timeout.warning.modal application configuration property. The property configures whether the session timeout warning is displayed to users, A valid Boolean value is required, where the default value is `true`.

- **Customize the session timeout warning notice period**
  Edit the curam.environment.internal.timeout.warning.modal.time application configuration property. The property configures the notice period that users are given in seconds, through the display of the session timeout warning, that their browser session is about to time out. For example, if the default browser session length is 30 minutes, and the `timeout` attribute value is configured to `120`, which corresponds to a value of 2 minutes, the session timeout warning is displayed after 28 minutes of inactivity. Then, users must click a button in the user interface to prevent the session from automatically timing out. A valid integer value is required, where the default value is `120`.
- **Customize the session expiry logout page**
  Edit the curam.environment.internal.timeout.warning.modal.logoutpage application configuration property, where the default value is `internal-logout-wrapper`. The property configures the logout page that is displayed when a user's session expires and the user is automatically logged out. The property value must be a valid UIM page.

CDEJ resource properties

- Customize the following CDEJ resource properties for the session timeout warning as required:

  - **Customize the title on the session timeout warning modal dialog**
    Edit the timeout.warning.modal.title CDEJ property. The property configures the title that is displayed on the timeout warning modal dialog. The default value is `Timeout Warning`.
  - **Customize the message in the session timeout warning modal dialog**
    Edit the timeout.warning.modal.user.message CDEJ property. The property configures the message that is displayed to the user before the session expires. The default value is `You are about to be logged out. Time remaining:`
  - **Customize the session expiry message**
    Edit the timeout.warning.modal.expired.user.message CDEJ property. The property configures the message that is displayed to the user after the session expires. The default value is `You've been logged out due to inactivity. Log in to continue.`
  - **Customize the Continue button text in the session timeout warning modal dialog**
    Edit the timeout.warning.modal.continue.button CDEJ property. The property configures the text that is displayed on the **Continue** button in the modal dialog that is displayed to the user before the session expires. The default value is `Continue`.
  - **Customize the Quit button text in the session timeout warning modal dialog**
    Edit the timeout.warning.modal.quit.button CDEJ property. The property configures the text that is displayed on the **Quit** button in the modal dialog that is displayed to the user before the session expires. The default value is `Quit`.

Customizing the session timeout warning in Universal Access

Customize the session timeout warning in external applications, such as the Universal Access application, by configuring system application properties and CDEJ resource properties.

About this task

Settings that you customize in CDEJ properties apply to the whole of Cúram.

If the `timeout-warning` element is configured for a specific application, the application configuration takes precedence over the corresponding values that are configured in the application configuration properties and the CDEJ properties. For more information, see the Application timeout-warning element on page 110.

To customize system application properties, do the following preliminary steps:

1. Log on to Cúram as a system administrative user.
2. Click **System Configurations**.
3. In the Shortcuts panel, click **Application Data** > **Property Administration**.
4. Search for and edit each property that you want to configure.
5. To publish the property change, click **Publish**.

Procedure

Application configuration properties

- Customize the following application configuration properties for the session timeout warning as required:

    - **Enable or disable the session timeout warning**
      Edit the curam.environment.enable.timeout.warning.modal application configuration property. The property configures whether the session timeout warning is displayed to users, A valid Boolean value is required, where the default value is `true`.
    - **Customize the session timeout warning notice period**
      Edit the curam.environment.timeout.warning.modal.time application configuration property. The property configures the notice period that users are given in seconds, through the display of the session timeout warning, that their browser session is about to time out. For example, if the default browser session length is 30 minutes, and the `timeout` attribute value is configured to `120`, which corresponds to a value of 2 minutes, the session timeout warning is displayed after 28 minutes of inactivity. Then, users must click a button in the user interface to prevent the session from automatically timing out. A valid integer value is required, where the default value is `120`.
    - **Customize the session expiry logout page**
      Edit the curam.environment.timeout.warning.modal.logoutpage application configuration property, where the default value is `LogoutWrapper`. The property configures the logout page that is displayed when a user's session expires and the user is automatically logged out. The property value must be a valid UIM page.

CDEJ resource properties

- Customize the following CDEJ resource properties for the session timeout warning as required:

    - **Customize the title on the session timeout warning modal dialog**
      Edit the timeout.warning.modal.title CDEJ property. The property configures the title that is displayed on the timeout warning modal dialog. The default value is `Timeout Warning`.
    - **Customize the message in the session timeout warning modal dialog**
      Edit the timeout.warning.modal.user.message CDEJ property. The property configures the message that is displayed to the user before the session expires. The default value is `You are about to be logged out. Time remaining:`

- **Customize the session expiry message**
  Edit the timeout.warning.modal.expired.user.message CDEJ property. The property configures the message that is displayed to the user after the session expires. The default value is `You've been logged out due to inactivity. Log in to continue.`
- **Customize the Continue button text in the session timeout warning modal dialog**
  Edit the timeout.warning.modal.continue.button CDEJ property. The property configures the text that is displayed on the **Continue** button in the modal dialog that is displayed to the user before the session expires. The default value is `Continue`.
- **Customize the Quit button text in the session timeout warning modal dialog**
  Edit the timeout.warning.modal.quit.button CDEJ property. The property configures the text that is displayed on the **Quit** button in the modal dialog that is displayed to the user before the session expires. The default value is `Quit`.

Customizing the session timeout warning for a specific application

You can configure the session timeout warning individually for each application by configuring the optional `timeout-warning` element.

About this task

Optionally, configure the `timeout-warning` element in the application configuration XML file, which has the extension *.app*. If you configure the `timeout-warning` element in the application, the values takes precedence over both the values that are configured in the system application configuration properties and the default values. For more information, see the Application timeout-warning element.

Procedure

- Configure the following attributes as required in an application's configuration file:
  - **`title`**
    Configures the title that is displayed on the timeout warning dialog.
  - **`user-message`**
    Configures the message that is displayed to the user before the session expires. If the message requires more than two lines of text, to prevent text cutoff from occurring a scroll bar is automatically displayed in the session timeout warning modal. Because the scroll bar is implemented by using CSS styling, it is not possible to disable it by configuring a property.
  - **`expired-user-message`**
    Configures the message that is displayed to the user after the session expires.
  - **`quit-button`**
    Configures the text that is displayed on the **Quit** button in the modal dialog that is displayed to the user before the session expires.
  - **`continue-button`**
    Configures the text that is displayed on the **Continue** button in the modal dialog that is displayed to the user before the session expires.

    - **`timeout`**
      Configures the notice period that users are given in seconds, through the display of the session timeout warning, that their browser session is about to time out. For example, if the default browser session length is 30 minutes, and the `timeout` attribute value is configured to `120`, which corresponds to a value of 2 minutes, the session timeout warning is displayed after 28 minutes of inactivity. Then, users must click a button in the user interface to prevent the session from automatically timing out.
    - **`width`**
      Configures the width of the session timeout warning modal in pixels.
    - **`height`**
      Configures the height of the session timeout warning modal in pixels.
- For an application in Universal Access, you can enable a specific logout page to be associated with the **Quit** button for a modal dialog. On the logout banner menu item that is on the person banner menu, you must set the logout attribute to `true`, as shown in the following example:

```
<ac:banner-menu type="person" title="person.title" page-id="somPageID"/>
<ac:menu-item id="logout" title="menu.logout.title" text="menu.logout.text"
page-id="LogoutWrapper" logout="true"/>
<ac:banner-menu/>
<ac:timeout-warning title="timeout.title"
user-message="timeout.user-message"
expired-user-message = "timeout.expired-message"
continue-button="timeout.continue"
quit-button="timeout.logout"
timeout="300"
width="650"
height="300"/>
```

Example

The following example demonstrates how to specify values for the `timeout-warning` attributes:

```
<ac:timeout-warning title="timeout.title"
user-message="timeout.user-message"
expired-user-message = "timeout.expired-message"
continue-button="timeout.continue"
quit-button="timeout.logout"
timeout="300"
width="580"
height="200"/>
```

Configuring a customized logon page

If a browser session times out because of no user interaction, users are redirected to an application logon page that is specified by the configuration properties. The logon page displays a session expiry message that tells users that they have been logged out because of a period of inactivity on their account.

About this task

In the configuration properties, you can specify the application logon page that is displayed both in the Cúram application and in the Universal access application.

If the application is configured to display a customized logon page instead of the default page, then use the following procedure to insert a customized session expiry message into the customized logon page. If a user's session times out automatically, the customized session expiry message is then displayed in the customized logon page that the user is redirected to.

Procedure

1. To configure the custom logon JSP page, do the following steps:
   a) Import the class `JSPUtil` by using the following page directive:

   ```
   <jsp:directive.page import="curam.util.client.jsp.JspUtil"/>
   ```

   b) Insert the scriptlet to print the session expired message on the page:

   ```
   <jsp:scriptlet>
     <![CDATA[ JspUtil. printSessionExpiredMessage(pageContext); ]]>
   </jsp:scriptlet>
   ```

2. To configure the custom logon renderer class, do the following steps:
   a) Create a `div` with a custom ID on your logon page to wrap the session expired message.
   b) Call the following method and pass in the ID of the `div` as a parameter:

   ```
   JspUtil.getSessionExpiredMessageScript(div.id);
   ```

5.5 Tab session 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.

5.6 Browser Specific Session Management

The version of the browser that is used can affect new sessions when they are started and when they are shared. Two browser instances that share the same session result in the same set of open tabs that are displayed in both instances. This sharing can cause interference and unpredictability of the persisted data in the same way if two users that use the same user ID and password from different computers.

Example Session Issue A user logs in to the Cúram application in one browser instance as the *′admin′* user. They then open a new browser tab, which shares the session. From here, they access the Cúram login page and login as a *′caseworker′* user.

In this situation, the original browser tab still displays the tabs for the admin user. If the user refreshes the original tab, then the tabs and application view are restored for the caseworker application. Alternatively, if the user opens new tabs that apply to the admin application only, the tabs are persisted for the caseworker user. Within the same browser session, a user must always log out to end the session and be able to log in as a different user.

Users logging into two separate applications (for example, internal and external applications) within the same browser also causes problems. Within one browser, users cannot log in to external and internal applications at the same time.

6 Browser management

You can configure specific behavior for the Cúram supported browsers, such as warning users if they are about to leave a page without saving data, or notifying them when they are not using an optimal browser version.

6.1 Configuring browser Back, Refresh, and Close button behavior

The standard Cúram application does not support using the browser **Back** and **Refresh** buttons to navigate the application. Also, if users click the **Close** button to close the application, they might lose data. In both the caseworker user interface and the classic Universal Access user interface, if users click either the **Back**, **Refresh**, or **Close** browser buttons, by default a warning message is displayed in a confirmation window. The warning message prompts users to either stay on the page or leave the page as requested. You can configure properties to either enable or disable the confirmation message from displaying on the internal application, the external application, or both. The warning dialog to prevent the loss of data is in the internal application.

Before you begin

You must log on to Cúram as a system administrative user.

About this task

Use the following procedure to either enable or disable confirmation messages from being displayed when users click either the **Back**, **Refresh**, or **Close** browser buttons in either the caseworker user interface or the Universal Access user interface.

The content of the confirmation message depends on the browser, and cannot be customized.

Note: Browser-specific behavior

- **All browsers**

  In all browsers, when a warning message confirmation window is displayed after you click the **Back**, **Refresh**, or **Close** buttons, the following actions are recommended:

  - It is recommended that users do not click the **Leave** button. Clicking the **Leave** button causes unpredictable results that depend on the browser that is being used, and on where users are within the application. Instead, it is recommended that users click the **Stay** button in the warning message confirmation window. To save any data that is entered on the page, users click **Stay on the Page**. To run the requested action, users click **Leave the Page**.
  - For the **Refresh** button, where users are asked to confirm whether they want to reload the page, it is recommended that users do not click the **Reload** button. Clicking the **Reload** button causes unpredictable results that depend on the browser that is being used, and on where users are within the application. Instead, it is recommended that users click the **Don't Reload** button in the warning message confirmation window.
  - If the user clicks the **Back** button before the page is loaded, the browser dialog is not displayed in the browser.
  - If a user does not interact with a page by clicking, touching, scrolling, or typing on the elements, the warning message is not displayed when the user clicks the **Back** button.

  Users can then use the supported navigational options that are provided in the application to do the actions that users require.
- **Chrome and Microsoft Edge**

  If you enable the confirmation message to be displayed, both Chrome and Microsoft Edge display an extra checkbox that users can select to stop the page from opening more message or confirmation windows. If users select the checkbox, the message or confirmation window is not displayed again. It is recommended that users do not select the checkbox.
- **Firefox**

  In Universal Access, if a user does not interact with a page by clicking, touching, scrolling, or typing on the elements, the warning message is not displayed when the user clicks the **Back** button. In this case, data is not lost if the user leaves the page.
- **Safari**
  In Universal Access, if the warning message confirmation dialog is displayed and the user clicks the **Leave** button, the feature is disabled in the Safari browser. The **Leave** button is enabled again only when the user closes the main current browser and then reopens the browser.
- **Tablets**
  The warning message confirmation dialog is not displayed on tablets.

Procedure

1. Click **System Configurations**.
2. In the Shortcuts pane, click **Application Data** > **Property Administration**.

3. To enable or disable the confirmation window in the Cúram user interface or the Universal Access user interface, search for and edit the value of the proceeding properties. By default, the warnings are enabled on both the internal and external application.
   - For the internal application, edit the value of the curam.internal.app.guard.against.leaving property. The property is used to indicate whether warning messages are enabled or disabled when the user leaves or refreshes the internal application.
   - For the external application, edit the value of the curam.app.guard.against.leaving property. The property is used to indicate whether warning messages are enabled or disabled when the user leaves or refreshes the external application.
4. To publish the property change, click **Publish**.

6.2 Optimal browser support

A number of browsers and a range of browser versions are supported for use with Cúram. Users can be notified when they are not using the optimal version of a supported web browser. You can configure the range of supported versions for a browser, the message that users see, and the frequency at which the message is displayed.

Note: Cúram external applications are public facing applications when `mode="external"` is set in the application configuration file (*.app). Health Care Reform is an example of this type of application.

The user's web browser is considered suboptimal if it is below the supported minimum version of the browser, or above the supported maximum version of the browser.

A message displays at the top of the banner, which can be dismissed. Once the optimal browser message is dismissed and if the browser is not updated, the message will be displayed again when a certain number of days have elapsed. This is assuming that the fully qualified URL to the application remains the same. An example of a fully qualified URL might be `https://myserver.ibm.com:9044/CitizenPortal/application.do`.The number of days that have elapsed before the next optimal browser check is configurable and by default it is sixty days in the future. The default optimal browser message links to a website that assists the user to take action and update their version of the web browser to an optimal one.

The optimal browser message essentially has three components as follows:

- **Warning icon**
  The warning icon gets the attention of the user that they should update their web browser.
- **Optimal browser message content**
  The message content that will be displayed to the user. It will consist of plain text and optionally a hyperlink which directs the user to a website where they can take action to update their web browser.
- **Optimal browser message exit icon**
  Allows the user to dismiss the optimal browser message.

Related reference

Application configuration properties on page 50

The *ApplicationConfiguration.properties* file defines the most important application configuration settings. You might want to change some of the settings that are relevant to the client application.

CDEJResources.properties on page 188
This properties file can be localized. For more information, see *Locales*. Images defined in this file can also be customized per locale.

Optimal browser support configuration

Use properties in the *ApplicationConfiguration.properties* file to enable or disable optimal browser support, configure the number of days before the next browser check, and set the for minimum and maximum browser versions.

- **optimal.browser.detection.enabled**
  Example: `optimal.browser.detection.enabled=true`. This is an application wide setting. It allows this feature to be enabled or disabled. Valid values for this property are; “true”, and “false”. The default value is “false”.
- **optimal.browser.next.check**
  Example: `optimal.browser.next.check=20`. This property configures the number of days that will elapse before the next check is done to determine if a user's web browser is at an optimal level.

  Note: This must an integer value. It is recommended to use a value between 1 and 60 (inclusive). The default value is set to 60.

  If this value is incorrectly configured it will be set to the default value. Additionally, an exception will be reported in the server logs when client side tracing is enabled. For more information about setting client side tracing, see Tracing server function calls on page 55. It should be noted that if this value is changed, it will not take effect until the optimal browser message is displayed again.

Properties for minimum and maximum browser versions

The following the properties define what constitutes an optimal browser. The default value for each of these properties is in line with that supported by Cúram for external applications.

Note: The value of these properties must be an integer or double value, otherwise a default value of “0” will be set and the optimal browser feature will not work as expected when using the an application in the associated web browser. An exception will be reported in the server logs if client side tracing is enabled.

- **chrome.min.version**
  Example: `chrome.min.version=0`. This property is used to configure the minimum supported version of the Chrome web browser. Any version below this is not considered an optimal Chrome browser when using an Cúram application. The default value is set to zero because there is no minimum supported version for Chrome.

- **chrome.max.version**
  Example: `chrome.max.version=0`. This property is used to configure the maximum supported version of the Chrome web browser. Any version above this is not considered an optimal Chrome browser when using an Cúram application. The default value is set to zero because there is no maximum supported version for Chrome.
- **ff.min.version**
  Example: `ff.min.version=0`. This property is used to configure the minimum supported version of the Firefox web browser. Any version below this is not considered an optimal Firefox browser when using an Cúram application. The default value is set to zero because there is no minimum supported version for Firefox.
- **ff.max.version**
  Example: `ff.max.version=0`. This property is used to configure the maximum supported version of the Firefox web browser. Any version above this is not considered an optimal Firefox browser when using an Cúram application. The default value is set to zero because there is no maximum supported version for Firefox.
- **safari.min.version**
  Example: `safari.min.version=0`. This property is used to configure the minimum supported version of the Safari web browser. Any version below this is not considered an optimal Safari browser when using an Cúram application.
- **safari.max.version**
  Example: `safari.max.version=0`. This property is used to configure the maximum supported version of the Safari web browser. Any version above this is not considered an optimal Safari browser when using an Cúram application.
- **edge.min.version**
  Example: `edge.min.version=0`. This property is used to configure the minimum supported version of the Microsoft™ Edge web browser. Any version below this is not considered an optimal Microsoft™ Edge browser when using a Cúram application.
- **edge.max.version**
  Example: `edge.max.version=0`. This property is used to configure the maximum supported version of the Microsoft™ Edge web browser. Any version below this is not considered an optimal Microsoft™ Edge browser when using an Cúram application.

Optimal browser message configuration

Use properties in the *`ApplicationConfiguration.properties`* file to configure the text in the optimal browser message.

- **optimal.browser.msg.description**
  Example: `optimal.browser.msg.description=optimal browser message banner`. This property configures the text for the description of the optimal browser support feature so that it can be read by the screen reader. A default value is provided.
- **optimal.browser.msg.text**
  Example: `optimal.browser.msg.text=For a better experience, please {0.link:http://www.whatbrowser.org/}update your browser{0.end}`. This property configures content of the optimal browser message. The text between the `{0.link:` and `{0.end}` mark-up tags configures the hyperlink and hyperlink text. These

mark-up tags are optional. If they are omitted from the value of this property then the optimal browser message will be displayed as plain text. If the mark-up tags are included but not specified correctly, i.e. the specified hyperlink (URL) is not in the correct format or the format of the markup tags themselves are not correct, then the optimal message content will not be displayed as expected.

- **optimal.browser.msg.info**
  Example: `optimal.browser.msg.info=Rendering..`. This property is used to configure the text while the optimal browser message is being rendered. A default value for this property is provided.
- **optimal.browser.dismiss**
  Example: `optimal.browser.dismiss=dismiss`. This property is used to configure the tooltip text associated with the button to dismiss the optimal browser message. A default value for this property is provided.
- **optimal.browser.warning**
  Example: `optimal.browser.warning=warning`. This property is used to configure the text for the warning icon so that it can be read by the screen reader. A default value for this property is provided.

7 Autorecovery

Autorecovery protects users from losing in-progress work due to a system interruption. Form data that users had entered in modals is restored in certain situations. Specifically, the browser tab being refreshed, the tab being closed and reopened or, after login, where the user's session previously timed out while a modal window was open.

When caseworkers log back in to the application after an interruption, autorecovery restores the user's Cúram tab session and the previously open modal window with the data that was last entered. This means that no data is lost for supported fields, and users can continue from where they stopped before the interruption.

The autorecovered record is only viewable to the user who was creating it at the time of the interruption. When users log back in after a system interruption, users are presented with recovered data only one time. If they cancel or exit the page, the data is lost, and can no longer be recovered.

The autorecovery process works alongside session management. Where a system interruption occurs, session management displays the same tabs that were open before the interruption. Autorecovery complements session management by restoring previously entered data within modals that were being worked on at the time of the interruption.

Basic autorecovery scenario

The following example outlines a basic scenario for autorecovery.

A caseworker has one browser tab open that contains Cúram. They are creating a new note for client James Smith when they experience a timeout. When they log back in, they are immediately returned to the New Note modal that displays the data they were entering for James Smith before the interruption. The caseworker can continue working on the note without losing any data.

7.1 Technical overview

Autorecovery works alongside session management so that the user is returned to where they were before a system interruption. Data the user previously entered is restored.

Autorecovery

When the user is in an area of the application that is supported by autorecovery, data that is entered in a supported field is automatically stored when the user changes or enters a value and leaves the field. By default, autorecovery is not enabled. For autorecovery to work, you must first enable it within system administration.

Where autorecovery is enabled and the user is in a supported area of the application, the system recovers data that is entered on modal pages when a system interruption occurs. Autorecovery configuration is global, that is the whole application and portals are enabled for autorecovery.

An autorecovery record is created that contains the modal the user is on when the system interruption occurs and, if applicable, the current data. When the data is posted to the server, the data is stored in the client cache and is persisted to the database when the session times out. Autorecovery supports only one record per user, which represents the last modal the user interacted with.

When the modal is closed through the user saving or cancelling, the user's autorecovery record is deleted.

Autorecovery and session management

Autorecovery works alongside session management. While a session is still active, the recovery is managed within the session state. For example, the user can close the browser tab and re-open it, and the user is then returned to where they were. If the user's session times out, the recovery is managed by storing the data to the database when the timeout occurs and is restored from the database when the user logs back in. For more information about tab restoration and session management, see 5 Session management on page 191.

Throttle strategy

If uncontrolled, autorecovery could overload the server with requests. For this reason, the number of requests that can be sent by the same user is limited by time. This is referred to as throttling. The default setting is 500 milliseconds. Each autorecovery save request sends all fields in the form, so only the last generated request is sent when the throttle time has elapsed. Throttling is most relevant when using the rich text field where a user's progress is saved as the user is typing instead of waiting for the user to exit the field.

7.2 Areas supported by autorecovery

Autorecovery supports a variety of modals and form elements across Cúram.

User interface metadata (UIM) modals

Autorecovery of form data is supported in user interface metadata (UIM) modal dialogs only. The following list outlines the areas of the application where autorecovery support is provided:

- Standard, dynamic, and generated UIM modals
- Wizards (for example, Create Contact Log)
- Portals for the multidisciplinary team (MDT)
- The Cúram application for all users, including caseworker, administrator, system administrator, and Cúram Provider Management.

Note: Modals that are launched from other modals (for example, searching for a person modal when a user is adding participants to a contact log) are not supported by autorecovery.

Note: If the user is on a modal with no fields that are supported by autorecovery, there is no autorecovery for that modal.

Form elements

Autorecovery restores data for UIM form input fields that are supported by autorecovery. The following list outlines the form elements that are supported by autorecovery:

- Text input fields
- Rich Text Editor
- Most drop-down lists (for exceptions, see the list of form elements that are not supported by autorecovery)
- Date pickers and date fields
- Date time fields
- Time and duration in the format hours:minutes
- Single checkboxes
- Address widgets
- Comments
- Text areas

The following list outlines the two general strategies for when autorecovery stores the data:

- When the user exits a field (when focus is changed).
- As the user is typing. By default, this is every 500 milliseconds as defined by the throttling configuration. For more information about throttling, see Technical overview.

Note: The following list outlines the form elements that are not supported by autorecovery:

- File editing
- File uploading
- Password fields
- Radio buttons
- Frequency Pattern Selector
- Code table hierarchies
- Scheduling appointments
- Search fields with a search that takes the user to another page (for example, to search for a user or person)
- Certain selection lists:
  - Drop-down lists that are populated from server interface properties (not populated from code tables). For example, case participant evidence.
  - A single select list box where there is no drop-down chevron. A subset of the values is initially displayed and there is a scrollbar.
  - A multiple select list box where users can select more than one value using a scrollbar. For example, the purpose field on a contact log.
  - A single list of checkboxes where users can scroll and select a checkbox.
  - A multiple list of checkboxes where users can select more than one value.
  - A transfer list widget. For example, transfer evidence where users can transfer from one text field to another, such as in the Cúram Child Welfare application.

7.3 Supported scenarios

Autorecovery supports system interruptions due to session timeouts, the browser tab being refreshed, or the browser tab being closed and reopened.

If the session timed out while the browser window was open, the user can log back in using any supported browser. For example, if the user inadvertently closes the browser tab or window, reopening the browser tab or window restores the data if the session is still active.

Note: Autorecovery is not supported in scenarios where there is a loss of network connectivity and a timeout then occurs before the connection is re-established for browsers. For example, the system won't restore the modal a user had open in a scenario where the user times out while the modal is open but the network connection has been lost.

Autorecovery is also not supported for virtual private networks (VPNs) where there is a loss of connectivity and a timeout occurs before the VPN connection is re-established, or where the computer crashes or goes to sleep and a timeout occurs before it can be re-started.

The concurrent use of multiple devices by one user at the same time or users sharing logins is not supported.

While a session is still active, the recovery is managed within the session state. For example, the user can close the browser tab, and re-open it and the user is then returned to where they were. If the user’s session times out, the recovery is managed by storing the data to the database when the timeout occurs and is restored from the database when the user logs back in.

A session can be terminated unexpectedly, for example because the network connection is lost and a timeout then occurs before the connection is re-established. Autorecovery cannot access the data as it was not stored to the database when the timeout occurred due to the loss of connection prior to the timeout.

For more information about tab restoration and session management, see Session management and Tab Restoration.

Multiple browser tabs or windows

Autorecovery also supports having more than one Cúram session open in the same browser in different browser tabs or browser windows. For example, where a caseworker is logged in to Cúram using Google Chrome and has two browser tabs open, one where the user is writing a note and the other where the caseworker is viewing the person home page so they can look up information about the client. As all browser tabs and browser windows share the same browser session, when the user logs back in after an interruption, all open tabs display the same information. If more than one modal was open in different tabs or windows at the time of the interruption, the modal that is recovered is the one the user last interacted with. If more than one background page was open in different tabs or windows at the time of the interruption, the page that is recovered is the background page the user last navigated to.

Note: Autorecovery does not support the concurrent use of multiple different browsers or instances of the same browser, for example incognito sessions. Results may be unpredictable.

7.4 Enabling autorecovery

You can configure specific behavior related to autorecovery for optimal use.

For more information about configuring autorecovery, see the *System Administration Guide*. For information about session management properties, see Session Configuration.

Table 51: Autorecovery configuration steps

Step	**Reason**
Enable `curam.sessionmanagement.tabsession.autorecovery.enabled`	Enable the property so that the user's data can be autorecovered after a system interruption. The setting applies to all the application areas where autorecovery is supported. If the value is set to true, the system recovers any unsaved data that caseworkers entered before the interruption occurred and returns the caseworker to where they were in the application before the interruption. If the value is set to false, the system does not recover unsaved data in any part of the application. The property is dependent on theCuramFormsAPI.
Enable `curam.sessionmanagement.tabsession.autorecovery.enabled`	The CuramFormsAPI provides the infrastructure that permits autorecovery to detect changes to forms and to set form data. If it is not enabled, CuramFormsAPI won't communicate form changes to autorecovery and it won't respond to requests to set data. The CuramFormsAPI must be enabled for autorecovery to work.
Enable `curam.sessionmanagement.tabsession.autorecovery.throttle.interval`	The length of time in milliseconds to be applied between autorecovery post requests to the server. The autorecovery throttle interval property setting applies to all areas of the application where autorecovery is supported. A value of 0 means that throttling is disabled.
Set `tabSessionUpdateCountThreshold`	For optimal use, set `tabSessionUpdateCountThreshold` to 1. A value of 1 initiates writes on every update and ensures the optimal use of autorecovery.
Write a custom batch process as needed	The autorecovered data is kept in the autorecovery record until the user logs back in. Customer organizations, based on their view of the governance of the data, must consider whether they want to customize locally to remove the data at some point.

Production usage of autorecovery

The following list outlines the steps to perform when you are using autorecovery in production:

- For optimum use, enable autorecovery and the other settings during system downtime and not at runtime. For autorecovery to take effect, a redeployment or server restart is required.
- In some situations, an organization might decide that it no longer wants to use autorecovery. If necessary, the autorecovery property can be disabled after it was turned on. For optimum use, disable autorecovery during system downtime. A server restart is required for the change to take effect.
- Inform users that when they clear their browser cache, they are also clearing their autorecovery data. As autorecovery uses the browser's client cache to temporarily store the in-process data, avoid clearing the cache. When a user clears their session on their browser, for example by clearing their browser's cache, they must log back in. This creates a new session. The new session can then be autorecovered through the standard autorecovery processes.

Note: Before you upgrade to a new version of Cúram, save any outstanding work to ensure that unsaved data is not lost.

8 Domain-specific controls

Domain-specific controls enable a more sophisticated interface for user information than the standard set of HTML controls. Examples of domains that require sophisticated controls include dates, date-times, the meeting view, and the rules decision tree.

A web page that is generated for UIM pages that contains a server access bean with fields of this nature that contains a custom control appropriate to the type. For example, when a server bean contains the *`CALENDAR_XML_STRING`* domain, a calendar is generated that expects server information in a particular XML format.

8.1 Dates

Dates are mapped to the *`SVR_DATE`* domain. Any server access bean that contains fields of this type shows a date selector to the user for data input. These selectors are HTML fields with an adjacent pop-up icon that causes a menu to be displayed allowing the user to select a date or date time with ease.

Note: This function is based on JavaScript and it is important that the user enable JavaScript 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. For more information, see CSS on page 62.

The initially configured date 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 selection populates the date field.

The date format string that is 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
```

If this value is not set, the date format string will default to the date format setting that is specified in the *`ApplicationConfiguration.properties`* file.

Three Field Date Selector

Dates can be mapped to the `THREE_FIELD_DATE` domain to enable use of an alternative date selector widget. Server access beans that contain fields of this type will display three drop-down elements to the user for data input.

The order of the drop-down elements and the display values of the month element reflect the date format, as configured by the `dateformat` property in the *`ApplicationConfiguration.properties`* file. The day drop-down is populated with numbers that range 1 - 31. Validation at the infrastructure level prevents users from selecting an invalid date, for example, February 31, 2015. The year drop-down element is populated with values that start 100 years in the past to 30 years in the future. The range and order of the options are not configurable.

A selection from the drop-down elements is made either by scrolling to the wanted value or by typing the value when the drop-down element is active.

To use the **Three Field Date Selector** widget, model a property on a struct to use a data type derived from the `THREE_FIELD_DATE` domain.

8.2 Date-Times

Date-times are mapped to the `SVR_DATETIME` domain. Any server access bean that contains fields of this type will display a date selector (as described in the **Dates** topic) next to a time entry field.

Similar to the date selector, the pop-up here requires JavaScript to function correctly. An extra control exists 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.

Note: The user needs to enable JavaScript in their browser for these selectors to work.

The date input field will not be displayed when the `CURAM_TIME` domain (a descendant of the `SVR_DATETIME` domain) is used,

The date time format string that is 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
```

Figure 5: Customizing the Date-Time format

If this value is not set, the date time format string will default to `HH mm ss`.

Related reference

Dates on page 215

Dates are mapped to the *SVR_DATE* domain. Any server access bean that contains fields of this type shows a date selector to the user for data input. These selectors are HTML fields with an adjacent pop-up icon that causes a menu to be displayed allowing the user to select a date or date time with ease.

Representing Time-Only Values

Cúram has a base type for **date-only** and **date-time** values. No specific base type exists for **time-only** values.

A `CURAM_TIME` domain is provided in the initial configuration of Cúram and this configuration is used by the client infrastructure to display a corresponding time-only widget. The widget also initiates 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 it is working with time-only domains the corresponding server-side code must ignore completely the date part of the value.

Because time-only domains are based on the `SVR_DATETIME` domain, the default values also will 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 that is rendered for `CURAM_TIME` domain is an editable combination box as the following example shows. The time input field contains selectable time values for every 30 minutes. The exact time value also can 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 ends to follow the same format.

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

For more information, see Application configuration properties on page 50.

8.3 Frequency Pattern Selector

In the frequency pattern selector pop-up, users can configure a frequency pattern, such as daily, weekly, monthly, bi-monthly, or yearly. 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.

The 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. For more information, see CSS on page 62.

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.

8.4 Selection lists

The use of the standard HTML selection list, the `select` element, is supported. Selection lists truncate long data strings to preserve the correct page layout. The full value of the data string is available as a tooltip for each item in the list. The list can be populated with data in a number of ways.

Adding an empty entry to a list for non-mandatory fields

By default, browsers select the first item in a selection list if no item is marked as selected. In certain cases you might not want to suggest a value to the user and 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.

Drop-down, scrollable and check-boxed list types

Drop-down and scrollable lists

A 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. On 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. 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 dependent on the `HEIGHT`value 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.

Check-boxed lists

A check-boxed 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.

Enabling multiple selection in lists

You can enable multiple items to be selected in scrollable lists, but not in drop-down lists.

To enable this add the following items to the *`curam-config.xml`* file.

```
<MULTIPLE_SELECT>
  <DOMAIN NAME="MY_DOMAIN" MULTIPLE="true"/>
</MULTIPLE_SELECT>
```

For each domain that 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 are enabled. When the form containing the selection list is submitted, the selected values are 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 the values should match some of those values specified by `HIDDEN_PROPERTY`.

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 displays all code-table entries that are marked as “enabled”. The entries are sorted alphabetically according to their code descriptions.

You can override this behavior by setting the “sort order” of each entry. See the *Server Developer's 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 more than one entries are initially displayed. Set the HEIGHT attribute of the FIELD element to a value greater than 1.

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.

An example selection list on an insert page is shown:

```
<FIELD LABEL="Field.Label">
  <CONNECT>
    <INITIAL NAME="DISPLAY" PROPERTY="personName"
             HIDDEN_PROPERTY="personID"/>
  </CONNECT>
  <CONNECT>
    <TARGET NAME="ACTION" PROPERTY="personID"/>
  </CONNECT>
</FIELD>
```

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>
```

Transfer List widget

The Transfer List widget is a control to facilitate multiple selections for a user, which you can use as an alternative to a regular list with multiple selection.

The Transfer List widget consists of two HTML select controls placed side by side.

- The left control contains the items from which selections can be made, see Drop-down, scrollable and check-boxed list types on page 218.
- The right control displays already selected items.

Four buttons between the lists allow for selecting or deselecting individual list items or all list items, transferring them from one list to another and back as required.

Transfer List configuration

A Transfer List widget is displayed instead of a regular HTML multiple selection control when configured in one of the following two ways:

- To display all multiple selection controls in an application as Transfer List widgets, set the `TRANSFER_LISTS_MODE` element value to `true` in the *`curam-config.xml`* file.
- To display individual selection controls in an application as Transfer List widgets, set the `CONTROL` attribute on the appropriate UIM `FIELD` element to be `TRANSFER_LIST`. This setting is applicable only for fields that are rendered as multiple selection controls on the resulting UIM page and is 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.

8.5 User Preferences Editor

The User Preferences Editor allows a user to edit a user preference value for use anywhere within the application.

For information about defining user preferences, see the *Server Developer's Guide*.

The editor may be accessed from the taskbar by clicking the preferences button. On clicking this button a popup window displays a list of all visible user preferences. Those preferences that are editable are shown as text fields, radio buttons or drop-down menus, depending on the type.

Users can 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 by using the editor will be applied immediately.

User can click `Reset to Default` to return the values to those that were originally defined.

8.6 Rules Trees

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 that displays all rules.

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 Rules Editor on page 226.

You can use the `CONTROL` attribute of the `FIELD` element to change the format of the rules display. You can use the `CONFIG` attribute of the `FIELD` element 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 than rule that succeed.

Default Rules View

The default rules view of the rules tree, 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.

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 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, Failed Rules View on page 222, 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 Dynamic Rules View on page 222.

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 Summary Rules View on page 221

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 Dynamic Rules View on page 222

Dynamic Rules View

When the `CONTROL` attribute is set to `DYNAMIC`, an expanding or contracting version of the decision is 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 Behavior of Summary and Highlight-On-Failure Indicator on page 224 for more details on the summary flag.

The dynamic view provides users 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.

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 4.7 JavaScript externalized strings on page 183. 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 CSS on page 62.

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>
```

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) 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>
```

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>
  <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>
```

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 it's 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.

Dynamic Full Tree Rules View

When the `CONTROL` attribute is set to `DYNAMIC_FULL_TREE`, a view is displayed.

The functionality of the `DYNAMIC_FULL_TREE` view is similar to the dynamic rules view. For more information about the functionality of the dynamic rules view, see the *Dynamic Rules View* related link. The main difference between the views is that for the `DYNAMIC_FULL_TREE` view the entire rule set is displayed. While similar to the default rules view, the tree is interactive. There is no filtering of the display of rule groups in the `DYNAMIC_FULL_TREE` view, which potentially makes it difficult to understand for a user who is not familiar with the rules engine. To configure the view, use the `RulesDecisionConfig.xml` file. For more information, see the *Dynamic Rules View* related link.

Related reference

Dynamic Rules View on page 222
When the `CONTROL` attribute is set to `DYNAMIC`, an expanding or contracting version of the decision is displayed instead of a static tree.

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.

For more information, see Default Rules View on page 221 and Dynamic Rules View on page 222.

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>
```

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>
```

8.7 Meeting View

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. The control has two modes of operation, single and multiple selection.

Single selection mode

In the single selection mode meeting view, the first column contains 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, if

John Smith is busy from 10:30 until 12:30, it is not 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.

Note that any parameters passed to a page containing the meeting view will be included in any links within the view.

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.

Meeting View XML format

Configuration settings for the meeting view must be in a file called *`MeetingViewConfig.xml`* in a component. The meeting view control expects information in a specific XML format.

An example Meeting View XML format is shown.

```
<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>
```

The `MODE` attribute is either `Single` or `Multiple`.

The `DURATION` attribute is in minutes.

The `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”.

The `TYPE` attribute associates the schedule information with configuration settings which are also specified in an XML format as shown:

```
<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 with valid values of `15`, `30`, or `60` only. The `START` and `END` attributes detail the beginning and end times of the time line 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.

8.8 Charts

Charts are displayed when one of either the CHART_XML or BARCHART_XML domains (or any derivation of them) is used as the source of a field.

Chart appearance

A bar chart displays a number of rows horizontally with a horizontal and vertical axis. 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.

The chart scale is chosen to fit the biggest stack of bars, which you can override 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, 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. Both bar links and caption links are configurable, as described in Chart configuration on page 231.

Textual captions 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.

Textual captions are truncated to better maintain the scale and readability of the chart. Users can click the enlarge button to see a larger version and to read any truncated labels.

A column 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.

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.

Data tips 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 that is attributed to the element and the element stack. The data tip also shows the category or group to which that element belongs, and the type of the element, which corresponds to an entry in the legend if a legend is included.

Chart configuration

You can configure various aspects of charts by setting the `CONFIG` attribute on the appropriate UIM fields. 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 must 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).

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.

Table 52: Attributes of the CONFIG element

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.

Attribute	Description
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 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 *BarChart.properties*. This text is used as the label for the x-axis in a column chart, or y-axis in the bar chart.
Y_AXIS_LABEL	Key to a text property in *BarChart.properties*. This text is used as the label for the y-axis in a column chart, or x-axis in the bar 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 MIN_HEIGHT 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 MAX_HEIGHT 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 MAX_HEIGHT 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 MAX_HEIGHT 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 *BarChart.properties*. This text is used as the label for the whole chart.

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.

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. 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 application'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 tool-tips is customizable using the key total.tooltip.text in the *`BarChart.properties`* file.

Collapsible Cluster Support Collapsible clusters are not supported for any cluster containing this widget.

Customizing chart colors in system administration

Complete the following steps to modify the colors on the Assessment Delivery Results Chart, the Assessment Delivery Details Chart, the Assessment Tracking Chart, or the Factor Ratings Line Chart.

About this task

- The default colors for the Assessment Delivery Results Chart, the Assessment Delivery Details Chart, and the Assessment Tracking Chart, which are defined by the curam.assessmentplanning.graphRGBColors application property, are 6929C4, 1192E8, 005D5D, 9F1853, FA4D56, 520408, and 198038.
- The default color for the Factor Ratings Line Chart, which is defined by the curam.outcomeplanning.factorGraphRGBColor application property, is B28600.

Procedure

1. Log in to Cúram as a system administrator.

2. Select **System Configurations** > **Shortcuts** > **Application Data** > **Property Administration**.
3. Enter the application property in the **Name** field and click **Search**.
4. Select **...** > **Edit Value**.
5. Update the values and click **Save** to save your changes.
6. Click **Publish** for your changes to take effect.

Customizing colors on horizontal and vertical bar charts

Complete the following steps to modify the colors on horizontal and vertical bar charts.

Before you begin

The default colors were defined in a specific sequence to meet contrast ratio guidelines. Ensure that any changes you make also meet contrast ratio guidelines.

Procedure

1. In your Cúram application development environment, edit the *`%CURAM_DIR%\CuramCDEJ\lib\curam\xml\xsl\chart\charts.xsl`* file.

   Where `%CURAM_DIR%` is the installation directory, by default `C:\Merative\Curam\Development`
2. Update the colors, which are defined in the `<ibm:colors>` XML tag.

```
<ibm:colors>
 <ibm:color>6929c4</ibm:color>
 <ibm:color>1192E8</ibm:color>
 <ibm:color>005D5D</ibm:color>
 <ibm:color>9F1853</ibm:color>
 <ibm:color>FA4D56</ibm:color>
 <ibm:color>520408</ibm:color>
 <ibm:color>198038</ibm:color>
 </ibm:colors>
```

3. Run a Cúram client build to pick up the changes.

Customizing colors on the Participation Summary chart

Complete the following steps to modify the default background and hover background colors for Scheduled Hours and Actual Hours on the Participation Summary chart in the Cúram.

About this task

The default background and hover background colors are:

Scheduled Hours	Actual Hours
6929C4	1192E8

Procedure

1. In your Cúram application development environment, edit the *`<install>/webclient/components/CAAssessmentTracking/javasource/caassessmenttracking/ParticipationSummaryResultRenderer.java`* file.

   Where `<install>` is the installation directory, by default:

- `C:\Merative\Curam\Development` for Microsoft™ Windows™.
- `/opt/Merative ™/Curam/Development/` for Linux®.

2. Update the color values in this line of code.

```
final string jsonColorString = "{\"schColor\":\"#6929c4\",\"actColor\":
\"#1192e8\"}";
```

3. Run a Cúram client build to pick up the changes.

Chart data formats

The data to be displayed in a chart comes from the server in XML format.

An example of the XML used to create a chart is shown.

```
<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>
```

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, or column. 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. 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.

8.9 Heatmap Widget

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>
```

Here, the `REGION` elements specify the importance level ("heat") of their contained `ITEM` s. 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.

Configuration

Different types of heatmap can be configured by creating entries in the *`HeatmapConfig.xml`* file in your *`components`* directory.

An example of the format is shown.

```
<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:

Table 53: Attributes for `CONFIG` element

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.
`LEGEND_POSITION`	By default, the Heatmap legend is drawn to the right 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 4 Localization on page 177 for more details on localization).

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.

8.10 Workflow

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. 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.

In a 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 represented 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.

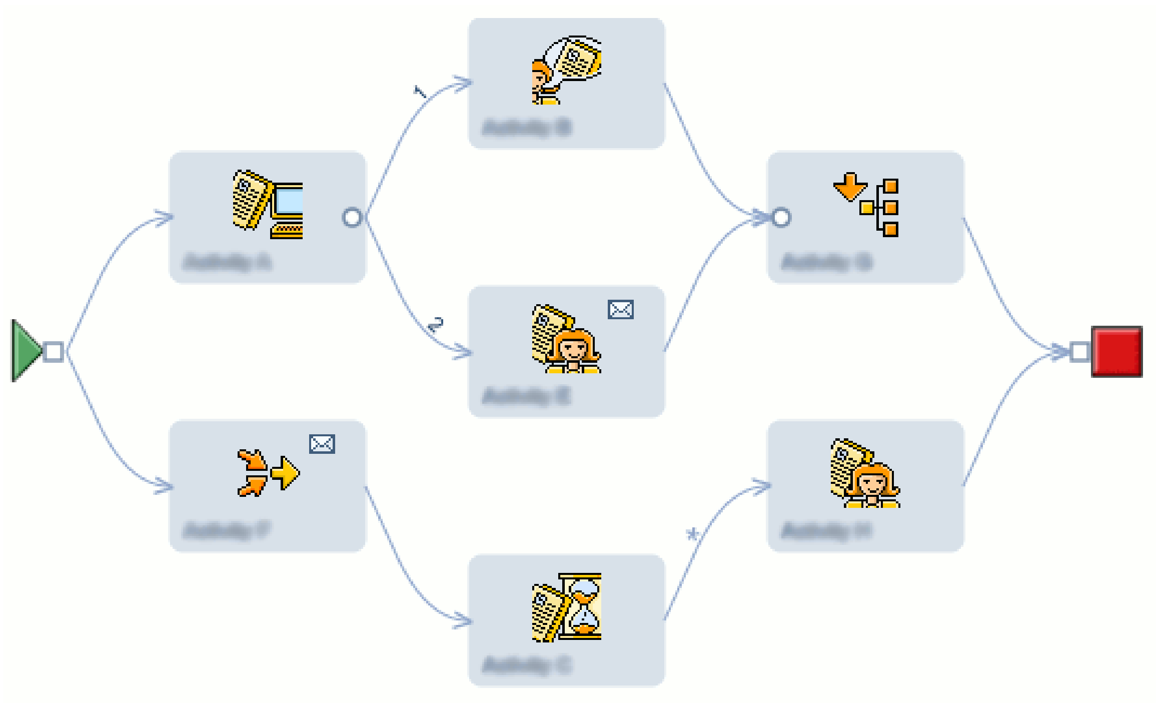


Figure 6: Workflow

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.

An an example of workflow XML data is shown.

```
<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:

Table 54: Attributes of a Node

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.

Attribute	Description
`END-DATE-TIME`	The end date time of an activity instance or transition 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-RESERVED-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.

The `EDGE` element represents a single transition in the workflow. All attributes of an edge are defined in the following table:

Table 55: Attributes of an Edge

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 `true` 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 enabled as a hyperlink. This attribute is `false` by default.
`ORDER`	Indicates the order of an edge relative to other edges.

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:

Table 56: Attributes of Workflow 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 `ACTIVITY-TYPE-CODE` attribute values.
`TASKSTATUS_CODETABLE`	Codetable name for resolving `TASK-STATUS` attribute values.
`PROCESSSTATUS_CODETABLE`	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 `true` and `false`.
`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.
`HIGHLIGHT_ACTIVITY_PARAM`	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.

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.

8.11 Evidence view

The Evidence view has two modes for displaying and comparing evidence data, evidence display mode and evidence comparison mode.

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 its domain.

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 column to show previous values against current values) column.

Evidence view configuration

Configure the evidence view by changing settings in appropriate properties files. Use *`DisplayEvidence.properties`* for Evidence Display mode, and *`ComparedEvidence.properties`* for Evidence Comparison mode. You must create these properties files 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
```

Evidence view XML data formats

The Evidence View expects specific XML formats for the Evidence Comparison and Evidence Display modes.

An XML format example for Evidence Comparison mode is shown.

```
<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 XML format is an example for 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.

8.12 Calendar

The calendar is used by any UIM page that 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.

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 for Calendar:

```
<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>
```

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 attributes allowed on the `EVENT` and the `SINGLE_DAY_EVENT` elements respectively.

Table 57: EVENT attributes in schema

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 ActivityTimeStatus code table in the reference application.	No

Attribute Name	Description	Required
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 58: SINGLE_DAY_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

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.

Table 59: Calendar View Type Values

Code	Value
CVT1	Day view
CVT2	Week view

Code	Value
CVT3	Month view

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. An example of the structure of the *CalendarConfig.xml* file is shown.

```
<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>
```

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.

Table 60: Parameters Passed to Event Description Pages

Parameter Name	Description
ID	the event ID
RE	Recurrence indicator
AT	Attendee indicator
RO	Read-only indicator
LV_	Activity level
AC	Acceptance indicator

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.

8.13 Payment Statement view

The payment statement view is used for displaying under or over payment within the Cúram application framework.

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. A sample *PaymentStatement.properties* file is shown.

```
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
```

8.14 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.

8.15 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. Except for the **STATE** field, each of the fields are, by default, text input fields. The **STATE** field is a drop-down.

To parse the address and display the address, the elements the address consists of must be defined in the *curam-config.xml* file. Different address configurations for different locales in the

Cúram application can be defined. The following address configuration demonstrates how to set the configuration by 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" DESCRIPTION="Address.Description">
    <ADDRESS_ELEMENT LABEL="Address.Label.AptSuite"
                     NAME="ADD1"
                     CONDITIONAL_MANDATORY="true"/>
    <ADDRESS_ELEMENT LABEL="Address.Label.Street.1"
                     NAME="ADD2"
                     CONDITIONAL_MANDATORY="true"/>
    <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>
```

The ADDRESS_CONFIG element builds by using multiple LOCALE_MAPPING and ADDRESS_FORMAT elements. In Cúram application deployments with multiple locales, you might want to use a different address format for each locale. To use a different address format for each locale, use the LOCALE_MAPPING element. The element contains a LOCALE attribute that defines the locale and an ADDRESS_FORMAT_NAME attribute that defines the ADDRESS_FORMAT element to be mapped. By default, the Cúram application includes defined ADDRESS_FORMAT elements that are mapped to specific locales. As the locales are already mapped, it is not required to define LOCALE_MAPPING elements for the locales. However, customers can modify the elements or create new configurations, depending on a customer's implementation. The previous configuration example illustrates how the LOCALE_MAPPING element is used for the US and UK address formats The following table lists the default address formats and the corresponding locale mappings.

Table 61: Address format configurations

Address Format Name	Locale Mapping
US	en_US

Address Format Name	Locale Mapping
UK	en_GB
DE	de
CA	en_CA
KR	ko
JP	ja
TW	zh_TW
CN	zh_CN

When an address is created, the ADDRESS_FORMAT includes an optional COUNTRY_CODE attribute that is used in the address header. If the attribute is not set, the COUNTRY_CODE defaults to GBR when the address format specified is UK. When any other address format is specified, the COUNTRY_CODE is set to US. The COUNTRY_CODE is not used by the infrastructure. It is one of the fields in the address string that the application uses, but the infrastructure provides an initial value for it. The ADDRESS_FORMAT includes another optional DESCRIPTION attribute. The attribute refers to the property that is in the *CDEJResources.properties* file. The string provides information text that is displayed above the address fields.

The ADDRESS_FORMAT elements contain ADDRESS_ELEMENT elements that define the fields in the address tag. The ADDRESS_ELEMENT element includes a LABEL attribute that refers to properties that are contained in the *CDEJResources.properties* file. The address builds by using ADDRESS_ELEMENT tags. The tags must have a name and label. A code table can also be specified for each ADDRESS_ELEMENT. When a code table is specified, a dropdown displays the code table entries and the default code is selected.

The optional MANDATORY attribute specifies whether an address element must be completed.

Note: The MANDATORY settings in *curam-config.xml* require that the field that provides the address data must be marked as mandatory the application model.

The optional CONDITIONAL_MANDATORY attribute specifies whether an address element is partially mandatory. For the US address format, as a minimum the first two address fields must be competed in a form. When the CONDITIONAL_MANDATORY attribute is set on an ADDRESS_ELEMENT, an aria-label is set on the field with the description text. Screen readers use the description text for visually impaired users.

Note: To provide the aria-label with the description text, the DESCRIPTION attribute must be set on the ADDRESS_FORMAT.

8.16 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 set up of this particular view involves a particular setup in a page's UIM file. An example of the UIM for a schedule field is shown.

```
<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>
```

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

16777212 is the integer value which translates to the bit field 00000000111111111111111111111100. 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.

8.17 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.

8.18 Pop-up pages

Use this information to set up a pop-up page or a multiple pop-up page as needed. The Cúram application has a number of built-in pop-up pages, such as the Date Selector pop-up, which are “helpers” that are used to enter data. Developers can also 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 person's 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 default pop-up widget has a grey input field with a search, - in the form of a magnifying glass, and a clear icon beside it. When the user clicks on the search icon it activates a pop-up page. The user can select an item from the pop-page which populates the text input field on the pop-up widget.

Using multiple pop-up search pages for a single field

In some cases, you might 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. The resulting pop-up widget has an additional drop-down field rendered to the left of the text input field. 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.

Configure a pop-up page

The first step is to configure a 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>
```

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.

Table 62: Attributes of the `POPUP_PAGE` element.

Name	Description
`PAGE_ID`	Specifies the UIM page id of the pop-up page to open to search for an existing item.
`CREATE_PAGE_ID`	Specifies the UIM page id of the pop-up page to open to create a new item.
`CONTROL_TYPE`	Specifies the type of control where the value returned from the pop-up will be written to. The default value is `textunderline` which displays static text with an underline. To display a text input field set the value to `textinput`. When a text input control is configured, on the UIM `FIELD` which uses a pop-up, the `HEIGHT` attribute can be used to change from a single line text input to a multi-line text area.
`CONTROL_EDITABLE`	This attribute is only valid when `CONTROL_TYPE` is set to `textinput`. It controls whether the text input field is editable or not. Set to `true` to create a editable field and `false` to create a non-editable field.
`CONTROL_INSERT_MODE`	This attribute is only valid when `CONTROL_TYPE` is set to `textinput`. It allows you to configure how a value selected from a pop-up is inserted into the associated input control. The default is `overwrite` which means the selected value will overwrite the previous contents. Setting the attribute to `insert` means the selected value will be inserted at the current cursor position. Setting the attribute to `append` means the selected value will be appended to the previous contents of the input control.

Table 63: Child elements 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_HOVER	Location of image that is displayed when a user hovers over the search pop-up icon. Set the IMAGE_HOVER element if the IMAGE element has been set to a location other than the default location. If the IMAGE_HOVER element is not set, then a default image is displayed when a user hovers over a search pop-up icon.
IMAGE_PROPERTY	Optional key in the *CDEJResources.properties* 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 *CDEJResources.properties* file under which the locale-specific location of the pop-up launcher image otherwise specified by HIGH_CONTRAST_IMAGE attribute is stored. If the 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 *CDEJResources.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 *CDEJResources.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.

Name	Description
`CREATE_LABEL_PROPERTY`	Optional key in the *`CDEJResources.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.

Create a 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 that 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 persons 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 that 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$personID" />
    <TARGET NAME="PAGE" PROPERTY="value" />
  </CONNECT>
  <CONNECT>
    <SOURCE NAME="ACTION"
            PROPERTY="dtls$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.

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.

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>
```

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 that standard fields on a modify page because there are actually two source values to 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 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>
```

When invoking a pop-up, you can also supply page parameters to the pop-up. This is a slight variation on the previous two examples and involves the use of the `LINK` element. The following example shows two parameters that are 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` are to 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>
```

Configure a multiple pop-up page

A a multiple pop-up page can be configured through the MULTIPLE_POPUP_DOMAINS element in *curam-config.xml*.

An example of multiple pop-up domains is shown.

```
<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>
```

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, 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 the pop-up page that is opened. This link requires the CT_CODE child element of the POPUP_PAGE element to be set to the code-table code value.

Using a multiple pop-up page

When the configuration is done, the final step is to write the UIM that necessary to display the pop-up search.

An example of the UIM to use multiple pop-up windows is shown.

```
<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>
```

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 to be returned from the pop-up. The container must not include any other FIELD elements.

- The first field must be based on a code-table domain that contains a list of codes that correspond to the `CT_CODE` element.
- The second field must have a target connection that is based on a domain using the `MULTIPLE_POPUP_DOMAIN` element.

8.19 Agenda Player

The Agenda Player, or player, is a wizard-like control which provides guided navigation through a specified set of screens. The screens in the Agenda Player are generally part of an agenda or scenario, typically involving the step-by-step collection of information.

Note: Agenda Player widget is not supported outside the modal dialog context. An attempt to open it in the tab content panel or elsewhere, such as in the inline page of an expandable list results in an error message.

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.
- 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.

  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.

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 progress 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 Agenda Player configuration on page 265

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).

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 - 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 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.

Agenda Player configuration

Configure the Agenda Player 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.

Table 64: Attributes of the PLAYER element

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 Agenda Player customization on page 265.
`MODE`	This attribute allows for specifying Agenda Player navigation mode. It might have values of `basic`, `incremental` or `full`, `incremental` being the default one, used if the attribute is skipped in an configuration.
`NAVIGATOR-HIDDEN`	When this attribute is specified and set to `true`, 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 **Cancel** button. When present and set to `true`, 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.

Agenda Player customization

The Agenda Player comes with support for customization and 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>/components/<component_name>/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*.

Agenda 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"/>
          <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.

Table 65: Attributes of the page element

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 **Back** button should be disabled on this page.
`disableforward`	Set to `true` if the **Forward** button should be disabled on this page.
`submitonnext`	Set to `true` if the **Forward** button should submit the form on this page.

Attribute	Description
close-on-submit	This attribute applies to summary element only and allows for alternative way of quiting the player, as described below.

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 CSS on page 62.

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. For more information, see the `CONDITION` element on page 340. To hide or 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. An example condition is shown that indicates that the action set should be displayed only when that Action Set is on a page that is displaying within a Agenda Player.

```
<ACTION_SET ...>
  <CONDITION>
    <IS_TRUE NAME="CONTEXT" PROPERTY="inWizard"/>
  </CONDITION>
  ...
</ACTION_SET>
```

8.20 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 Server Development Environment (SDEJ) provides support for formatting a message catalog entry in this way so it can be returned to the client. For more information about working with message catalogs, see the *Server Developer's Guide*.

Once the message catalog entry has been formatted on the server side it must be assigned to a field which is based on the LOCALIZED_MESSAGE domain and returned to the client. The message entry is displayed according to the current locale and values is assigned to the message placeholders.

9 Custom data conversion and sorting

Use this information to learn about data formatting, parsing, validation, and sorting behavior in the Cúram web application.

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.

Warning: Unsupported Customizations

This information 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.

9.1 Data conversion and sorting operations

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.

The following operations are performed on the data values.

- **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. For more information about domain definition options, see the *Modeling Reference Guide*. 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 Converter plug-ins on page 282 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.

9.2 Data conversion lifecycle

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:

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:

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.

9.3 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 operations, 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, pre-parse, 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 in 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.

9.4 Domain plug-ins

Domain plug-ins are Java classes that conform to well-defined interfaces. 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

The process of writing plug-ins is straightforward. You must implement methods 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()`**

  This 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 9.11 Plug-in instance management on page 306. You should use the default implementations of these methods that are 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.

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 9.13 Generic parse operations on page 307 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 Custom error reporting on page 303). 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.

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. For more information, see the Javadoc for the `java.util.Comparator` interface. 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.

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 Default value plug-ins on page 290.

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.

9.5 Domain plug-in configuration

Domain plug-ins are configured by using an XML configuration file, which contains a `domains` root element. Insert a `domain` element for each domain. Within the `domain` element, `plug-in` elements 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. An example of a configuration file is shown.

```
<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>
```

The configuration elements are defined in the XML namespace. 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.

The example shows the configuration of two domains, which the actual default configurations for these domains in the Cúramapplication. Three plug-ins are configured for the Cúram root domain SVR_INT64. This set 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 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 are inherited from SVR_INT64. This particular configuration is used in 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” is formatted by the `InternalIDConverter` class as “1234567”. This formatting 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 are used.

A primary configuration file that is called *`domains-config.xml`* is 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 change this file as 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 by using the described mechanism.

Domain plug-in configuration follows the typical pattern for 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. The merge results in a single custom configuration, which can override or extend the primary 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 elements 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 inherit the new configuration. When you configure plug-ins, the name that is returned by a plug-in's `getName` method must match the `name` attribute value that is defined on the `plug-in` element in the configuration file. This naming 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 primary 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 previous example. Further configuration examples are included in 9.9 Customization guidelines for data conversion and sorting on page 295.

9.6 Provided domain plug-ins

Domain plug-ins for all of the root domain definitions and a few others are provided in the Cúram application.

Extending existing plug-ins

Rather than write your own plug-in implementation from scratch, it is easier to extend one of the 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 that 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:

Table 66: Behavior of the Abstract Plug-in Classes

Abstract Plug-in Class	Behavior
`AbstractConverter`	Returns the correct name for this type of plug-in: "converter".
	Formats an object that is an instance of `java.lang.Number` using the standard Java locale-specific number format. Other object types are formatted by calling their `toString` method.
	Pre-parses an object by trimming leading and trailing whitespace, compressing sequences of spaces, and converting to upper-case if specified by the UML domain definition options for the domain.
	Does not implement any parse operation.
	Pre-validates an object by checking its maximum and minimum values if these are specified by the UML domain definition options for the domain.
	Validates an object by throwing a `java.lang.NullPointerException` if an object is null, but otherwise performs no validation.
	Performs generic parsing by invoking the ordinary parse operation that must be implemented in the sub-class. See 9.13 Generic parse operations on page 307 for information on protecting the generic parse operation from side-effects.
	Performs generic formatting by invoking the object's `toString` method.
	Returns the correct value for the generic locale.
`AbstractComparator`	Returns the correct name for this type of plug-in: "comparator".
`AbstractDefault`	Returns the correct name for this type of plug-in: "default".
	Defines constants with suitable assumed default values for each of the root domains.
	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.
	Does not implement `getAssumedDefault`.

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 9.11 Plug-in instance management on page 306 for details).

While it is possible to write plug-ins from scratch, it is better to follow the guidelines 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.

Converter plug-ins

Converter plug-ins implement the format, parse, validate, and related operations.

The following converter plug-ins are provided. While most are pre-configured against certain domains, others must be configured. All of the plug-ins are in the `curam.util.client.domain.convert` Java package.

Table 67: Provided Converter Plug-ins

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`
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`

Domain	Converter Plug-in Class
CODETABLE_CODE	`CodeTableCodeConverter`
N/A	`SvrInt8BareConverter`
N/A	`SvrInt16BareConverter`
N/A	`SvrInt32BareConverter`
N/A	`SvrInt64BareConverter`

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:

Table 68: Behavior of the Format Operations

Plug-in Class	Formatting Behavior
`SvrBlobConverter`	Formatted as base-64 encoded strings. The base-64 encoding scheme is defined in RFC 2045.
`SvrBooleanConverter`	Formatted as the string values `true` or `false`. 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.
`DateTimeConverter`	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.
`CuramTimeConverter`	Formatted using the application time format. If the date-time is the system “zero” date-time, an empty string is returned.
`SvrDoubleConverter`	Formatted as numbers with grouping separator (e.g., thousands separator) and decimal point characters appropriate for the active user's locale.
`SvrFloatConverter`	Formatted in the same manner as the `SvrDoubleConverter`.
`SvrInt8Converter`	Formatted as numbers with grouping separator (e.g., thousands separator) characters appropriate for the active user's locale, but without any decimal point.

Plug-in Class	Formatting Behavior
`SvrInt16Converter`	Formatted in the same manner as the `SvrInt8Converter`.
`SvrInt32Converter`	Formatted in the same manner as the `SvrInt8Converter`.
`SvrInt64Converter`	Formatted in the same manner as the `SvrInt8Converter`.
`InternalIDConverter`	Formatted as numbers in a non-locale-specific manner without grouping separator characters.
`SvrInt8BareConverter`	Formatted in the same manner as `InternalIDConverter`.
`SvrInt16BareConverter`	Formatted in the same manner as `InternalIDConverter`.
`SvrInt32BareConverter`	Formatted in the same manner as `InternalIDConverter`.
`SvrInt64BareConverter`	Formatted in the same manner as `InternalIDConverter`.
`SvrMoneyConverter`	Formatted in the same manner as the `SvrDoubleConverter`, but with exactly two significant digits after the decimal point.
`SvrStringConverter`	Formatted literally, i.e., strings are not changed by the format operation.
`LocalizedMessageConverter`	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.
`CodeTableCodeConverter`	Formatted as the code description corresponding to the code value using the active user's locale and the domain's associated code-table.

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 upper-case. 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 then 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:

Table 69: Behavior of the Parse Operations

Plug-in Class	Parsing Behavior
`SvrBlobConverter`	Parsed as a base-64 encoded string.
`SvrBooleanConverter`	Recognizes any of `true`, `yes`, or `on` as Boolean `true` values, and any of `false`, `no`, or `off` as Boolean `false` values. The parsing is not case-sensitive or locale-aware. Other values are reported as errors.
`SvrCharConverter`	Parsed as a single Unicode character. The presence of extra characters is reported as an error.
`SvrDateConverter`	Parsed using the application date format and the active user's locale.
`DateTimeConverter`	Parsed using the application date and time formats and the active user's locale. The user's preferred time zone is assumed.
`CuramTimeConverter`	Parsed using the application time format. The server's time zone is assumed.
`SvrDoubleConverter`	Parsed as a number with optional grouping separator characters and decimal point characters appropriate for the active user's locale.
`SvrFloatConverter`	Parsed in the same manner as SVR_DOUBLE values.
`SvrInt8Converter`	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.
`SvrInt16Converter`	Parsed in the same manner as the `SvrInt8Converter`.
`SvrInt32Converter`	Parsed in the same manner as the `SvrInt8Converter`.
`SvrInt64Converter`	Parsed in the same manner as the `SvrInt8Converter`.
`InternalIDConverter`	Parsed in a non-locale-specific manner. Grouping separators are not permitted and for negative values the minus sign must be on the left.
`SvrInt8BareConverter`	Parsed in the same manner as the `InternalIDConverter`.
`SvrInt16BareConverter`	Parsed in the same manner as the `InternalIDConverter`.
`SvrInt32BareConverter`	Parsed in the same manner as the `InternalIDConverter`.

Plug-in Class	Parsing Behavior
`SvrInt64BareConverter`	Parsed in the same manner as the `InternalIDConverter`.
`SvrMoneyConverter`	Parsed in the same manner as SVR_DOUBLE values, but the magnitude of the values are limited to 1e13 to avoid the possibility of rounding errors.
`SvrStringConverter`	Parsed literally, i.e., strings are not changed by the parse operation.
`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.

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:

Table 70: Behavior of the Pre-Validate Operations

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

Plug-in Class	Max./Min. Size	Max./Min Value	Pattern Match
`SvrInt8BareConverter`	No	Yes	No
`SvrInt16BareConverter`	No	Yes	No
`SvrInt32BareConverter`	No	Yes	No
`SvrInt64BareConverter`	No	Yes	No
`SvrMoneyConverter`	No	Yes	No
`LocalizedMessageConverter`	Yes	No	Yes
`SvrStringConverter`	Yes	No	Yes
`CodeTableCodeConverter`	Yes	No	No

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.

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.

Table 71: Provided Comparator Plug-ins

Domain	Plug-in Class	Behavior
SVR_BLOB	`SvrBlobComparator`	Not sorted, as there is no useful sort order for these non-human-readable values.
SVR_BOOLEAN	`SvrBooleanComparator`	Sorted with Boolean `true` values before `false` values.
SVR_CHAR	`SvrCharComparator`	Sorted strictly numerically with no locale-aware processing.
SVR_DATE	`SvrDateComparator`	Sorted chronologically with the earliest date first.
SVR_DATETIME	`SvrDateTimeComparator`	Sorted chronologically with the earliest date-time first.

Domain	Plug-in Class	Behavior
CURAM_TIME	`CuramTimeComparator`	Sorted chronologically with the earliest time first. CURAM_TIME is based on the SVR_DATETIME domain, so values may included date information, but for comparisons, the date part is ignored and only the time part is used to determine the sort order.
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 Unicode Collation Algorithm for the locale. See Localized (cultural-aware) string sorting on page 289 for details.
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.

Domain	Plug-in Class	Behavior
CODETABLE_CODE	`CodeTableCodeCaseInsensitiveComparator`	Sorted identically to `CodeTableCodeComparator` except case is ignored.
CODETABLE_CODE	`CodeTableCodeLocaleAwareComparator`	Similar to the above, but the comparison of code descriptions uses the sorting rules defined by Unicode Collation Algorithm for the locale. See Localized (cultural-aware) string sorting on page 289 for details.

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 9.5 Domain plug-in configuration on page 279 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.

Localized (cultural-aware) string sorting

When sorting the textual strings, Unicode Collation Algorithm implementation is used to ensure the sort order expected by the users in different cultural environments.

The sorting order depends on both the current user locale and the so called collation strength. This strength is configurable to ensure the exact requirements for different languages and applications.

In order to change the default strength the application property '`curam.collator.strength`' should be set to one of the valid values summarized in the table Table 72: Collation strength summary on page 289 below.

'`curam.collator.strength`' is a static property and requires a server restart upon changing.

Table 72: Collation strength summary

'`curam.collator.strength`	Strength Name	Description
1	PRIMARY	Alphabetical sorting which accounts for the base letter differences.
2	SECONDARY	Diacritic sort order which takes into account character accents.

'curam.collator.strength	Strength Name	Description
3	TERTIARY	Character case based refinement of the sort order. This is the default value of the 'curam.collator.strength' and also the fall-back value where the set value cannot be interpreted.
4	QUATERNARY	Used to ignore punctuation when setting the sort order, and to account for minor differences. This level should also be used when sorting Japanese text according to JIS X 4061 standard.
5	IDENTICAL	The tie-breaking level, the character code point values are compared at this stage.

Note: If any value beyond the acceptable range is entered for the 'curam.collator.strength', a runtime fall-back to the default strength will occur. The notification of this will be recorded in the application server logs.

Note: As the collation strength is increased this can have an impact on performance.

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 in the `curam.util.client.domain.defaults` package.

Table 73: Default value plug-ins

Domain	Plug-in Class	Assumed Value	Displayed Value
SVR_BLOB	`SvrBlobDefault`	Empty BLOB	Empty BLOB
SVR_BOOLEAN	`SvrBooleanDefault`	False	False
SVR_CHAR	`SvrCharDefault`	Character zero	Character zero
SVR_DATE	`SvrDateDefault`	Zero date	Current date
SVR_DATETIME	`SvrDateTimeDefault`	Zero date-time	Current date-midnight
SVR_DATETIME	`SvrDateTimeDefaultCurrTime`	Zero date-time	Current date - Current time
SVR_DOUBLE	`SvrDoubleDefault`	Zero	Zero
SVR_FLOAT	`SvrFloatDefault`	Zero	Zero

Domain	Plug-in Class	Assumed Value	Displayed Value
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	`CodeTableCodeDefault`	Empty code string	Empty code string

The zero date and time is represented as midnight on January 1,0001 in the application, which is interpreted as if no date and time is set.

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.

The SvrDateTimeDefault plug-in is time-zone aware and the displayed value that 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. The result is that the time part of the date-time value is set to midnight regardless of the time zone settings.

9.7 Error Reporting

Infrastructure errors

Some built-in infrastructure errors exist for which the developer can perhaps do no more than retry the page or restart the web application. If these problems persist, technical support should be notified.

These errors should be reported by keeping a copy of the error page source. The information in the source of the page may be useful in identifying and resolving the error.

Exception classes

Many customizations require you to add 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 than 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.

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 that is used for error reporting in the client infrastructure. If you need only one error message catalog, you need only 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;
  }
}
```

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.

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'.
```

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 appended is treated as a single message catalog. The custom exception class in the example above refers to the message catalog as “custom.ErrorMessages”, 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 CDEJ project folder structure on page 36 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);
```

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));
```

9.8 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, you must understand the type of data that is being processed.

The following table shows the Java class used for data objects for each of the root domains.

Table 74: Classes Used for Java Object Representations

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`

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.

9.9 Customization guidelines for data conversion and sorting

Most customizations aim to control one or more of the data conversion or sorting operations. These guidelines show you how each of these operations can be customized. Follow these guidelines to 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. For more information about configuring plug-ins, see 9.5 Domain plug-in configuration on page 279. The Java source files for your custom plug-in classes are added to the web application in the same way as the Java source code files for your custom exception classes, see Custom exception classes on page 292. They are placed in their appropriate package folders in your `<client-dir>/<custom>/javasource` folder, where `<custom>` is the name of a custom component.

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.

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. The out-of-the-box Cúram application uses a different scheme for representing money values in different currencies. Custom converter plug-ins might 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 Cúram application includes 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);
  }
}
```

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. 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="http://www.curamsoftware.com/curam/util/common/domain-config">
    <dc:plug-in name="converter"
                class="custom.USDollarConverter"/>
  </dc:domain>
</dc:domains>
```

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();
  }
}
```

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 super-class, 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.

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.

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);
  }
}
```

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.

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. You can also 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 *Modeling Reference Guide* and summarized in Converter plug-ins on page 282. 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.

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);
    }
  }
}
```

The error message entry in the custom message catalog may look like this:

```
-200010=ERROR: The field '%0s' must be an even number.
```

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="http://www.curamsoftware.com/curam/util/common/domain-config">
  <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>
```

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 by 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 Comparator plug-ins on page 287. 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 9.8 Java object representations on page 294, 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:

1. Create a new sub-class of the `AbstractComparator` class described in Extending existing plug-ins on page 280.
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. An example of a custom comparator plug-in that sorts strings numerically for values that are limited to no more than ten characters is shown.

```
/**
 * 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;
  }
}
```

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. An example of sorting formatted values is shown.

```
/**
 * 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;
    }
  }
}
```

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 sorting zero dates 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);
  }
}
```

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 plug-ins 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 custom sorting configuration looks like this example:

```
<dc:domains xmlns:dc="http://www.curamsoftware.com/curam/util/common/domain-config">
  <dc:domain name="SVR_DATE">
    <dc:plug-in name="comparator"
                class="custom.ZeroDateComparator"/>
  </dc:domain>
</dc:domains>
```

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 according 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.

Custom error reporting

A plug-in might do operations exactly as you require, but you might 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.

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 Custom exception classes on page 292 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;
    }
  }
}
```

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'.
```

Domains that require this converter can be configured in the same manner as shown for the other converters above.

When using the error messages interception, please keep in mind, that 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.

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.

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 `getDisplayedDefault` 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 9.8 Java object representations on page 294.
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);
    }
  }
}
```

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 `getDisplayedDefault` 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 Custom exception classes on page 292. 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.

9.10 Type checking and null checking

It is not necessary to pass string or object values to the methods that are being checked to see if they are null or of the wrong type. 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.

9.11 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 Extending existing plug-ins on page 280) 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.Comparator` 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, that is, 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 Cúram application, no explicit default constructor is required.

9.12 Naming conventions

Custom domain plug-in classes can implement utility methods to support the implementation of the main interface methods.

For example, `_pad` method shown in Custom sorting on page 299. To avoid inadvertently overriding another inherited method, or using a method name that conflicts with a method introduced in a later Cúram release, prefix such utility methods with an underscore character as shown. Underscore characters are not used in the client infrastructure, so they 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. Some examples use the package name prefix “custom”, but this is not a requirement.

9.13 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 , such as 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 provided 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 can override both methods.

9.14 Code tables

Manage data conversion and sorting for code-table domains in 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 customizations 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 by the administration interface.

10 Online help development

You can embed context-sensitive help information in the Cúram web client with the Cúram online help system. Users can access help by clicking the Help button or on a page, the help page opens in a new tab or window. For accessibility, alternate text is provided for the help buttons.

10.1 The online help system

Where help is required on a page, help properties are created in the UIM file for the page and a properties file is updated with the help content. The help content is generated as part of the "client" build target. At runtime, online help is generated dynamically and does not need to be deployed separately to the main application. This helps developers to review their online help pages quickly.

Single Source Development

Each client page has an associated user interface metadata (UIM) file that defines its content, such as the links, buttons, and fields. The UIM file does not contain any actual text but references externalized properties files, which map property names to text strings. UIM files may also import VIM files. VIM files are in the same format as UIM files, they basically define a fragment of a UIM file. They also have properties files. Properties files that are associated with UIM or VIM have the same name but a different extension.

Each property that is defined in a property file is immediately followed by a corresponding help definition. This enables online help developers to easily compare and update UIM properties and help entries. In addition, having application properties and help within the same file removes the need to maintain and synchronize a separate set of files for the help system.

The online help content is composed of the help entries in the property files. These entries provide help about specific properties, such as fields or actions, in the associated UIM file. Within these property files, help entries are on the line immediately following the corresponding property definition. When the online help page is generated, all field and action help definitions are listed in an easy to understand table format.

Integrated Localization

Online help localization is integrated with application localization. When localized properties files are created for a particular locale, those property files will contain localized entries for both UIM properties and the help properties.

10.2 Online help elements

The online help content is composed of entries in client property files which correspond to page descriptions, fields, columns, links, and actions.

Client pages are installed in the *`webclient/components`* directory of the Cúram installation.

Properties are lines of text of the form:

```
PropertyName=Value of Property
```

If a button on a page is labeled in the UIM file with the property *`Button.Save`*, the following properties file entry will exist.

```
Button.Save=Save
```

To explain this in the online help, create another property called `Button.Save.Help`

```
Button.Save.Help=Use this button to save.
```

The online help framework generates this content into the online help format.

Page descriptions

Use the `Help.PageDescription` property to provide information about the user tasks on the page.

```
Help.PageDescription=You can view a clause
record. Clauses describe the precedents for a decision made
on an appeal and the legal articles that affect it. These
clauses can be dynamically inserted into decision documents.
```

Fields and columns

Help entries can also be provided for labeled fields or columns on a page. The online help system will generate a separate table for these help entries.

```
Field.Label.Language=Language
Field.Label.Language.Help=The language for the clause from the drop-down list of
 languages, e.g., English, French.
```

Links and actions

If there are any labeled links or action controls on the page, a help entry can be provided with a description for them. The online help system will create a table for them, complete with title and abstract.

```
ActionControl.Label.Save=Save
ActionControl.Label.Save.Help=The Save action creates a new record from the information
 entered on the page.
```

10.3 Adding or updating help content

To add new help content, you add a help property to the UIM file for the page and add the help content to the associated properties file. To update existing help, complete the following steps. Adjust the steps if you are updating domain-specific controls.

About this task

For domain-specific controls, the approach is slightly different. A good example of a domain-specific control is the address elements for a particular type of address. The field elements that are displayed on a page in the application depend on the locale that is specified. The format of the address elements displayed for an address in the US would be different from those displayed in the UK. For this reason, the online help cannot be specified for each of the elements within an address.

For example, the **Register Employer** page in the application has a registered address and a business address. The page properties file is *Employer_registerView.properties*. To update the online help regarding the Employer's business address and registered address, we could add help properties as follows:

```
# ADDING HELP HERE FOR REGISTERED ADDRESS
Field.Label.RegisteredAddress.Help=
The Employer can enter their registered address in the fields displayed.
 The format of the
Employers registered address will depend on the Country in which they reside.
Field.Label.BusinessAddress=Business Address
# ADDING HELP HERE FOR BUSINESS ADDRESS
Field.Label.BusinessAddress.Help=
The Employer can enter their business address in the fields displayed.
The format of the
Employers business address will depend on the Country in which they reside.
```

For more information about domain-specific controls, see 8 Domain-specific controls on page 215.

Procedure

1. Identify the correct property file to edit. The help text is contained in the property file with the same name as the UIM file.
   For example, to update the online help for Person Search in the application, update the help content in the *Person_search.properties* file, which is referenced by the *Person_search.uim* file.
2. You must modify the property file only in *webclient/components/custom*' directory. For example, if you need to update *webclient/components/core/Person/Search/Person_search.properties*, then copy this file into the *webclient/components/custom* directory. The *Person/Search* directories don't need to be created in the *custom* directory.
3. Update the help content in the properties file.
4. Build the client with your changes. Help is built by default as part of the client build target. The help is generated dynamically at runtime and does not need to be explicitly included in the application.

Related reference

Externalized strings on page 57

All string values in UIM files and JavaScript must be externalized, which helps with maintenance and allows the application to be localized. JavaScript, UIM pages, and UIM views can reference externalized strings.

11 Maintaining Dynamic UIM pages

Use this information to learn 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.

Important: Currently the development of custom dynamic UIM pages is only supported for the presentation of decision details only. Development of dynamic UIM for any purpose beyond this is not supported.

For more information about calculating and displaying decision details, see the *Inside Eligibility and Entitlement Using Cúram Rules Guide*.

11.1 Working in a development environment

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 is 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>
  <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 categorized (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.

11.2 Working in a running system

From the dynamic UIM administration screen in the application, you can add, edit, delete, or validate dynamic UIM pages in the Resource Store.

Go to the home dynamic UIM administration screen in the application by completing the following steps:

- 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

Search for dynamic UIM pages by category

To view the current list of dynamic UIM pages in the resource store, you must search based on the resource store category from the home dynamic UIM administration screen.

Procedure

1. Select a menu item for the drop-down list on **Category Search** field.
2. Click **Search** to see the list of dynamic UIM pages for the selected category.

Uploading a dynamic UIM page to the resource store

From the home dynamic UIM administration screen, you can add a dynamic UIM page can be added to the resource store.

Procedure

1. Select **New Dynamic Page...**. A modal dialog page with four mandatory fields opens.
2. 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, or an error message will be displayed.
3. Select the locale from the drop-down list on the **locale** field. The default is locale is English.
4. 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.
5. Use the **Browse** button on the **Properties File** field to navigate to the associated properties file to upload to the resource store.

Editing a dynamic UIM page in the resource store

From the home dynamic UIM administration, a dynamic UIM page can be edited in the resource store.

Procedure

1. 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 from the list actions menu. This should open a modal dialog page with three fields.
2. To download the current version of the dynamic UIM file and associated properties file from the Resource Store to the local file system for editing, select the **Download** button and save the zip file to the file system. The dynamic UIM file and associated properties file can then be extracted from the downloaded archive and edited as required.
3. To upload an edited dynamic UIM file, 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.
4. To upload an edited properties file, use the **Browse** button on the **Properties File** field to navigate to the associated properties file to upload to the resource store.

Deleting a dynamic UIM File from the resource store

From the home dynamic UIM administration, you can delete a dynamic UIM page from the resource store.

Procedure

1. From the list of dynamic UIM pages displayed, navigate to the dynamic UIM page that you would like to delete (by `Page ID`), and select the **Delete** from the list actions menu. A confirmation message is displayed.
2. Click**Yes**.

Results

A search for dynamic UIM pages in the resource store can confirm that the dynamic UIM page was removed.

Validating a dynamic UIM file in the resource store

From the home dynamic UIM administration, you can validate a dynamic UIM page can be validated in the resource store.

Procedure

From the list of dynamic UIM pages displayed, navigate to the dynamic UIM page that you would like to validate (by `Page ID`), and select **Validate** from the list actions menu. A modal dialog displays 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.

Publish dynamic UIM files

Changes to dynamic UIM files are not made public until you publish them to the resource store.

Procedure

Select **Publish** from the home dynamic UIM administration screen and confirm that you want to publish.

12 UI test automation

You can use the `data-testid` attribute to target specific UI components in your UI automation test framework.

All of the following components can be directly targeted on application screens with their `data-testid` attribute.

- Checkbox
- Cluster
- DatePicker
- Date Time
- Dropdown
- Inline menu items
- Login and Logout buttons, which are custom additions of the attribute to support automation only.
- Modal buttons, such as Next, Cancel, Back, or Submit.
- Multiselect Checkbox
- Search Popup
- Text Area
- Text Input
- Time Picker

The `data-testid` attribute

8.1.3.0

The format of the `data-testid` attribute concatenates the label to the UIM component prefix.

- The format of the `data-testid` attribute is `<component>_<labelPropertyKey>`.
- The `data-testid` attribute format for a form control with multiple parts is `<component>-<index>_<labelPropertyKey>`. Examples of such form controls include address fields, date-time pickers, and code table hierarchy dropdowns.
- If a component shares a label with another component on the page, to prevent duplicate `data-testid` attributes, an index suffix is appended to any subsequent instance.
  - The format of the `data-testid` attribute is `<component>_<labelPropertyKey>_<index_suffix>`.
  - The `data-testid` attribute format for a form control with multiple parts is `<component>-<index>_<labelPropertyKey>_<index_suffix>`.

The following default prefixes are used:

- The default prefix for view components is `uicomponent_<labelPropertyKey>`.
- The default prefix for form components is `formitem_<labelPropertyKey>`.
- The default prefix for a form control from a custom renderer is `formcontrol_<labelPropertyKey>`.

Custom renderer `data-testid` configuration

To generate a `data-testid`, you must call `context.getDataTestID()` from your custom renderer.

To ensure that your `data-testid` reflects your environment, you can set meaningful values in your configuration files. Set `dataTestidComponentPrefix` to a meaningful value, as shown in the following examples. If you don't set a custom value, the default value is used.

- **Example configuration files**
  The following example shows the configuration for the Text input and Text area components in the `TextEditRenderer` plug-in class.

```
<pc:plug-in purpose="curam-util-client::edit-text" name="edit-renderer"
class="curam.util.client.domain.render.edit.TextEditRenderer">
<pc:property name="dataTestidComponentPrefix">textinput</pc:property>
</pc:plug-in>
```

  The following example shows the configuration for the Date component in the `DateEditRenderer` plug-in class.

```
<pc:plug-in purpose="edit-date" name="edit-renderer"
class="curam.util.client.domain.render.edit.DateEditRenderer">
<pc:local-plug-in local-purpose="text-input" name="edit-renderer"

purpose="edit-date-text"/>
<pc:property name="dataTestidComponentPrefix">date</pc:property>
</pc:plug-in>
```

  The following example shows the configuration for the Time component in the `TimeEditRenderer` plug-in class.

```
<pc:plug-in purpose="curam-util-client::edit-time" name="edit-renderer"

class="curam.util.client.domain.render.edit.TimeEditRenderer">
<pc:property name="allow-blank">true</pc:property>
<pc:property name="time-domain">CURAM_TIME</pc:property>
<pc:property name="dataTestidComponentPrefix">time</pc:property>
</pc:plug-in>
```

  The following example shows the configuration for the Checkbox component in the `CheckboxEditRenderer` plug-in class.

```
<pc:plug-in purpose="curam-util-client::edit-boolean" name="edit-renderer"
class="curam.util.client.domain.render.edit.CheckboxEditRenderer">
<pc:property name="dataTestidComponentPrefix">checkbox</pc:property>
</pc:plug-in>
```

Benefits of the `data-testid` attribute compared to the previous approach

The new `data-testid` attribute enables improved targeting of specific UI components in automated UI framework selectors, which in turn improves the automation script and framework maintenance and robustness. Adopting the new attribute ensures that less effort is needed when you upgrade to future product versions. The following examples illustrate the change from the old to the new format.

Examples from 7.0.11 to 8.0.1

- **Scenario 1: Replacing an ID attribute that is often assigned a generic token on page load**

  For example, to target the `First Name` text input field in the second stage of the `Register Person` process.

  Previously, the input field in the DOM might be located with the following CSS selector:

  ```
  input[id="__o3id2"]
  ```

  This ID has the following issues:

  - The ID is generic. There's nothing in your UI automation code that obviously identifies it as being the ID for the first name field. If the code fails, it's more difficult to debug. Human-readable IDs always work better for clean code and better maintenance as the IDs are self-explaining about the DOM element they belong to.
  - The token, the id2 part of the ID value, is tied to the order in which the text input field appears in the UI. If the UI changes, and this text field becomes the third or the fourth text field to be loaded, this ID changes. This change makes this ID brittle and prone to failures. You need to update the IDs in your code repeatedly.

  With `data-testid`, the CSS selector becomes:

  ```
  input[data-testid="textinput_First Name"]
  ```

  The test ID points straight to the element, and is not tied to any generic tokens. The value consistently remains the same across all future iterations of the UI.
- **Scenario 2 - Replacing the use of the title attribute to uniquely identify UI elements in the DOM**

  For example, to target the `Last Name` text input field in the second stage of the `Register Person` process. Previously, the input field in the DOM might use the following CSS selector:

  ```
  input[title="Last Name Mandatory"]
  ```

  This ID has the following issues:

  - The title attribute value is tied to the English language in the UI. `Last Name Mandatory` as a value for the title doesn't work if you display the UI in a different language.
  - The ID is also tied to the text label of the input field. For example, if you change the label to **Surname**, the title attribute changes with it, breaking the CSS selector.

  With `data-testid`, the CSS selector becomes:

  ```
  input[data-testid="textinput_Last Name"]
  ```

  The test ID points straight to the element, an is not tied to the language used in the UI. The value consistently remains the same across all future iterations of the UI.

- **Scenario 3 - Replacing elongated CSS selector paths that have no identifying ID**

  For example, to target the **Next** button in the first stage of the **Register Person** process. Previously, you might locate the button in the DOM with the following CSS selector:

  ```
  div[class*="action-set"] > a[class*="btn-id-2"] > span > span > span[class="middle"]
  ```

  This ID has the following issues:

  - No unique ID or any other attribute is used to identify the physical button to click.
  - You must traverse through five elements in the DOM hierarchy in an exact order to reach the button in the DOM. If any part of that DOM hierarchy is changed, this selector breaks. It can be difficult and time consuming to follow this list of DOM elements from source to target to identify exactly where the path was broken.

  With `data-testid`, the CSS selector becomes:

  ```
  button[class*="bx--btn--primary"][data-testid$="_modal-button_1"]
  ```

  The test ID points straight to the element, and doesn't need an elongated DOM traversal to reach the target. The value consistently remains the same across all future iterations of the UI.

13 UIM reference

User interface metadata (UIM) is an XML dialect that is used to specify the contents of the Cúram web application client pages. UIM files must be well-formed XML.

The CDEJ translates UIM files into JSP files to be deployed to your web application server.

You can use any text editor to write UIM documents, but a specialized XML editor is preferable. The CDEJ includes an XML Schema file defining the syntax of a UIM document. In a schema-aware XML editor, you have access to many time-saving facilities such as auto-completion or syntax checking.

13.1 UIM document types

Four valid root elements are used to create two types of UIM document. Use the `PAGE`, `VIEW`, `PAGE_GROUP` and `APPLICATIONS` elements to create `PAGE` or `VIEW` UIM documents.

- **`PAGE`**
  `PAGE` defines a UIM page that is 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 is *`.uim`*. You can organize UIM pages into any folder structure in 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`**
  `VIEW` defines a portion of a page that can be included in 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 is *`.vim`*. You can organize view pages into any folder structure in a component folder, but the file names must be unique within that component.

13.2 UIM pages

A UIM page is the one of the main elements of the Cúram user interface. Developers create the UIM page definitions in files with a *`.uim`* extension, with each file corresponding to a single page. Individual pages consist of different elements such as page titles, labels, buttons, and links and the data content.

For the basic concepts behind UIM pages and an understanding of clusters, lists, action sets, action controls, containers, and fields, see the 2 Web client overview on page 17.

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.

13.3 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.

13.4 UIM page field level validations

Field level validations display in a cluster above the fields. The validation messages do not display in the same order as the fields are displayed.

13.5 Externalized strings

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

For more information, see Externalized strings on page 57.

13.6 UIM pages and views reference

This reference information describes the PAGE and VIEW elements and all of the child elements that they can contain with the exception of WIDGET elements.

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.

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`.

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 `JSCRIPT_REF` or `PAGE`. These values designate objects which supply or consume data. `JSCRIPT_REF` can only be a `TARGET` connection with either `PAGE` or a server interface defined in the `DISPLAY` phase, as the `SOURCE` connection. `TEXT` or `CONSTANT` can only be used when `TARGET` has a server interface defined in the `ACTION` phase. See the following connection types example.

```
<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>
```

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

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 that 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, that is, 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, which are described in the full UIM element descriptions.

`ACTION_CONTROL` element

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

Cancel button

A UIM action control with a TYPE of ACTION and no explicit link specified implicitly leads to the page which linked to it and had the "SAVE_LINK=true" specified in UIM for that link. This type of action is used for the Cancel button.

However, as no page history is memorized and supported, only a single implicit transfer back is possible. When consecutive screens contain 'previous' controls, only the 'previous' control on the last screen can correctly pass the flow back with the subsequent attempt to get to the page before it results in an error.

The screen flow with 'previous' controls is not recommended in the tabbed navigation content pane as it breaks usability. However, it can still be used in the modal wizard type flows or other contexts not bound to the tabbed navigation (like nested pages). If a screen flow contains more than one consecutive "previous" control, they must explicitly link to the page to go to when clicked.

File Downloads

An `ACTION_CONTROL` with the `TYPE` set to `FILE_DOWNLOAD` generates a hyperlink on the page. Clicking the hyperlink calls a special `FileDownload` servlet that is included in the CDEJ, which 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.

```
<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>
```

A `WIDGET` with the `TYPE` set to `FILE_DOWNLOAD` can also be used to generate a hyperlink to download a file. 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.

Each configuration for downloading files is contained in a `FILE_DOWNLOAD` element within the `FILE_DOWNLOAD_CONFIG` element 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 that contains the action controls to which this configuration is applied.
- `CLASS` for the name of the server interface to be called by the `FileDownload` servlet when the generated hyperlink is invoked.

The `FILE_DOWNLOAD` element can contain zero or more `INPUT` elements that specify 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 as they can in UIM documents.

Attributes

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

Table 75: Attributes of the ACTION_CONTROL Element

Attribute Name	Required	Default	Description
`LABEL`	Yes		A reference to an externalized string containing the label text for this action control. If the `TYPE` is `ACTION`, it is the text of the hyperlink. If the `TYPE` is `SUBMIT`, it is the caption of the **Submit** button.
`LABEL_ABBREVIATION`	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 `LIST`.
`TYPE`	No	`ACTION`	The type of action control to create. There are six types: • `ACTION` (the default) defines a link to another page. • `SUBMIT` forwards the page's form data to the action phase for processing. • `FILE_DOWNLOAD` defines a link that triggers the download of a file from the server. • `CLIPBOARD` copies a predefined value to the system clipboard. • `DISMISS` closes a pop-up page, see 8.18 Pop-up pages on page 255. • `SUBMIT_AND_DISMISS` combines a submit with closing a pop-up page.
`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		Opens 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 to submit when the user presses the Enter key. This is especially recommended when the submit action controls are in different action sets as the default action might be different than the first submit action declared on the page. The default action can be specified by setting this attribute to `true`. Only one submit action on a page can have a `DEFAULT` value of `true`.

Attribute Name	Required	Default	Description
ACTION_ID	No		A custom identifier for action controls of TYPE = SUBMIT. Use it in conjunction with the ACTION_ID_PROPERTY attribute of the SERVER_INTERFACE element to inform the server side code which action control made the server call. This attribute is only valid on action controls of TYPE = SUBMIT. The value of this attribute among the action controls on the page must be unique. 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. This attribute must be either specified on all action controls within the page or not specified on any of them. If this attribute is specified then the ACTION_ID_PROPERTY attribute of SERVER_INTERFACE must also be specified.
IMAGE	No		The value of this attribute refers to an externalized string that maps to a specific icon or graphic in the application. An action control with this attribute can only be used in a CONTAINER element.
ALIGNMENT	No	RIGHT	When contained in a page level ACTION_SET of a Modal Dialog, the ALIGNMENT attribute is supported. It defines the individual horizontal alignment of the action control. It can be set to LEFT or RIGHT. The default is right aligned.
8.1.3.0 APPEND_ELLIPSIS	No	true	Specifies whether to append ellipses (...) to the end of the Action Control label. This is useful when indicating that additional input will be required to complete the action associated with the Action Control.

Child elements

The ACTION_CONTROL element can contain the following child elements.

Table 76: Child Elements of the ACTION_CONTROL Element

Element Name	Cardinality / Description
LINK	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 **Cancel** 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. 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. 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.
CONNECT	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. Text to be copied to clipboard can be sourced from the server, the request or a properties file. 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.
SCRIPT	0..n. A script element associated with an action control. For a detailed description of this element see SCRIPT element on page 383. 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.

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

ACTION_SET element

The ACTION_SET element groups a number of ACTION_CONTROL elements, such as for actions menus. Action controls displayed differently depending on the context in which the action set is defined.

Standardized behavior for inline menu items on actions menus was introduced in 8.0.3 as optional and became the default behavior in 8.1. For more information, see 3.11 Tab, page, and list actions menus on page 168.

Where actions controls are in a drop-down menu, the SEPARATOR child element inserts a gray separator into the drop-down menu at the position that is indicated in the UIM file.

For inline action controls in actions menus, the SEPARATOR child element has no affect.

At the cluster or list level, the action controls can be displayed over or under the element with which the action set is associated and are aligned horizontally.

In all scenarios, conditional links that evaluate to false do not display if the 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.

Table 77: Attributes of the ACTION_SET element

Attribute Name	Required	Default	Description
TOP	No	true	Defines whether the action controls are displayed over the associated element. Can be set to true (the default) or false.
BOTTOM	No	true	Defines whether the action controls are displayed under the associated element. Can be set to true (the default) or false.
ALIGNMENT	No	DEFAULT	Defines the horizontal alignment of the set of action controls with the associated element. Can be set to LEFT, RIGHT, CENTER, or DEFAULT. The DEFAULT value corresponds to the CSS class *ac_default* in *curam_common.css*, which left-aligned by default. For a page-level ACTION_SET in a modal dialog, ALIGNMENT is ignored as the action control size and position are determined by the number of action controls and their type.

Attribute Name	Required	Default	Description
`TYPE`	No	`DEFAULT`	Defines the location of the action set. Can be set to `LIST_ROW_MENU` or `DEFAULT`. `LIST_ROW_MENU` is applicable where the `ACTION_SET` is in a `LIST`. It indicates that the action set is displayed as a list actions menu in a list row entry. **Note:** Do not use an `ACTION_SET` of type `LIST_ROW_MENU` to open a pop-up page, see Using the pop-up page on page 261.
`MAX_INLINE_ITEMS`	No		Specifies the maximum number of items to display inline for a specific page actions or list actions menu. By default, two page actions and one list action are displayed online. This attribute is only applicable for pages or lists when the `curam.actionmenus.display-inline.enabled.page` or `curam.actionmenus.display-inline.enabled.list` application properties are enabled. The `curam.actionmenus.display-inline.enabled.page` or `curam.actionmenus.display-inline.enabled.list` application properties are enabled by default. The value that you set for a page or list actions menu overrides the default value and any value that is set for the application by the `curam.actionmenus.max-inline-items.page` or `curam.actionmenus.max-inline-items.list` application properties. A maximum value of 4 for page actions and a maximum value of 2 for list actions is recommended. For more information, see 3.11 Tab, page, and list actions menus on page 168.

Child elements

The `ACTION_SET` element can contain the following child elements.

Table 78: Child elements of the ACTION_SET element

Element Name	Cardinality / Description
`ACTION_CONTROL`	1..n. The context of the `ACTION_SET` parent element determines which `ACTION_CONTROL` elements are valid.
`CONDITION`	0..1. Affects whether the `ACTION_SET` is displayed.
`SEPARATOR`	0..n. The ability to add a visual separator between action controls that display in the page actions dropdown. The separator has no affect for inline menu items.

CLUSTER element

Use the `CLUSTER` element to lay out related information on a screen. Clusters generally show field and label pairs in a number of columns. Clusters can also include layout elements, such as other clusters and lists, for more complex layouts. Clusters attributes can render data from data sources, such as server interface property values, externalized string values, or page parameter values. Fields can accept data and sent it to other data targets, such as server interface properties, or page parameters.

Attributes

The `CLUSTER` element has the following attributes.

Table 79: Attributes of the CLUSTER element

Attribute Name	Required	Default	Description
`TITLE`	See note in description.		A reference to an externalized string that contains the title string for this cluster. Avoid redundant titles when you use clusters for layout only. **Note:** The `TITLE` attribute is required for collapsible clusters and optional otherwise.

Attribute Name	Required	Default	Description
NUM_COLS	See note in description.	1	Using a single column for forms is recommended. Multi-column forms are more prone to misinterpretation and interrupt the vertical momentum of moving down the form. However, it makes sense to place short and logically related fields on the same row, such as **First name** and **Last name**. For more information about building forms, see the Carbon Design System. For content panels under 672 px, which is the IBM® Carbon Design System medium breakpoint, the cluster is responsive and breaks into a single column full-width layout. For more information about breakpoints, see 2x Grid in the IBM® Carbon Design System. For content panels larger than 672 px, you can use NUM_COLS. NUM_COLS sets the number of columns to display in the cluster, where a cluster column includes both the label and the field. To maintain consistent grid layout, clusters with three columns are divided into four, but only three columns are displayed. Clusters with more than four columns are divided into 8, but only the specified number of columns is displayed. Although you can use other numbers of columns, use 1, 2, 4, and 8-column clusters when you want to fill all available horizontal space. Use multiple nested or sibling clusters to achieve layouts of varying column spans. **Note:** The NUM_COLS attribute is required when a cluster contains a field element that has the ADDRESS_DATA domain data type. The NUM_COLS attribute is optional for all other domain data types.
TAB_ORDER	No	COLUMN	The order to lay out elements in a multi-column cluster. The elements can be ordered by ROW or COLUMN (default). If a CLUSTER has NUM_COLS set to 2 or more and contains a mix of LIST and FIELD elements, the TAB_ORDER must be set to ROW.
SHOW_LABELS	No	true	Set to true (default) to show labels at the side of the field values or false to show no labels.

Attribute Name	Required	Default	Description
LAYOUT_ORDER	No	LABEL (for read-only fields only)	For clusters with input or editable fields, labels display over the fields according to IBM® Carbon Design System guidelines. For clusters with read-only fields only, you can also use LAYOUT_ORDER. LAYOUT_ORDER sets whether labels display to the left or to the right of fields. Set the attribute value to LABEL to show labels to the left (default) or FIELD to show labels to the right. Labels and field values are laid out horizontally where the available space in the column is at least 8rem wide for labels and 16rem wide for field values. Where not enough space is available, values wrap under labels.
WIDTH	No		The percentage width of the containing area of the cluster. By default, this attribute is not set and a cluster fills 100% of its container up to the maximum width of the Carbon Grid. On extra large screens, the cluster might not always occupy 100% of the viewing area. The maximum width prevents columns that contain inputs and text from being too wide and having excessive white space. For more information about the Carbon Grid maximum width, see https://v10.carbondesignsystem.com/guidelines/2x-grid/implementation. If needed, you can override the Carbon Grid maximum width by setting this attribute to 100 to force the cluster to always occupy 100% of its container, regardless of screen size. If you want to set the cluster width to a size value other than 100% of its container, you can set this attribute to the correct percentage. The value that you set applies only to sizes over the medium breakpoint. Under the median breakpoint, the width is always 100%.
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. It is displayed under the title on the page.

Attribute Name	Required	Default	Description
LABEL_WIDTH	No		For clusters with input or editable fields, labels use 100% of the available width of the field, according to IBM® Carbon Design System guidelines. For clusters with read-only fields, you can use LABEL_WIDTH. LABEL_WIDTH specifies the percentage of the width of a cluster column for the label. By default, labels display at least 8rem wide. The max-width of read-only cluster labels is 12rem. This attribute applies even if SHOW_LABELS is set to false. You can, for example, use action controls instead 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 is applied to every other column. Whether it starts with the first or second column depends on the LAYOUT_ORDER attribute. The LABEL_WIDTH attribute does not apply to code table hierarchy fields when SHOW_LABELS is set to false, or when the FIELD attribute CONFIG has a value of CT_DISPLAY_LABELS. For more information about code table hierarchies, see the CONFIG attribute in FIELD element on page 345.
BEHAVIOR	No	EXPANDED	Where suitable, configure clusters to be collapsible to avoid the extra visual noise and interaction complexity that clusters can add to screens. Avoid nesting collapsible clusters inside other collapsible clusters. Collapsible clusters can be initially expanded or collapsed on a page by setting the EXPANDED or collapsed COLLAPSED attributes. To remove the collapsible function from a cluster, set the attribute to NONE. This attribute is applicable only when the property ENABLE_COLLAPSIBLE_CLUSTERS is not set or is set to true in *curam_config.xml*. For more information, see General configuration on page 64. This feature is not supported on clusters that contain Charts, Evidence Review Widgets, Evidence Comparison Widgets, or Evidence Tab Container Widgets.
SUMMARY	No		A reference to an externalized string that contains the summary of this cluster. The SUMMARY attribute describes the purpose and structure of a cluster.
SCROLL_HEIGHT	No		Specifies the maximum height of a scrollable cluster in pixels.

Child elements

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

Table 80: Child Elements of the CLUSTER Element

Element Name	Cardinality and Description
`CONDITION`	0..1. Affects whether the cluster is displayed.
`TITLE`	0..1. The `TITLE` element is displayed over the `CLUSTER`.
`DESCRIPTION`	0..1 The `DESCRIPTION` element on page 345 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, the element takes precedence over the attribute.
`ACTION_SET`	0..1. The action set can contain `ACTION_CONTROL` elements of any type. The action controls are displayed over or under 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.

Dynamic conditional clusters

A dynamic conditional cluster allows input field controls within one cluster on a page to control whether another cluster is displayed or not at runtime. The input field controls are mapped to a JavaScript function for a cluster. When the JavaScript function is evaluated at runtime, the selected value of the input field controls determines whether the cluster is displayed or not.

The `TYPE` attribute on the `CONDITION` element marks a conditional cluster as being dynamic. If it is not set to dynamic, the data that evaluates whether a cluster on a page is displayed or hidden comes from previous pages, not from any fields on the current page.

There are three potential sources of data for dynamic conditional clusters. One source of data is dynamic and comes from user interactions with input field controls on that page. The two other sources of data are static in nature and come from page connections and server interface connections.

For dynamic data the following input field controls can be used to control the behavior of a dynamic conditional cluster:;

1. Drop Down Lists.

These can be populated from a code table, for more information, see Populated from a code table on page 219. Alternatively they can be populated from a display phase server interface, for more information, see 8.4 Selection lists on page 217.

2. Radio Button Group,

   For more information, see 8.17 Radio button group on page 255.
3. Check box Fields.

   Single check box fields based on the `SVR_BOOLEAN` domain are supported

Data from a page connection or display phase server interface connection can be used in addition to dynamic data, to evaluate whether a dynamic conditional cluster gets displayed or not. For more information, see Connection types on page 325 . For more information about display phase server interfaces, see `SERVER_INTERFACE` element on page 384. For data from a page connection to control a dynamic conditional cluster, there needs to be a source page connection mapped to a `JSCRIPT_REF` target connection. For data from a display phase server interface connection to control a dynamic conditional cluster, there needs to be a display phase server interface connection mapped to a `JSCRIPT_REF` target connection

Only data types derived from the following underlying domains are supported:

- CURAM_BOOLEAN
- SVR_DATE
- SVR_DATETIME
- THREE_FIELD_DATE
- CURAM_TIME
- SVR_DOUBLE
- SVR_FLOAT
- SVR_INT8
- SVR_INT16
- SVR_INT32
- SVR_INT64
- SVR_CHAR
- SVR_STRING
- FREQUENCY_PATTERN

A dynamic conditional cluster can be displayed when a page is initially loaded (without any user interaction) if the data that controls the cluster evaluates to true within the configured JavaScript. When a user interacts with a input field control and selects a particular value from it, it is the raw value that will be immediately passed to a configured JavaScript function which can be used to evaluate whether the cluster will be displayed or hidden. When data is submitted to the server, regardless of whether it's source is static or dynamic, it is the raw value that will be sent. For example, a raw Boolean value will be sent for Boolean data, a raw unformatted string will be sent for frequency pattern data, a raw integer value will be sent for integer data, e.t.c

Data entered into the fields of a dynamic conditional cluster will only be submitted to the server if the cluster is displayed. If the cluster is hidden when the data is submitted, then default values will be submitted to the server. For example, for input field controls with integer data, the raw value '0' will be submitted. For input fields with string data, the raw the raw value '' will be submitted e.t.c. If a user changes their selection value to display a cluster that was previously hidden, any data entered into the fields within the cluster will be reset to the default value. Dynamic conditional clusters can also be pre-populated with initial values when the cluster in initially loaded. Initial data that has been specified in fields contained within dynamic conditional clusters will only be submitted to the server if that cluster is shown. However if a user changes their selection value to display a cluster with initial data that was previously hidden, the initial data will be displayed again to the user.

There are guidelines for configuring dynamic conditional clusters within the application:

- Nested dynamic conditional clusters are supported but it is recommended that there should be a limit of three nested levels deep, otherwise the performance and responsiveness of the page may be impacted.

  Additionally when configuring nested dynamic conditional clusters, the value of the `CONTROL_REF` attribute on each field must be unique and the value of each `EXPRESSION` attribute must be unique. For more information about the `EXPRESSION` attribute, see `SCRIPT` element on page 383.
- Multiple fields can control a single dynamic conditional cluster and the opposite is also true where one field can control multiple dynamic conditional clusters.
- A controlling field in VIM referenced in a UIM Page can control a dynamic conditional cluster present in that UIM page. The opposite also holds true where a controlling field in a UIM can control a dynamic conditional cluster present in a referenced VIM.

The following are unsupported for dynamic conditional clusters:

- Only the three aforementioned input field controls are supported for dynamic data. No other input field controls are supported.
- When configuring static data for dynamic conditional clusters, only single values are supported, not lists of values.
- Mandatory fields within dynamic conditional clusters are not supported.
- When configuring the values of the `CONTROL_REF` and `EXPRESSION` attributes, please ensure that it is not a JavaScript reserved word, otherwise a JavaScript error will occur.

Configuring conditional clustering

As stated static data can be configured to control dynamic conditional clusters by configuring a page connection or display phase server interface connection as a `SOURCE` connection and `JSCRIPT_REF` as the `TARGET` connection.

The value of the `PROPERTY` attribute on the `JSCRIPT_REF` target connection will be transformed in to a JavaScript variable with the same name and which can be referenced as *curam.dcl.getField('PROPERTY_VALUE')* , where PROPERTY_VALUE refers to the value of the `PROPERTY` attribute on the `JSCRIPT_REF` target. Likewise the value of the `CONTROL_REF` attribute will be transformed into a JavaScript variable with the same name and which can be referenced as *curam.dcl.getField('CONTROL_REF_VALUE')*, where CONTROL_REF_VALUE refers to the value of the `CONTROL_REF` attribute. The *curam.dcl.getField()* function gets the value from a data source as described above. See more information on this function within the JavaScript documentation.

The following steps are required to configure dynamic conditional clusters:

1. Configure the SCRIPT_FILE attribute of the PAGE element to configure the JavaScript file that contains the configured JavaScript functions. For more information, see `CONDITION` element on page 340.
2. The following example shows how to configure static data from a page connection and display server interface connection respectively.

```
<CONNECT>
  <SOURCE NAME="PAGE" PROPERTY="param1"/>
  <TARGET NAME="JSCRIPT_REF" PROPERTY="staticRef1"/>
</CONNECT>
<CONNECT>
  <SOURCE NAME="DISPLAY" PROPERTY="stringField1">
  <TARGET NAME="JSCRIPT_REF" PROPERTY="staticRef2">
</CONNECT>
```

3. To configure a field that will control a dynamic conditional cluster, add the `CONTROL_REF` attribute on the appropriate `FIELD` element. The following example shows this.

```
<FIELD LABEL="Field.Label" CONTROL_REF="widgetRef1">
 ....
</FIELD>
```

4. At the cluster level, set the `TYPE` attribute to `DYNAMIC` on the `CONDITION` element. For more information, see `CONDITION` element on page 340. An example of the setting is as follows:

```
<CLUSTER TITLE="Cluster.Title" ... />
   <CONDITION TYPE="DYNAMIC">
     <SCRIPT EXPRESSION="displayCluster1"/>
   </CONDITION>
</CLUSTER>
```

5. The recommended location for the JavaScript file that is referenced by the `SCRIPT_FILE` attribute and which contains the function to evaluate whether the cluster(s) are displayed or not. It should be located in the same location as the UIM page or an appropriate *`jscript`* directory that contains other JavaScript files. Any variables referenced by *curam.dcl.getField()* must refer to a `JSCRIPT_REF` property value or the value of a `CONTROL_REF` attribute or a JavaScript error occurs. The following example shows how a JavaScript function consumes the data configured and can be evaluated to display or hide a cluster.

```
function displayCluster1() {
  // field1 is the value defined in the CONTROL_REF attribute
  if(curam.dcl.getField('staticRef1') == true && curam.dcl.getField('staticRef1') ==
 'Astring' && curam.dcl.getField('widgetRef1') == 'A_CODE_TABLE_VALUE')
   {
      return curam.dcl.CLUSTER_SHOW;;
   }
   return curam.dcl.CLUSTER_HIDE;
 }
```

`CONDITION` element

The `CONDITION` element specifies the condition under which an `ACTION_SET`, `ACTION_CONTROL`, `LIST`, or a `CLUSTER` is displayed.

If the condition evaluates to true, the parent element is displayed.

If the condition evaluates to false, the parent element is not displayed with the following exception. An ACTION_SET or ACTION_CONTROL element displays disabled links if the HIDE_CONDITIONAL_LINKS attribute on the PAGE element or in the *curam_config.xml* file is set to false. Conditional ACTION_SETS and ACTION_CONTROLS are mutually exclusive so you must set the CONDITION element for one or the other, but not both.

If the condition equates to false for conditional action sets or action controls that display as drop-down menu items, a single disabled menu item called, 'No Contents' is displayed upon selecting the drop-down menu icon.

Attributes

The CONDITION element has the following attributes.

Table 81: Attributes of the CONDITION element

Attribute Name	Required	Default	Description
TYPE	No		Configuring the TYPE to be DYNAMIC enables a cluster be dynamically displayed depending on input from the current UIM page.

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.

Table 82: Child elements of the CONDITION element

Element Name	Cardinality / Description
IS_TRUE	0..1 If the property that is referenced by the IS_TRUE element returns true then the condition is true.
IS_FALSE	0..1 If the property that is referenced by the IS_FALSE element returns false then the condition is true.
SCRIPT	0..1 This is used to configure a JavaScript function that evaluates a Dynamic Conditional Cluster.

For Agenda Player specific use, see 8.19 Agenda Player on page 263

CONNECT element

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

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.

Table 83: Child Elements of the CONNECT Element

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

`CONTAINER` element

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.

Table 84: Attributes of the CONTAINER Element

Attribute Name	Required	Default	Description
`LABEL`	No		A reference to an externalized string that should be used as the associated label for this container.
`LABEL_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.

Attribute Name	Required	Default	Description
STYLE	No		A CSS class to be applied to this container.
SHOW_LABELS	No	false	When set to true the LABEL for a child input FIELD element will be displayed above the input FIELD on the screen if a LABEL attribute has been specified for it. 8.1.3.0 The CONTAINER SHOW_LABELS attribute takes precedence over its parent CLUSTER SHOW_LABELS attribute. For example, if the parent CLUSTER SHOW_LABELS attribute is false and the child CONTAINER SHOW_LABELS attribute is true, the CONTAINER LABEL (if specified) will not be displayed on the screen but the LABEL for each of the CONTAINER's child input FIELD elements will be displayed. **Note:** 8.1.3.0 This attribute is supported for CONTAINER elements which represent a Phone or Fax number input field only.

Child elements

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

Table 85: Child Elements of the CONTAINER 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.

DETAILS_ROW element

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 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.

Table 86: Attributes of the DETAILS_ROW Element

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.

Child elements

The `DETAILS_ROW` element contains the following child elements.

Table 87: Child Elements of the INFORMATIONAL Element

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.

`DESCRIPTION` element

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.

Table 88: Attributes of the DESCRIPTION Element

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

Child elements

The `DESCRIPTION` element can contain child elements as follows:

Table 89: Child Elements of the DESCRIPTION Element

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.

`FIELD` element

The `FIELD` element can specify a data value to be displayed in a `CLUSTER`, a value to be retrieved from the user by using 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.

Behavior of dropdowns

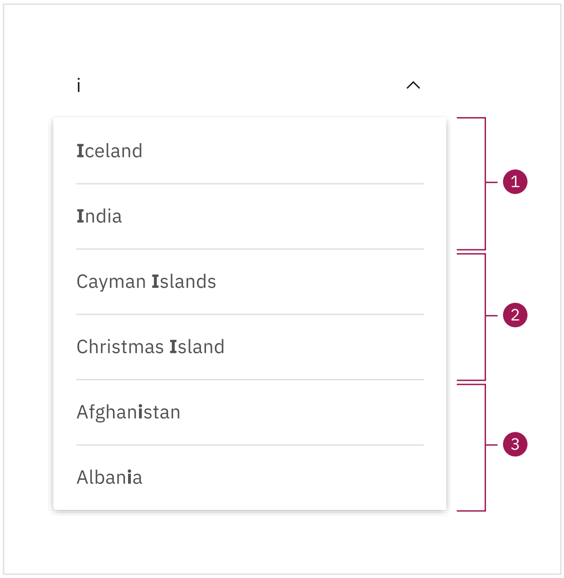
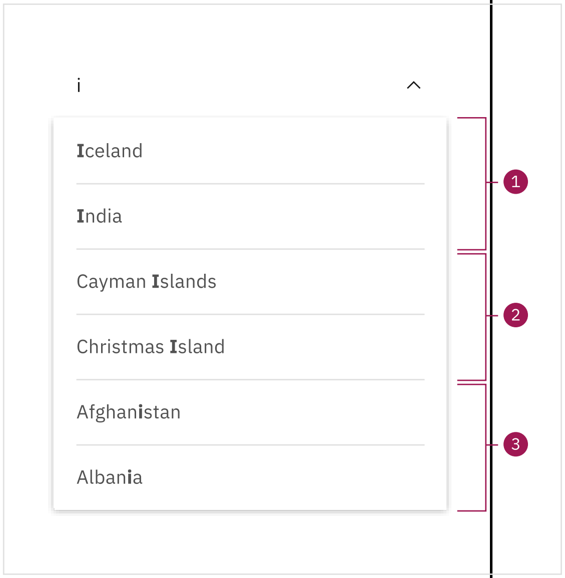
Note: IEG has not been updated to use Carbon components, so IEG does not use the Carbon dropdown.

The behavior of dropdowns depends on whether you use code table hierarchies or the Carbon Combo box. For more information about Carbon, see Carbon Design System related link.

Note: While not a Carbon component, the code table hierarchy dropdown was updated to match the Carbon Combo box dropdown style without changing the behavior from previous versions.

The following table outlines the differences in behavior between dropdowns that use code table hierarchies and dropdowns that use the Carbon Combo box.

Code table hierarchy dropdowns	**Carbon Combo box dropdowns**
To open the menu, the user must click the chevron directly.	To open the menu, the user can click anywhere or use the keyboard to Tab-focus on the field.

Code table hierarchy dropdowns	Carbon Combo box dropdowns
The list of options is sorted and filtered as caseworkers type into the field as  i Iceland India Cayman Islands Christmas Island Afghanistan Albania 1 2 3 follows: **1.** Options that start with the characters that are entered are listed, for example, entering 'i' displays Iceland. **2.** For options that contain more than one word, words that starts with the entered characters are then listed, for example, entering 'i' displays Cayman Island. **3.** Options that contain the entered characters are listed, for example, entering 'i' displays Albania. Matching characters are highlighted to indicate why an option was included. In previous releases, instead of filtering, the Combo box highlights the best match that is based on options that 'contain' the entered text.	The list of options is sorted and filtered as caseworkers type into the field as  i Iceland India Cayman Islands Christmas Island Afghanistan Albania 1 2 3 follows: **1.** Options that start with the characters that are entered are listed, for example, entering 'i' displays Iceland. **2.** For options that contain more than one word, words that starts with the entered characters are then listed, for example, entering 'i' displays Cayman Island. **3.** Options that contain the entered characters are listed, for example, entering 'i' displays Albania. Matching characters are highlighted to indicate why an option was included. In previous releases, instead of filtering, the Combo box highlights the best match that is based on options that 'contain' the entered text.
There is no **Clear** button.	There is a **Clear** button.

Code table hierarchy dropdowns	Carbon Combo box dropdowns
The selected item in the menu is indicated by a checkmark and a background color change.	The selected item in the menu is indicated by a checkmark and a background color change.
Users must explicitly select by using the Enter key or by clicking with the mouse.If users don't select a value, the entered text is cleared when focus is moved away from the field.	8.1.3.0 If a user enters invalid text into the input field to select an option and submits the form, a validation error will be displayed, providing details about why the selected option is invalid.
As the user enters text to filter options, the matching string of characters in the list of options are highlighted.	As the user enters text to filter options, the matching string of characters in the list of options are not highlighted.
On focus, the dropdown highlights options in the menu.	On focus, the dropdown highlights options in the menu.

When you use the `FIELD` element to display a code-table hierarchy on an edit or ready-only page, the following behavior applies:

- On an edit page, only one `FIELD` element is needed to display a code-table hierarchy with a domain definition that is inherited from `CODETABLE_CODE` that has the code-table name set to the lowest-level code table in a hierarchy. The CDEJ infrastructure automatically determines its code-table hierarchy and then displays however many drop-downs it has. For example, for a three-level hierarchy, then the three levels are displayed.
- On a read-only page, only the lowest level code-table value is displayed on the screen by using a single `FIELD` element as the edit page. The CDEJ infrastructure does not support displaying the full hierarchy.

Attributes

The `FIELD` element has the following attributes.

Table 90: Attributes of the `FIELD` element

Attribute Name	Required	Default	Description
`LABEL`	See description for details.		A reference to an externalized string that is used as the associated label for this field. Currently, the `LABEL` attribute is mandatory only when a `CONNECT` element exists that contains a `TARGET`. However, not providing a `LABEL` value results in the following accessibility violation so ensure that you provide a value for new and existing fields. `Each form control must have associated label`
`LABEL_ABBREVIATION`	No		A reference to an externalized string that contains the associated label abbreviation text for this field. This label abbreviation is placed only on table headers in a `LIST`.

Attribute Name	Required	Default	Description
DESCRIPTION	No		A reference to an externalized string that is displayed under the label text.
ALT_TEXT	No		A reference to an externalized string that is used as the alternative text for the field. This reference is applicable only when the field has a target connection, that is, it is an input field. If this attribute is added to a mandatory input field, the text `Mandatory` is appended to the externalized string. If this attribute is not specified, the `LABEL` is used. Browsers that are supported by the Cúram application display alternative 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. The default value is 100% if no value is specified. If you specify a width other than 100 on a FIELD that renders as a drop-down menu, the maximum width of the drop-down menu is determined by the width of the options. The following form components support `WIDTH_UNITS=CHARS` attributes: • Text input. • Text area. • Search pop-up. • Dropdown.
WIDTH_UNITS	No	PERCENT	The units in which the width is interpreted. This measurement can be `PERCENT` to indicate the percentage of the space available to the field, or `CHARS` to indicate the number of visible characters the field needs to accommodate.
HEIGHT	No	1	For input fields that resolve to a text input control, this input specifies the number of visible lines of text that the control displays. For input fields that resolve to a selection list, this value specifies the number of entries that are initially displayed. For example, 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 the *`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.

Attribute Name	Required	Default	Description
`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 is displayed. For example, numeric fields display a zero, string fields are empty, and date fields defaults to the current date.
`USE_BLANK`	No	`false`	If the field source is a code-table-based property, or a server interface list property, it is displayed in a list. If this attribute is set to true, an extra blank value is added to the beginning of the list and a Clear (x) button is displayed to clear the selected value. Do not use USE_BLANK=true for code tables that are used in a MULTIPLE_POPUP_DOMAINS control.
`CONTROL`	No	`DEFAULT`	The `CONTROL` attribute can take one of a number of values: • `DEFAULT` The field behaves in the standard fashion. • `SUMMARY`, `DYNAMIC`, `DYNAMIC_FULL_TREE`, and `FAILURE` These settings apply only to rules fields. For more information, see 8.6 Rules Trees on page 221. • `SKIP` Indicates that the field is only present to occupy space in a `CLUSTER` to balance the layout. No label or value is displayed. However, the label background still is presented. • `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 selections. • `CT_HIERARCHY_HORIZONTAL` Displays a list as a horizontal code-table hierarchy. • `CT_HIERARCHY_VERTICAL` Displays a list as a vertical code-table hierarchy. For more information about code-table hierarchies, see the *Server Developer's Guide*.

Attribute Name	Required	Default	Description
CONFIG	No		Identifies configuration details for this FIELD instance. This attribute can be used only 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 needs to match an ID on a config element in the *RulesDecisionConfig.xml* file. For more information about configuration, see Dynamic Rules View on page 222. CT_DISPLAY_LABELS displays labels for each code table in a code-table hierarchy. For more information about code-table hierarchies, see the CONTROL attribute.
INITIAL_FOCUS	No	false	A FIELD element, whose INITIAL_FOCUS attribute is set to true, gets focus when the page is displayed. In other words, the cursor is placed in that field ready for data entry. If no FIELD requests the initial focus, the cursor is 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.
PROMPT	No	false	This attribute is used to configure a placeholder value in the field that is associated with a Date Selector, if the field is blank. On focus, the placeholder text disappears to allow for data entry.
CONTROL_REF	No		This setting is used to configure Dynamic Conditional Clusters. The purpose of the CONTROL_REF is to set the controlling input. If something is selected, a cluster becomes visible. The CONTROL_REF attribute is set to an identifier that is evaluated by the JavaScript.

Child elements

The FIELD element can contain the following child elements.

Table 91: Child elements of the FIELD element

Element Name	Cardinality / Description
`CONNECT`	0..3. A field can contain up to three `CONNECT` elements. The `SOURCE` connection defines the initial value for the field. This value is the static value that is 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 to be set from the field value during the action phase. If a `TARGET` end point is specified, the `SOURCE` end point must be from a server interface property because domain information is needed to correctly format the value for display in the input control. If an `INITIAL` end point is used and the property is not a list value, it specifies the visible value of the field, which is read-only. The `SOURCE` value is hidden, and the pair of values can be changed only with a pop-up search page. The `TARGET` end point is supplied with the hidden value. 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 to be supplied to the `TARGET` end point. In this instance, the `SOURCE` end point specifies one of the hidden values in the list to be used as the initial list selection (the corresponding visible value is displayed).
`LINK`	0..1. Only valid for output fields. That is, fields with no `TARGET` connection end point. The value of the output field is used as the text for the hyperlink specified by this `LINK` element. If the field is based on a domain that needs a pop-up window, the `LINK` element can be used to supply parameters to the pop-up page. In this case, do not specify a `PAGE_ID` attribute for the `LINK` element, see Using the pop-up page on page 261.
`LABEL`	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 `LABEL` element on page 361 for more details.
`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 `SCRIPT` element on page 383 for more details and limitations on this element usage.

`FOOTER_ROW` element

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.

Two CSS classes are 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 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 for a FOOTER_ROW in a List is shown.

```
<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>
```

Attributes

The `FOOTER_ROW` element has no attributes.

Child elements

The `FOOTER_ROW` element contains the following child elements.

Table 92: Child Elements of the FOOTER_ROW Element

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`.

`IMAGE` element

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

Attributes

The `IMAGE` element has attributes as follows:

Table 93: Attributes of the IMAGE Element

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

Child Elements

The `IMAGE` element has no child elements.

`INCLUDE` element

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:

Table 94: Attributes of the INCLUDE Element

Attribute Name	Required	Default	Description
`FILE_NAME`	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.

Child elements

The `INCLUDE` element has no child elements.

`INITIAL` element

The `INITIAL` 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 8.18 Pop-up pages on page 255
- For selection lists populated from server interface properties see 8.4 Selection lists on page 217

Attributes

The `INITIAL` element has the following attributes:

Table 95: Attributes of the INITIAL Element

Attribute Name	Required	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.

Child Elements

The INITIAL element contains no child elements.

INFORMATIONAL element

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. For more information, see the *Server Developer's Guide*. 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 are never displayed within the context panel of the application, but instead are always 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.

Table 96: Child Elements of the INFORMATIONAL Element

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

`INLINE_PAGE` element

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.

Attributes

The `INLINE_PAGE` element has the following attributes.

Table 97: Attributes of the INLINE_PAGE Element

Attribute Name	Required	Default	Description
`PAGE_ID`	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.
`URI_SOURCE_NAME`	No		The name of the `SERVER_INTERFACE` instance to use as the source of the URI. This attribute is paired with `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_PROPERTY`	No		The name of the property to use as the source of the URI.

Child elements

The `INLINE_PAGE` element contains the following child elements.

Table 98: Child Elements of the INLINE_PAGE Element

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.

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.

`IS_FALSE` element

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.

Table 99: Attributes of the IS_FALSE Element

Attribute Name	Required	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.

See IS_TRUE element on page 358 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.

IS_TRUE element

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:

Table 100: Attributes of the IS_TRUE Element

Attribute Name	Required	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.

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.

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.

```
<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>
```

As the code within the JSP_SCRIPTLET element may contain reserved XML characters, 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, which prevents the XML parser from trying to interpret the contents of the block.

The reserved characters in XML are “ ' ”, “ " ”, “ & ”, “ < ”, and “ > ”. The respective XML character entities are “ &apos; ”, “ &quot; ”, “ &amp; ”, “ &lt; ”,and “ &gt; ”.

A common use of the JSP_SCRIPTLET element is to write code that redirects the current page to another page. Other uses are not recommended and patterns such as system parameter manipulation, the creation of logic for display purposes and the addition of JavaScript to such pages should be avoided. JSP SCRIPTLET on page 358, below, shows an example of how to use JSP_SCRIPTLET for redirection purposes.

```
<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>
```

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 *name* = *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 JSP SCRIPTLET on page 358, as this extra element would cause HTML content to be generated. Instead, you must declare the TextHelper instance within the scriptlet code as shown in the following example.

Note: When using `JSP_SCRIPTLET`, there is limited error handling capability so code should not make calls to secured server interface methods. Instead, secure the target page of any `JSP_SCRIPTLET` 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>
```

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 previous example. Code following that pattern must be included in 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.

`LABEL` element

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.

```
<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>
```

Attributes

The `LABEL` element has no attributes:

Child elements

The `LABEL` element can contain the following child elements.

Table 101: Child Elements of the LABEL Element

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.

LINK element

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.

Note that attributes that have the ability to link to external web pages or resources, such as mailto: links, 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 links back to the previous page. To keep this, the link must be to an internal Cúram page. To specify a link to an internal Cúram page, add the keyword 'curam:' before the link text.

Table 102: Attributes of the LINK Element

summary for complex table

Attribute Name	Required	Default	Description
PAGE_ID	No		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. 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. If the link is on an action control with a TYPE set to SUBMIT and this attribute is set to the value THIS, 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.
PAGE_ID_REF	No		A PAGE_ID can alternatively be specified by reference to an entry in the *CuramLinks.properties* file. This allows many links to refer to the same target page yet all can be updated by changing the entry in the *CuramLinks.properties* file.
URL	No		It is recommended to use the new URI attribute which is described below. The URL attribute is maintained for backward compatibility.

Attribute Name	Required	Default	Description
URI	No		Rather than link to another page in the application, the URI attribute allows the creation of a link to any URI whatsoever. This can be used to link to pages or other resources completely outside of the application. Parameters must be supplied by CONNECT elements within the LINK to ensure correct encoding.
URI_REF	No		A URI (or URL) can alternatively be specified by reference to an entry in the *CuramLinks.properties* file. This allows many links to refer to the same target yet all can be updated by changing the entry in the *CuramLinks.properties* file. The file can be placed in any component in the application.
URI_SOURCE_NAME	No		The name of the SERVER_INTERFACE instance to use as the source of the URI. This attribute is paired with 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_PROPERTY	No		The name of the property to use as the source of the URI.
OPEN_NEW	No	false	When set to true, this flag indicates that the linked page should be opened in a new window. When set to false (the default) the linked page will be opened in the current window. This setting is only supported for links to external sites.
SAVE_LINK	No	true	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 LINK 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 false so that subsequent pages do not return to their immediate previous page in the chain.
SET_HIERARCHY_RETURN_PAGE	No	false	This attribute is no longer used but has been retained in the UIM schema to avoid upgrade impact.

Attribute Name	Required	Default	Description
USE_HIERARCHY_RETURN_PAGE	No	false	This attribute is no longer used but has been retained in the UIM schema to avoid upgrade impact.
HOME_PAGE	No		If this attribute is set to `true`, 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. 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.
OPEN_MODAL	No	"false"	If this attribute is set to `true`, 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.
DISMISS_MODAL	No	"true"	If this attribute is set to `false`, the link will open the referenced page in the same pop-up window, modal or normal depending on what the browser supports.

Attribute Name	Required	Default	Description
`WINDOW_OPTIONS`	No	"width=700, height=auto-calculated"	Use the parameter to configure the size of each modal dialog. The value of the attribute is a comma-separated list of name value pairs. `Width` is the option that is supported. The `height` option is ignored, as the height is dynamically calculated. The modal content has a minimum height of 350 px. The `width` takes an integer value, which is converted to one of five sizes: x-small, small, default, large, and x-large. Each modal size has a responsive `width` that changes based on the browser size. As the browser decreases, the modal `width` percentage increases. This way, the modal `width` maintains a proper ratio between the modal and browser. Using any other parameters produces an error. Set the attribute only when `OPEN_MODAL` is set to `true` on the same `LINK` tag. The following table lists the width values and the corresponding sizes. **Width values** / **Size**: 0 - 420 → X-small; 421 - 576 → Small; 577 - 768 → Default; 769 - 1199 → Large; 1200+ → X-large

Width values	Size
0 - 420	X-small
421 - 576	Small
577 - 768	Default
769 - 1199	Large
1200+	X-large

Child Elements

The `LINK` element can contain the following child elements:

Table 103: Child Elements of the LINK Element

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 except the `TEXT`. Also, in the scenario where the `LINK` is inside an `ACTION_CONTROL` with `TYPE` = `SUBMIT`, 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.

Modal dialogs

Modal dialogs are typically used to present critical information or request user input that is needed to complete a user's workflow. When a modal dialog is active, users are blocked from the on-page content and cannot return to their previous workflow until the modal task is completed or the user closes the modal.

Comparison between modal dialogs and pop-up pages

A modal dialog is similar to a pop-up page as it opens a dialog box to display a page over the main application content. However, a 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 link, causes it to close. The parent page is changed, with the following exceptions:
  - If the linked page has the same ID as the current modal page, for example, a **Save & New** button or link, it is refreshed in the same modal window.
  - If the link has the `DISMISS_MODAL` attribute set to `false`, the linked page opens in the same modal window.
  - If the link has the `OPEN_MODAL` attribute set to `true`, it opens in a new modal window.
- The usage of modal dialogs is less complex that the usage of pop-up pages, and consists of using either one or two optional attributes on the `LINK` element.

Using modal dialogs

By default, a new page opens in the same window. You can choose to open a `LINK` element in a modal dialog by setting the `OPEN_MODAL` attribute to `true` as shown.

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

Setting `OPEN_MODAL` on a `LINK` inside an `ACTION_CONTROL` of type `SUBMIT` has no effect.

Setting `OPEN_MODAL=true` on a link implies `DISMISS_MODAL=false` so `DISMISS_MODAL=true` is ignored.

Similarly, setting `DISMISS_MODAL=false` implies `OPEN_MODAL=false` so you don't need to set it.

Configuring individual modal dialogs

To configure individual modal dialogs, set the `WINDOW_OPTIONS` attribute on a `LINK` element where the `OPEN_MODAL` attribute is set to `true`. The `WINDOW_OPTIONS` attribute is formatted as a comma-separated list of name value pairs.

Multiple options can be set by using this attribute. The parameter that is supported is `width`. `width` sets the width of the modal dialog and is measured in pixels as shown in the following example.

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

For the `WINDOW_OPTIONS` attribute, the `width` value is a simple integer with no alphabetic characters appended. Where no `width` parameter is specified, a default width of 600 pixels is used. If an unsupported parameter is placed in the `WINDOW_OPTIONS`, a build time exception results.

Where the `WINDOW_OPTIONS` attribute is also specified on the `PAGE` element of the page that a `LINK` element points to, it takes precedence over the value that is specified on the `LINK` element.

The `height` is automatically calculated to accommodate the page contents. The height is based on the content. If there is more content than can fit in the modal, then the height is based on a percentage of the browser window height. The `modal.dialogs.minimum.height property` in the `ApplicationConfiguration.properties` file no longer affects modal height and must not be used.

Controlling individual modal dialogs from custom JavaScript

You can control individual modal dialogs with custom JavaScript by using the provided `curam.util.UimDialog` API. For more information, see the full API documentation in *`<cdej-dir>\doc\JavaScript\index.html`*.

Loading custom non-UIM pages in a modal dialog

Custom non-UIM pages must hook into a specific set of API functions to work correctly in a modal dialog. These functions are provided by the `curam.util.Dialog` API. For more information, see the full API documentation in *`<cdej-dir>\doc\JavaScript\index.html`*.

Enabling movable modal dialogs

While users cannot interact with the on-page content while a modal dialog is active, you can use custom JavaScript to make modal dialogs draggable. Users can then drag a modal dialog to see content on the application page while they complete their task. A sample JavaScript file is provided to help you to customize this behavior for all modal dialogs at a global application level.

Copy the `webclient/components/core/WebContent/CDEJ/jscript/curam/application/modal/ModalHooks.js` file to a custom component so you can implement this behavior.

In your custom *`ModalHooks.js`* file, implement the `enableDraggableModals(modalRoot)` function. The `modalRoot` node of modal dialogs is available in this function and you can update it to enable movable modal dialogs across the application.

One of the most common ways to implement a movable modal dialog is to use event listeners. The event listeners can listen for when a user drags a modal dialog and update the position of the node by using the `style` attribute.

`LIST` element

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:

Table 104: Attributes of the LIST Element

Attribute Name	Required	Default	Description
`TITLE`	No		A reference to an externalized string containing the title string for this list. See also note below.
`STYLE`	No		The class name of the CSS style to associate with this list for formatting.
`DESCRIPTION`	No		A reference to an externalized string that provides more details about the list than the title alone. This string is displayed under the title on the page.
`SORTABLE`	No	`true`	Lists can be sorted by clicking on the appropriate headers. This behaviour is enabled by default without the use of the attribute. This attribute allows this feature to be controlled with `false` disabling the feature and `true` enabling it. Unsorted, sorted ascending, and sorted descending icons show that the column can be sorted, and the current sort status of the column.
`SUMMARY`	No		A reference to an externalized string containing the summary of this list. The `SUMMARY` attribute describes the purpose and structure of a list.
`SCROLL_HEIGHT`	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.

Attribute Name	Required	Default	Description
BEHAVIOR	No		Optional attribute that controls the display and behavior of the toggle button used to expand or collapse the list. Three value options are available for this attribute: • NONE which prevents the toggle button from being displayed in the list header. • EXPANDED : the toggle button is displayed and the list is initially expanded. • COLLAPSED : the toggle button is displayed and the list is initially collapsed. When the BEHAVIOR is not set for a list, its default value of EXPANDED is implied. 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 General configuration on page 64.
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.
DEFAULT_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 DEFAULT_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.

8.1.3.0

Note: Before version 8.1.3.0, only lists with an action menu (TYPE="LIST_ROW_MENU") configured in UIM had an automatic table layout in CSS. Starting with version 8.1.3.0, all lists, regardless of configuration, have an automatic table layout.

This feature offers several benefits:

- Automatic Column Widths: Each column's width is automatically adjusted based on the widest unbreakable content within that column. The browser examines the table's content first to set the column widths accordingly.
- Dynamic Layout: The layout of a list can change dynamically as content is added or removed.

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.

Table 105: Child Elements of the LIST Element

Element Name	Cardinality / Description
`TITLE`	0..1. The `TITLE` element will be displayed above the `LIST`.
`DESCRIPTION`	0..1. The `DESCRIPTION` element on page 345 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.

Element Name	Cardinality / Description
ACTION_CONTROL	Use the ACTION_CONTROL element within a list to describe the actions menu options. The ACTION_CONTROL element can define a text-based link, a button, or a file download link that users can activate on a page. For more information, see the ACTION_CONTROL element.
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, that is, 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, that is, fields with no target connection.
CONDITION	0..1. Affects whether or not the list is displayed.
DETAILS_ROW	By using the DETAILS_ROW element within a LIST element, users can expand each row to display more details about the row. When users expand the row, the child elements of DETAILS_ROW define the content that is displayed. Only the INLINE_PAGE element is supported as a child. For more information, see the DETAILS_ROW element.
FOOTER_ROW	0..n. This should be defined after all other child elements.
LIST_CONNECT	0..n. This should be defined after all other child elements. The only supported child elements are SOURCE and TARGET. The SOURCE connection must be a display phase bean. For more information, see the LIST_CONNECT element.
WIDGET	Use a WIDGET within a list. For example, you can use the MULTISELECT and the SINGLESELECT widgets in the first column of a list so that users can select a row.

Editable Lists

Pages might need some list items that are displayed to the user and are editable and other list items that are not. Editable lists are FIELD elements in a list that have SOURCE and TARGET connections. Read-only lists are FIELD elements in a list that have only SOURCE connections. A mixture of read-only fields and editable fields is permitted within a list.

FIELD elements that have a TARGET connection with no SOURCE connection are not supported. Only clusters that have input fields are to be used for creating business data in the application.

Two types of editable lists are available:

- **Editable lists controlled by a checkbox**

  If the first field in a list has SOURCE and TARGET connections and it has SVR_BOOLEAN as its underlying domain, the first column in the list is displayed as a checkbox. When a user selects the checkbox on a row, the other editable columns in that row can be edited (their value updated).

If a user does not select a checkbox, the other editable columns in that row are disabled and cannot be edited. In a checkbox-controlled editable list, all editable columns are controlled by the first column, the checkbox column.

- **Editable lists**

  If the first field in a list has `SOURCE` and `TARGET` connections and it doesn't have `SVR_BOOLEAN` as its underlying domain, it is treated as a normal editable list, where all of the editable columns are decoupled from one another. Even if there is a checkbox column within the list (not the first column), it cannot control whether the other editable lists are editable.

Within editable lists, you can submit hidden values (per list row) that are not visible in the list to the server with the `LIST_CONNECT` element on page 372. For example, in the context of a person object, for an editable list on a page that displayed details about the person, you might need to submit the persons ID (unique identifier) to the server as a hidden field without displaying it to the user. A single `LIST_CONNECT` element can be configured in the list to pass the unique ID for each person (each row represents a single person).

Only the following data types are supported on fields within editable lists:

- CURAM_BOOLEAN
- THREE_FIELD_DATE
- SVR_DOUBLE
- SVR_FLOAT
- SVR_INT8
- SVR_INT16
- SVR_INT32
- SVR_INT64
- SVR_CHAR
- SVR_STRING

`LIST_CONNECT` element

The `LIST_CONNECT` element is a child element of the LIST element and it defines a data connection between two connection end points. It can be used to pass a list in source to a target list on a page and is used on pages that contain editable lists.

Pages with editable lists can have some list items that are displayed to the user and are editable and other list items that are not. You can use the LIST_CONNECT element to pass the data for fields that are not displayed to the user to the target bean when the page is submitted, as shown in this example:

```
 <LIST>
   ..............
   <!-- This represents a list which is not displayed to the user but is submitted
with the page. -->
   <LIST_CONNECT>
     <SOURCE NAME="ListDisplayPhaseBean" PROPERTY="listField"/>
     <TARGET NAME="ListTargetBean" PROPERTY="listField"/>
   </LIST_CONNECT>
 </LIST>
```

For more information about editable lists, see Editable Lists on page 371.

Attributes

The `LIST_CONNECT` element has no attributes.

Child elements

The `LIST_CONNECT` element contains the following child elements.

Table 106: Child elements of the LIST_CONNECT element

Element Name	Cardinality / Description
`SOURCE NAME`	1..1. The `SOURCE` is the source of the value that is displayed in the field control. The SOURCE connection must be a display phase bean.
`TARGET NAME`	1..1. The `TARGET` is the property to which the value in the source is assigned.

`MENU` element

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 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 are appended to the "Navigation" menu. The `ACTION_CONTROL` elements must have the `TYPE` of `ACTION`.
- `DYNAMIC` : The menu is driven by XML data that is constructed on the server application.
- `INTEGRATED_CASE` : The menu is driven by XML data that is 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 display 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 that is stored in the server application.

Attributes

The `MENU` element has the following attribute:

Table 107: Attribute of the MENU element

Attribute Name	Required	Default	Description
MODE	No	STATIC	The type of menu to create. The mode can be STATIC (the default), NAVIGATION, DYNAMIC, INTEGRATED_CASE, IN_PAGE_NAVIGATION or WIZARD_PROGRESS_BAR. Static, navigation, and in-page-navigation menus contain one or more ACTION_CONTROL elements that represent links to other pages. • The static menu is displayed at the top of the main content area of the page. • Navigation menu items are appended to the navigation menu. • In-page-navigation menu items display at the top of the main content area. • The wizard progress bar displays at the top of the modal dialog content area. 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.

Child elements

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

Table 108: Child elements of the MENU element

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.

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 dynamic menu data sent by the application server is shown.

```
<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>
```

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 SDEJ provides an API to create the string representation of a message catalog entry shown in the previous example. For information about using message catalogs, see the *Server Developer's Guide* .
- `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` menus are *`DynamicMenuConfig.xml`* and *`ICDynamicMenuConfig.xml`* respectively. The following are examples of each configuration file.

Example of a `DYNAMIC` Menu 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 of an `INTEGRATED_CASE` 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>
```

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 separate 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 allows for the addition of a set of links, which are displayed as tabs embedded in a UIM page. Each UIM page in the set must define the same MENU element. The currently selected UIM page, for tab, is identified by the STYLE="in-page-current-link" attribute. This will differ on each of the UIM pages in the set and is set on the ACTION_CONTROL that matches the UIM page the MENU is contained in. See this example of the IN_PAGE_NAVIGATION menu in UIM.

```
<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>
```

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`. An example of wizard-type menu UIM is shown.

```
<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>
```

In the previous example, the connection in the `MENU` provides the identifier of the server-side resource describing this wizard.

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.

An example of the required properties in the resource store property file is shown.

```
    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
```

Table 109: Properties in the wizard defining resource

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.

Property Name	Description
`<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.
`Wizard.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.

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.

`PAGE` element

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:

Table 110: Attributes of the `PAGE` element

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`*.

Attribute Name	Required	Default	Description
`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 8.18 Pop-up pages on page 255 (i.e., search pop-up pages).
`SCRIPT_FILE`	No		The name of the script file containing 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.
`APPEND_COLON`	No		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 APPEND_COLON on page 70).
`WINDOW_OPTIONS`	No	"width=700, height=auto-calculated"	When the page is displayed in a modal dialog, use the parameter to configure the page. The value of the attribute is a comma-separated list of name value pairs. `Width` is the option that is supported. The `height` option is ignored, as the height is dynamically calculated. The modal content has a minimum height of 350 px. The `width` takes an integer value, which is converted to one of five sizes: x-small, small, default, large, and x-large. Each modal size has a responsive width that changes based on the browser size. As the browser decreases, the modal `width` percentage increases. This way, the modal width maintains a proper ratio between the modal and browser. Using any other parameters produces an error. **Width values** / **Size** 0 - 420 / X-small 421 - 576 / Small 577 - 768 / Default 769 - 1199 / Large 1200+ / X-large

Attribute Name	Required	Default	Description
TYPE	No	DEFAULT	Used to define specific types of UIM pages. Two types are supported, DETAILS and SPLIT_WINDOW. SPLIT_WINDOW enables the use of frames within the page. If the attribute is not present or is set to DEFAULT then frames are not used. See 8.19 Agenda Player on page 263 for an example of use. DETAILS defines a UIM page that will be used as a context panel page. For more information see Context panel UIM on page 150.
HIDE_CONDITIONAL_LINKS	No	TRUE	Set to true to hide conditional links that evaluate to false. Set to false to show a disabled conditional link that evaluate to false. This overrides the value of the HIDE_CONDITIONAL_LINKS element in the *curam-config.xml* file for that individual page (see APPEND_COLON on page 70).

Child elements

The PAGE element can contain child elements as follows.

Table 111: Child elements of the PAGE element

Element Name	Cardinality / Description
INCLUDE	0..1. This element can be used before any other child element of a PAGE element.
PAGE_TITLE	0..1. This does not apply to context panel UIM pages. In this case, the PAGE_TITLE element is mandatory. See Context panel UIM on page 150 for more information.
DESCRIPTION	0..1
SHORTCUT_TITLE	0..1
SERVER_INTERFACE	0..n. Multiple SERVER_INTERFACE elements are supported. However ,it is recommended that only one SERVER_INTERFACE with the PHASE attribute set to ACTION is defined per PAGE element. For more information, see SERVER_INTERFACE element on page 384
INFORMATIONAL	0..1
MENU	0..2. The page can contain one optional static and one optional dynamic menu as well as append extra items to the navigation menu.
ACTION_SET	0..1. In this context, the action set defines the set of action controls that display around the page's main content area.

Element Name	Cardinality / Description
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 needs 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 is preserved in the generated page.
CLUSTER	0..n. JSP_SCRIPTLET, CLUSTER and LIST can be intermingled freely and the order in UIM is preserved in the generated page.
LIST	0..n. JSP_SCRIPTLET, CLUSTER and LIST can be intermingled freely and the order in UIM is preserved in the generated page.
SCRIPT	0..n. A script associated with the PAGE that is activated in response to the specified event. See SCRIPT element on page 383 for more details.

Where a page is configured to contain a large number of scrollable list and cluster elements (approximately 15), it might cause JSP compile issues in Oracle WebLogic Server. This is due to a WebLogic Server 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.

PAGE_PARAMETER element

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:

Table 112: Attributes of the PAGE_PARAMETER Element

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

Child elements

The PAGE_PARAMETER element contains no child elements.

`PAGE_TITLE` element

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 Context panel UIM on page 150 for more information.

Note: In Curam, iFrames within expandable lists use the `title` attribute of HTML, and the `PAGE_TITLE` element in UIM is responsible for this. Every UIM page should have the `PAGE_TITLE` element. If not, iFrames within expandable lists will have empty `title` attributes, leading to confusion for accessibility screen readers.

Attributes

The `PAGE_TITLE` element has the following attributes:

Table 113: Attributes of the PAGE_TITLE Element

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 *`Image.properties`* file specifying the image file to use beside the title in the main content area.

Child elements

The `PAGE_TITLE` element can contain child elements as follows.

Table 114: Child Elements of the PAGE_TITLE Element

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.

Element Name	Cardinality / Description
DESCRIPTION	0..1 The DESCRIPTION element on page 345 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.

SCRIPT element

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:

Table 115: Attributes of the SCRIPT Element

Attribute Name	Required	Default	Description
EVENT	Yes		The JavaScript name of the event as defined in the W3C HTML recommendations. 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. Note that the ONCLICK event will be ignored for ACTION_CONTROL with a TYPE of CLIPBOARD. For more further information, see the ACTION_CONTROL element on page 326. In addition, by default when a link is clicked in the Cúram application the link is processed 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; someFunction() or someFunction(someParam) where *someParam* may be a global variable defined in script file.

Attribute Name	Required	Default	Description
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.
EXPRESSION	No		The name of the a JavaScript function identifier (excluding the parenthesis) that will be used to evaluate whether a dynamic conditional cluster will be displayed or not. The name should ideally reflect the encapsulated logic within the function.

Child elements

The SCRIPT element contains no child elements.

SERVER_INTERFACE element

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:

Table 116: Attributes of the SERVER_INTERFACE Element

Attribute Name	Required	Default	Description
NAME	Yes		A unique name for this instance of the server interface on this page.
CLASS	Yes		The name of the server interface class.
OPERATION	Yes		The name of the server interface operation on the class.

Attribute Name	Required	Default	Description
PHASE	No	DISPLAY	The phase of the page in which the server interface is called. This can be DISPLAY (the default) or ACTION. Server interfaces set to the DISPLAY phase are called as the page is displayed (i.e., the execution of the JSP page). Server interfaces set to the ACTION phase are only called in response to the activation of an ACTION_CONTROL with a TYPE of SUBMIT. It is recommended that only one SERVER_INTERFACE is set to the ACTION phase per PAGE.
ACTION_ID_PROPERTY	No		Specifies a name of the server access bean property that will be populated with ACTION_ID of the action control used to make the server call. The value 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 theProperty instead of the fully qualified dtls$theProperty). This attribute is only valid on server interfaces with PHASE = ACTION and must be specified on all server interfaces within the page or not specified on any of them. If multiple server interfaces specify 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. If this attribute is specified then the ACTION_ID attribute of ACTION_CONTROL element must also be specified.

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.

`SOURCE` element

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:

Table 117: Attributes of the SOURCE Element

Attribute Name	Required	Default	Description
`NAME`	Yes		The name of the `SERVER_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.

Child elements

The `SOURCE` element contains no child elements.

`TAB_NAME` element

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 Context panel UIM on page 150 for more information.

Child elements

The `TAB_NAME` element can contain child elements as follows:

Table 118: Child Elements of the TAB_NAME Element

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.

Element Name	Cardinality / Description
DESCRIPTION	0..1 The DESCRIPTION element on page 345 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.

TARGET element

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:

Table 119: Attributes of the TARGET Element

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.

Child elements

The TARGET element contains no child elements.

TITLE element

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:

Table 120: Attributes of the TITLE Element

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

Child elements

The `TITLE` element can contain child elements as follows.

Table 121: Child Elements of the TITLE Element

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.

`VIEW` element

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:

Table 122: Child Elements of the VIEW Element

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.
`JSP_SCRIPTLET`	See the `PAGE` element.
`CLUSTER`	See the `PAGE` element.
`LIST`	See the `PAGE` element.
`SCRIPT`	See the `PAGE` element.

13.7 UIM widgets reference

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.

Most widget types can only be defined within `CLUSTER` elements, any exceptions are described. There can also be restrictions on how many widgets of a particular type can be included in a single UIM document.

Use widgets 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.

`WIDGET` element

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:

Table 123: Attributes of the WIDGET Element

Attribute Name	Required	Default	Description
`TYPE`	Yes		The type of `WIDGET`. This can be one of the following: • `EVIDENCE_COMPARE` • `FILE_EDIT` • `FILE_UPLOAD` • `MULTISELECT` • `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.

Attribute Name	Required	Default	Description
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 Confirmation Pages on page 401.

Child elements

The WIDGET element can contain the following child element.

Table 124: Child Elements of the WIDGET Element

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

WIDGET_PARAMETER element

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:

Table 125: Attributes of the WIDGET_PARAMETER Element

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

Child Elements

The WIDGET_PARAMETER element can contain the following child element.

Table 126: Child Element of the WIDGET_PARAMETER Element

Element Name	Cardinality / Description
`CONNECT`	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: 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. The second way to connect a parameter is similar to the `CONNECT` elements in a `LINK` element. 1..n. `CONNECT` elements that each connect a `SOURCE` end-point to a `TARGET` end-point.

The FILE_EDIT widget

The FILE_EDIT widget allows users to edit a Microsoft™ Word® document on the user's local computer and then save the document to the Cúram database. The system can create a document automatically from a template where the template details can be set before the document is displayed to users to edit. The FILE_EDIT widget functionality is only supported on client machines that run Microsoft™ Windows™, with Microsoft™ Word installed locally.

Google Chrome version 104 and later and Microsoft™ Edge version 104 and later based on Chromium are supported by the FILE_EDIT widget feature. Internet Explorer is not supported for use with the feature.

Only the source and target documents and the template details are required by the FILE_EDIT widget. If key details, or other data, are required by the server interfaces that handle the document, use page parameters and page-level connections to provide these details.

Note: The native messaging-based solution that is supported by Google Chrome and Microsoft™ Edge requires a separate installation or configuration, see User computer configuration for the native messaging version on page 394.

When the page with the FILE_EDIT widget loads, it immediately starts the **File Edit Control** pane in the modal dialog. The pane displays the informational messages about the editing session initialization, other background events, and any error messages. The **File Edit Control** pane also allows for some minimal interaction with the application server.

The **Control panel** modal dialog can be closed up to the point that the Microsoft™ Word® application initializes correctly and the document opens. Thereafter, the close option is not available because closing Microsoft™ Word ends the process.

Once the document loads and is ready for editing, it is automatically saved locally and displays along with the corresponding notification in the Microsoft™ Word status bar and also the

Windows® taskbar. If so configured, see the *FILE_EDIT widget configuration* related link. Users can now edit the document and save it as needed.

Each document **Save** operation within Microsoft™ Word triggers the notification message in the application status bar and in the Windows® taskbar to notify users that their changes are saved locally, but not to the database. To save the interim versions of the document to the database, users can go back to the browser where there the **Commit changes** button is displayed in the **File Edit Control** pane. The button is initially disabled. It is enabled when the document is saved in the Microsoft™ Word application. When users select the **Commit changes** button, the current document version is passed back to the server and saved to the database. Users are notified of the result of the interim save in the pane and the **Commit changes** button is disabled again until the next **Save** operation in Microsoft™ Word.

Because of the specific invocation of the server interfaces by the FILE_EDIT widget, you cannot use any property of the *ACTION* phase server interface in a *SOURCE* connection of the submit button's **LINK** element.

When users finish editing the document, they finish the editing session by closing either the document that is being edited (if there are multiple documents open) or by closing Microsoft Word. At this stage, the application in the browser saves their final changes to the server. On successful save, the File Edit Control pane closes provided there are no error messages.

The application transfers users to the page that it is configured to go to as specified by the *ACTION_CONTROL* of *TYPE="SUBMIT"* in the UIM page that contains the FILE_EDIT widget. If no page is specified, the landing page is assumed to be the last visited page.

There are circumstances where this landing page is not available. In this case, the application searches to identify which page should be opened first based on configuration. Once a page is identified, it is used as the landing page, potentially changing the tab if necessary.

The FILE_EDIT widget can be used as follows, the *WIDGET* element must have the TYPE attribute set to FILE_EDIT. The following *WIDGET_PARAMETER* elements are required:

Table 127: Parameters to the FILE_EDIT widget.

Parameter Name	Required	Description and Connections
DOCUMENT	Yes	Defines the source document (usually a template) and the target to which to write the saved document. The parameter must contain a *CONNECT* element with a *SOURCE* set from a *DISPLAY* phase sever interface and a *TARGET* set from an *ACTION* phase server interface. Both fields should be Microsoft™ Word documents. The data type for both the source and target document must be SVR_BLOB.

Parameter Name	Required	Description and Connections
DETAILS	Yes	The template details that must be set in the document before it is presented to the user for editing. The parameter must contain a *CONNECT* element with a *SOURCE* set from a *DISPLAY* phase server interface. The details are in XML format. The data type for the template details must be SVR_BLOB.

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

```
<?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>
```

Your XML must use UTF-8 encoding to handle multi-byte characters. To preserve the correct encoding, any code that manipulates the XML must honor the encoding of the document. If the encoding is not honored, characters might display 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 is set.

Note: The data content that is passed to the template to populate the variables is limited to alphanumeric characters only. Any other characters might not be displayed correctly in the rendered document. In particular, the following restricted characters are used to populate the template variables. Do not use the restricted characters in data content that is passed:

- #|#
- |
- >
- New line characters

While editing the document in Microsoft™ Word, the user cannot browse to another page (which the modal **File Edit Control** pane and the absence of the **Close** button there would prevent) or close the browser. If the user attempts to close the originating browser window in the middle of the editing session, the browser warning is displayed notifying the user of the consequences.

If the user chooses to remain on the page, they can proceed with the editing or end the session by closing Microsoft™ Word or the document being edited. If the user chooses to leave the page, the editing session is terminated, and the document or Microsoft™ Word (if it was the only document open) closes along with the browser; the user changes are not saved in this case, however any saved interim changes before this termination happens are persisted in the database.

FILE_EDIT widget configuration

Use specific configuration settings to enhance the flexibility of the widget.

Table 128: FILE_EDIT widget configuration settings summary.

Setting Name	Location (.properties file)	Requi	Defaul value	Description
`fileedit.taskbar.messages`	*CDEJResources*	No	-	Can be set to either `true` or `false` values; prevents from the Word® status messages being duplicated by the Window® task bar notification messages. If set to `false`, no task bar notification will be displayed, while setting it to `true` or not having that property in *CDEJResources.properties* at all would cause the task bar notification messages to be displayed. Be aware that suppressing the additional task bar notification could affect the accessibility of the `FILE_EDIT` widget.
`fileedit.log.on`	*CDEJResources*	Yes	`false`	Can be set to either `true` or `false` values; when set to `true`, it causes the log (service) messages to be displayed in the `File Edit Control` panel dialog in addition to the regular status messages. These log messages do not have value to the end user and therefore are not translated (display in English). Turning on the logging should not normally be needed but might be useful when reporting or tracing the problems with the widget.

User computer configuration for the native messaging version

The Microsoft™ Word integration functionality for Google Chrome and Microsoft™ Edge Chromium is based around the extension and native messaging features of those browsers.

For more information, see the *System Administration Guide*.

The FILE_UPLOAD widget

The `FILE_UPLOAD` widget is a type of widget through which users can specify a file on a local computer to be uploaded to the server. Usually, the widget is displayed as a text field with a

Browse button beside it. The user can click the button to open a file dialog box and select a file for upload.

Button appearance The button is created by the browser. Therefore, the actual appearance of the button can vary depending on the browser that is being used. The normal widget attributes `WIDTH` and `WIDTH_UNITS` do not apply to the `FILE_UPLOAD` widget. Some browsers do not permit the width of the file name entry box to be set for security reasons. For example, if the width is set to zero width, the file name entry box could be hidden while it was still active.

Also, because the `FILE_UPLOAD` widget uses browser-specified controls, the text on the button is displayed in whatever locale the browser is set to, regardless of the locale that is configured in the application.

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 uploaded 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:

Table 129: Parameters to the FILE_UPLOAD Widget

Parameter Name	Required	Description and Connections
CONTENT	Yes	This parameter indicates the target connection for the actual content of the uploaded file. A single CONNECT element with a TARGET that connects to a property of an ACTION phase server interface is required.
FILE_NAME	No	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. If present, the parameter can include CONNECT elements for either or both 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. *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.*
CONTENT_TYPE	No	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 Microsoft Word document would have a content type of “application/msword”. A single CONNECT element with a TARGET that connects to a property of an ACTION phase server interface is required.
ACCEPTABLE_CONTENT_TYPES	No	An HTML page only allows certain types of content to be uploaded by default, where the actual default types depend on the browser. This parameter can specify the types of content that the page accepts. The value of the parameter must be a comma-separated list of content types. If a page contains more than one FILE_UPLOAD widget, the acceptable content types of all the widgets are pooled together and define what is acceptable for the page. This feature is a consequence of a limitation in the HTML specification. A single CONNECT element with a SOURCE that connects to a CONSTANT property is allowed.

Allow enough space for the Chrome browser

In Chrome, if the file upload widget is used adjacent to another field, once selected, the longer file name once selected might overlap with the label of that other field. To avoid this issue, do not have a file upload adjacent to another field, or allow for enough space in between the fields.

The FILE_DOWNLOAD widget

A `WIDGET` with the `TYPE` set to `FILE_DOWNLOAD` results in the generation of a hyperlink on the page. Clicking 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 a file download widget. For an example configuration, see the `ACTION_CONTROL` element on page 326.

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:

Table 130: Parameters to the FILE_DOWNLOAD Widget

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`.

Parameter Name	Required	Description and Connections
PARAMS	No	This optional parameter supplies the FileDownload servlet with the necessary parameters. 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.

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.

For more information, see ENABLE_SELECT_ALL_CHECKBOX on page 73.

Each check box can represents 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.

This example UIM document shows 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.

```
          <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"/>

  <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>
```

The main points to note in the 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 ENABLE_SELECT_ALL_CHECKBOX on page 73 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 Confirmation Pages on page 401

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

```
101|case121    102|case122    103|case123
```

NOTE: The MULTISELECT widget does not support the list pagination feature and all it's items will be displayed within one scrollable list. See PAGINATION on page 75 and LIST element on page 368 for more details on pagination support.

Table 131: Parameters to the MULTISELECT Widget

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.

Parameter Name	Required	Description and Connections
`MULTI_SELECT_INITIAL`	No	This parameter must include one `CONNECT` element that must specify a `SOURCE` end-point. The `SOURCE` end-point must be a string property containing the key data for the rows that are initially check when page is loaded.

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

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.

Note: The SINGLESELECT widget does not support the list pagination feature and all it's items will be displayed within one scrollable list. See PAGINATION on page 75 and `LIST` element on page 368 for more details on pagination support.

Table 132: Parameters to the SINGLESELECT Widget

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

The IEG_PLAYER widget

For more information, see the *Authoring Scripts using Intelligent Evidence Gathering Guide*.

13.8 Dynamic UIM

Dynamic UIM is cached in the resource store in contrast to static UIM, which is stored 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 13.6 UIM pages and views reference on page 324 are supported for dynamic UIM, except for those which are listed in 13.9 Unsupported features in dynamic UIM on page 403.

Refer to 11 Maintaining Dynamic UIM pages on page 313 on details about how to maintain dynamic UIM pages in the Resource Store.

13.9 Unsupported features in dynamic UIM

Learn about the elements and attributes that are not supported in dynamic UIM.

Unsupported `ACTION_CONTROL` features

The following `ACTION_CONTROL` features are not supported in dynamic UIM.

For information about the supported features of this element in static UIM, see `ACTION_CONTROL` element on page 326.

Table 133: Unsupported ACTION_CONTROL features

Name	Feature Type	Supported/Unsupported attribute values
`CONNECT`	Child Element	
`SCRIPT`	Child Element	
`CONDITION`	Child Element	
`LABEL_ABBREVIATION`	Attribute	
`IMAGE`	Attribute	
`CONFIRM`	Attribute	
`DEFAULT`	Attribute	
`ACTION_ID`	Attribute	
`TYPE`	Attribute	Only the values `ACTION` and `SUBMIT` are supported, all other values are unsupported An action of type SUBMIT is not supported within a list actions menu or a page level actions menu. All other submit actions are supported. • A list actions menu is an ACTION_SET element within a LIST that has a value of 'LIST_ROW_MENU' on it's 'TYPE' attribute. • A page level actions menu is an ACTION_SET defined at the PAGE level. See the `ACTION_SET` element on page 331 for further details.)

Unsupported `ACTION_SET` features

The following `ACTION_SET` features are not supported in dynamic UIM.

For full information about the supported features of this element in static UIM, see `ACTION_SET` element on page 331.

Table 134: Unsupported ACTION_SET features

Name	Feature Type
`CONDITION`	Child Element
`SEPARATOR`	Child Element
`TOP`	Attribute
`BOTTOM`	Attribute

Unsupported `CLUSTER` features

The following `CLUSTER` features are not supported in dynamic UIM.

For more information about the supported features of this element in static UIM, see CLUSTER element on page 333.

Table 135: Unsupported CLUSTER Features

Name	Feature Type	Supported/Unsupported attribute values
`TITLE`	Child Element	
`DESCRIPTION`	Child Element	
`WIDGET`	Child Element	
`SUMMARY`	Attribute	
`TAB_ORDER`	Attribute	

Note: Dynamic conditional clusters are not supported for dynamic UIM pages. For more information about dynamic conditional clusters, see Dynamic conditional clusters.

Unsupported `CONTAINER` features

The following `CONTAINER` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `CONTAINER` element on page 342.

Table 136: Unsupported CONTAINER Features

Name	Feature Type
`IMAGE`	Child Element
`LABEL_ABBREVIATION`	Attribute

Unsupported `FIELD` features

The following `FIELD` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `FIELD` element on page 345.

Table 137: Unsupported FIELD Features

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

Unsupported `INFORMATIONAL` features

The following `INFORMATIONAL` features are not supported in dynamic UIM.

Only informational messages whose connections endpoints are associated with a server interface defined in the `DISPLAY` phase, are supported. See `INFORMATIONAL` element on page 355 for more details on informational messages. Informational messages with other any type of connection endpoints are not supported.

Unsupported `INLINE_PAGE` features

The following `INLINE_PAGE` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `INLINE_PAGE` element on page 356.

Table 138: Unsupported INLINE_PAGE Features

Name	Feature Type
`URI_SOURCE_NAME`	Attribute
`URI_SOURCE_PROPERTY`	Attribute

Unsupported `LINK` features

The following `LINK` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `LINK` element on page 362.

Table 139: Unsupported LINK Features

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

Unsupported `LIST` features

The following `LIST` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `LIST` element on page 368.

Table 140: Unsupported LIST Features

Name	Feature Type	Supported/Unsupported attribute values
`TITLE`	Child Element	
`DESCRIPTION`	Child Element	

Name	Feature Type	Supported/Unsupported attribute values
`FOOTER_ROW`	Child Element	
`ACTION_CONTROL`	Child Element	
`SUMMARY`	Attribute	
`SORTABLE`	Attribute	
`PAGINATED`	Attribute	
`DEFAULT_PAGE_SIZE`	Attribute	
`PAGINATION_THRESHOLD`	Attribute	

Unsupported `MENU` features

The following `MENU` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `MENU` element on page 373.

Table 141: Unsupported MENU Features

Name	Feature Type	Supported/Unsupported attribute values
`CONNECT`	Child Element	
`MODE`	Attribute	Only the value `IN_PAGE_NAVIGATION` is supported, all other values are unsupported.

Unsupported `PAGE` features

The following `PAGE` features are not supported in dynamic UIM.

Table 142: Unsupported PAGE Features

Name	Feature Type
`FIELD`	Child Element
`CONTAINER`	Child Element
`WIDGET`	Child Element
`INCLUDE`	Child Element
`SHORTCUT_TITLE`	Child Element
`TAB_NAME`	Child Element

Name	Feature Type
`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

Unsupported `PAGE_TITLE` features

The following `PAGE_TITLE` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `PAGE_TITLE` element on page 382.

Table 143: Unsupported PAGE_TITLE Features

Name	Feature Type
`DESCRIPTION`	Child Element
`ICON`	Attribute

Unsupported `SERVER_INTERFACE` features

The following `SERVER_INTERFACE` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `SERVER_INTERFACE` element on page 384.

Table 144: Unsupported SERVER_INTERFACE Features

Name	Feature Type
`ACTION_ID_PROPERTY`	Attribute

Unsupported `WIDGET` features

The following `WIDGET` features are not supported in dynamic UIM.

For full details on the supported features of this element in static UIM, see `WIDGET` element on page 389.

Table 145: Unsupported WIDGET Features

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` and `MULTISELECT` are supported, all other values are unsupported

13.10 Dynamic UIM system initialization

The Dynamic UIM system can be initialized in two ways, 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.

To override the default initialization of the Dynamic UIM system - so that it is initialized when a Dynamic UIM page is first requested, add a configuration setting to the *`ApplicationConfiguration.properties`* file. This setting 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.

Notices

Privacy policy

Trademarks