Cúram 8.2

Citizen Context Viewer Developer Guide

Note

Before using this information and the product it supports, read the information in Notices on page 19

Edition

This edition applies to Cúram 8.2.

Contents

Chapter 1 Developing with the Citizen Context Viewer

Use this information to learn how to create custom UIM pages that launch the Citizen Context Viewer. Injection points for customizing the Citizen Context Viewer exist. Custom loaders and case handlers can be written. Node types and custom right-click menu options for the Citizen Context Viewer elements can be added.

1.1 Overview

The purpose of this guide is to provide instructions on how to customize the Citizen Context Viewer (CCV). It includes information on adding links to launch the CCV, on adding new elements to the CCV, and on localizing the CCV.

This guide is intended for developers responsible for integrating the CCV into specific components. Business analysts may find the guide useful in understanding the aspects of the CCV that can be customized to meet business requirements.

Prerequisites

There are two additional guides on the CCV: the *Cúram Citizen Context Viewer Guide* and the *Cúram Citizen Context Viewer Configuration* guide.

Sections in this Guide

The following list describes the sections within this guide:

- **Adding Links to Launch the Citizen Context Viewer**
  This section describes how to add links to custom uim pages that can launch the CCV.
- **Customizing the Citizen Context Viewer**
  This section describes the injection points for customizing the CCV and provides instructions on writing loaders and case handlers.
- **Adding Node Types and Right-click Menu Options**
  This section describes how a node type is defined for each element in the CCV tree and how to add right-click menu options for the CCV elements.
- **Localizing Citizen Context Viewer Data**
  This section provides information on localizing the data that appears as text in the CCV.

1.2 Adding Links to Launch the Citizen Context Viewer

It may be necessary to launch the CCV from customized pages. This can be achieved by creating links in the customized uims which reference the CCV javascript.

Example

The following code snippet outlines the link that specifies the CCV to be opened. The SOURCE should contain the field that holds the concern role ID for the person to be displayed in the CCV.

The TARGET should be as specified below:

```
<SCRIPT EVENT="ONCLICK"
        ACTION="openContextViewer(this,event)"
    SCRIPT_FILE="ContextViewerPopup.js"/>

    <LINK URI="../flex/Citizen_resolveCitizenViewer.jsp">

    <CONNECT>

    <SOURCE NAME="DISPLAY" PROPERTY="concernRoleID"/>

    <TARGET NAME="PAGE" PROPERTY="concernRoleID"/>

    </CONNECT>

    </LINK>
```

1.3 Customizing the Citizen Context Viewer

This section provides instructions on customizing the Citizen Context Viewer (CCV). The CCV interfaces define default implementations that can be replaced by injecting a custom implementation for that interface.

Injection Points in the CCV

Customization of the Citizen Context Viewer is facilitated via events and listeners. The following sections describe the areas where listeners might be implemented to carry out certain customizations of the CCV.

ContextNodeRootEvent

A listener to the ContextNodeRootEvent can add an implementation of the ContextCategory interface that will be used to populate the data displayed in the CCV. A implementation of the ContextCategory interface will allow loaders to be defined to show information in the CCV.

ContextCaseHandlerEvent

Cases that are displayed in the CCV can be created by a number of different components. Retrieval of information for these cases may therefore need variations depending on the type of case being read. The IContextCaseHandler interface describes case handlers for specific case types. These case handlers will know the specific retrieval methods for getting information such

as the case name. A default case handler exists that will get the case name from the case header table.

If the case type is not known by the OOTB CCV, then the default case handler will be used to read information for that case. If the default details for the case shown in the CCV are not specific enough then a customized case handler can be written. This case handler must implement the IContextCaseHandler interface and can be added by listening for the ContextCaseHandlerEvent setContextCaseHandlers method.

Writing a Loader

As previously mentioned, a loader class provides information that can be displayed in the CCV. Typically, a loader is written for each leaf node of the root node in the CCV.

For example, there are specific loaders for each of the leaf node categories including, Care And Protection, Communities, Family, and Dealings. These loaders gather all of the data for these categories. All loader classes must extend the ContextNode abstract class. A listener of ContextNodeRootEvent can be implemented to add new loaders to the existing set of CCV loaders via the method setChildNodesForContextType(Map<CONTEXT_TYPEEntry, ContextCategory> contextCategory)

Example

This loader example shows how a loader might be defined that reads Core interactions and sets up the data to be displayed in the CCV.

Comments are denoted by /** **/.

```
class ContextInteractionLoader extends ContextNode {
          /** The load method must be implemented by all loader
 classes. This
          is what is called when the CCV is opened. **/ public
 ContextNode
          load(Context_ID contextID) throws AppException,
          InformationalException { /**Set up and read the list of
 interactions
          from core for the context id, the context id in this
 instance is the
          concern role id for the citizen being displayed **/
          ClientInteraction clientInteractionObj =
          ClientInteractionFactory.newInstance();
 ClientInteractionKey
          clientInteractionKey = new ClientInteractionKey();
          ClientInteractionDtls clientInteractionDtls;
 ListInteractionKey
          listInteractionKey = new ListInteractionKey();
          listInteractionKey.concernRoleID =
 contextID.context_id;
          InteractionDetailsList interactionDetailsList =
          clientInteractionObj.list(listInteractionKey); /** If
 sensitivity
          settings do not allow this citizen to be shown then
 indicate that no
```

```
            interactions can be displayed **/ if
            (!ContextUtil.checkUserAuthorizationForParticipant(
            contextID.context_id)) { setLabelAllNotShown(

ContextUtil.getTextForLocale(BPOCONTEXTINTERACTION.ROOT),
            interactionDetailsList.dtls.size()); return this; } /**
Calling the
            setLabelIncludingChildren will display the Interaction
label as
            defined in the interaction message file and the number
of
            interactions **/ setLabelIncludingChildren(

ContextUtil.getTextForLocale(BPOCONTEXTINTERACTION.ROOT),
            interactionDetailsList.dtls.size()); /** Set the node
type. Node
            types define certain characteristics, including the
menu that will
            appear on right click **/ /** See Chapters 3 & 4 for
more
            information on Node Types and Menus **/
            setNodeType(CONTEXTNODETYPE.DEFAULTNODE); /** Set the
context ID. If
            a context ID is set then this will be used when
carrying out a
            specific action. **/ /** Such as opening an new page
from a menu
            item click **/ setContextID(0); if
            (interactionDetailsList.dtls.size() == 0) {
            setLabelIncludingChildren(

ContextUtil.getTextForLocale(BPOCONTEXTINTERACTION.ROOT),
            interactionDetailsList.dtls.size()); /** if there are
no
            interactions then add an empty child **/
addDefaultChild(new
            ContextLabelLoader(),
ContextUtil.getLocalisableStringForLocale((
            BPOCONTEXT.EMPTY)) .arg(ContextUtil.getTextForLocale(
            BPOCONTEXTINTERACTION.ROOT)).getMessage()); return
this; } else {
            Iterator interactionsIter =
interactionDetailsList.dtls.iterator();
            InteractionDetails interactionDetails = new
InteractionDetails();
            while (interactionsIter.hasNext()) { interactionDetails
=
            (InteractionDetails) interactionsIter.next();
            ContextInteractionLoader contextInteractionChild = new
            ContextInteractionLoader();
contextInteractionChild.setLabel(

curam.util.type.CodeTable.getOneItem( INTERACTIONTYPE.TABLENAME,
            interactionDetails.interactionTypeCode));
            clientInteractionKey.clientInteractionID =
            interactionDetails.clientInteractionID;
clientInteractionDtls =
            clientInteractionObj.read(clientInteractionKey); if
```

```
        (clientInteractionDtls.relatedType.equals(
        curam.codetable.RELATEDINTERACTIONTYPE.COMMUNICATION))
{ /** This is
        a communication interaction so set the corresponding
node type and
        menu **/ /** See Chapters 3 & 4 for more information on
node
        types and menus **/
contextInteractionChild.setNodeType(
        CONTEXTNODETYPE.COMM_INTERACTION_NODE); /** Set the
context id for
        this child node to be the interaction relatedID. This
will be used
        when opening any interaction pages from a CCV
interaction menu **/
        contextInteractionChild.setContextID(
        clientInteractionDtls.relatedID); } else if
        (clientInteractionDtls.relatedType.equals(
        curam.codetable.RELATEDINTERACTIONTYPE.PAYMENT)) { /**
This is a
        payment interaction so set the corresponding node type
and menu **/
        /** See Chapters 3 & 4 for more information on node
types and
        menus **/ contextInteractionChild.setNodeType(
        CONTEXTNODETYPE.PAY_INTERACTION_NODE); /** Set the
context id for
        this child node to be the interaction relatedID. This
will be used
        when opening any interaction pages from a CCV
interaction menu **/
        contextInteractionChild.setContextID(
        clientInteractionDtls.relatedID); } else {

contextInteractionChild.setNodeType( CONTEXTNODETYPE.DEFAULTNODE);
        contextInteractionChild.setContextID(0); } /** Add the
child to the
        loader **/ addChild(contextInteractionChild); } }
return this;
        }
```

Writing a Case Handler

A case handler can be used to get the name of a case where the retrieval of the case details is not covered by the CCV default case handler. The case handler can also be used to check property settings defined for a given case type. All case handlers must extend the abstract ContextCaseHandler class.

Example

This example outlines how a case handler might be defined for Integrated cases.

Comments are denoted by /** **/.

```
class IntegratedCaseHandler extends
```

```
        ContextCaseHandler { @Inject private
Provider<ContextCore>
        contextCoreProvider; public String getCaseName(long
caseID) throws
        AppException, InformationalException { return
        CodeTable.getOneItem(PRODUCTCATEGORY.TABLENAME,
        this.readType(caseID)); } protected String
getShowAllStatuses() {
        /** check to see if CCV is configured to display all IC
case
        statuses **/ return
ISEConfigurationUtility.getProperty(
        EnvVars.ENV_CCV_CASE_SHOW_ALL_IC_STATUSES,
        EnvVars.ENV_CCV_CASE_SHOW_ALL_IC_STATUSES_DEFAULT); }
protected
        String getShowAllTypes() { /** check to see if CCV is
configured to
        display all integrated case types **/ return
        ISEConfigurationUtility.getProperty(
        EnvVars.ENV_CCV_CASE_SHOW_ALL_IC_TYPES,
        EnvVars.ENV_CCV_CASE_SHOW_ALL_IC_TYPES_DEFAULT); }
protected
        ArrayList<String> listAcceptableCaseStatuses() { return
        ISEConfigurationUtility.getListPropertyValues(
        EnvVars.ENV_CCV_IC_CASE_STATUSES_TO_DISPLAY,
        EnvVars.ENV_CCV_IC_CASE_STATUSES_TO_DISPLAY_DEFAULT); }
protected
        ArrayList<String> listAcceptableTypes() { return
        ISEConfigurationUtility.getListPropertyValues(
        EnvVars.ENV_CCV_IC_CASE_TYPES_TO_DISPLAY,
        EnvVars.ENV_CCV_IC_CASE_TYPES_TO_DISPLAY_DEFAULT); }
protected
        String readType(long caseID) throws AppException,
        InformationalException { /** read the case type, how
this is done
        can be specific to the case type **/ ContextCore
contextCore =
        contextCoreProvider.get(); CaseHeaderDtls
caseHeaderDtls =
        contextCore.readCaseHeaderByCaseID(caseID); return
        caseHeaderDtls.integratedCaseType; } public String
        getShowAllCaseParticipantRoles() { return
        ISEConfigurationUtility.getProperty(
        EnvVars.ENV_CCV_CASE_SHOW_ALL_IC_CPROLES,
        EnvVars.ENV_CCV_CASE_SHOW_ALL_IC_CPROLES_DEFAULT); }
protected
        ArrayList<String> listAcceptableCaseParticipantRoles()
{
        return ISEConfigurationUtility.getListPropertyValues(
        EnvVars.ENV_CCV_IC_CPROLE_TYPES_TO_DISPLAY,
        EnvVars.ENV_CCV_IC_CPROLE_TYPES_TO_DISPLAY_DEFAULT); }
        }
```

1.4 Adding Node Types and Right-click Menu Options

This section provides instructions on adding node types to the Citizen Context Viewer tree and on adding right-click menu options for menu items in each node.

Adding Node Types

When a loader retrieves data to be displayed by the context viewer, each element in the context viewer tree will be assigned a node type. The node type describes the following about the element to which it is assigned.

- **CONTEXTNODETYPE**
  This attribute holds the ID for the node type. The ID maps to a code table value in the ContextNodeType code table to allow the node type to be referenced in code via its java identifier.
- **Loadable**
  This boolean flag allows the context viewer to decide whether expanding an element results in a server call to retrieve additional context data.
- **menuID**
  This is the ID for the menu to which this node type links.
- **isBranch**
  This boolean flag allows the context viewer to decide whether the node has children. A node can have children as a result of a read from the CCV, or as a result of a user click (which results in a read from the CCV to get the children). If loadable and isBranch are both true then the element will indicate that children are available, but the children will not be read until the user clicks the element.

Example

Example of a context node type:

```
<?xml version="1.0" encoding="UTF-8"?>
      <CONTEXTNODETYPE id="CNDT101">
      <loadable value="true"/>
      <menuID value="CNMU101"/>
      <isBranch value="true"/>
      </CONTEXTNODETYPE>
```

Adding Right-click Menu Options

It is possible to add menu items that will appear upon right click of an element in the context viewer. This is done via the contextmenu dmx files loaded as initial data.

Setting Attributes for Context Menu

The following attributes can be set for a context menu:

- **CONTEXTMENU**
  This attribute holds the ID for the menu. This ID is referenced from a node type definition.
- **MENUITEM**
  There can be none, one, or many menu items and they map to one of the right-click options displayed in the context viewer.
- **MENUITEMLINK**
  This attribute defines the page or url that will be opened when the right click option is selected.

Menu item links can contain the following:

- **PAGEPARAM**
  This defines the page parameter required by the page being opened.
- **PAGECONFIG**
  This specifies the javascript window.open options when opening the page on right-click of the menu item.

Example

Example of a context menu:

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

      <CONTEXTMENU id="CNMU103">

      <MENUITEM value="View Contact Details">

      <MENUITEMLINK
      value="MultidisciplinaryTeamMember_viewModalPage.do">

      <PAGEPARAM value="multidisciplinaryTeamMemberID"/>

      <PAGECONFIG
      value="height=450,width=800,top=100,
         left=100,resizable=yes"/>

      </MENUITEMLINK>

      </MENUITEM>

      </CONTEXTMENU>
```

1.5 Localizing Citizen Context Viewer Data

Any data that is displayed by the context viewer that is not read directly from application database tables is read via data loaded from message files.

For example, the text that is displayed when case security is set, and a case cannot be viewed in the context viewer, comes from the Context message file. Since the content of the xml tree displayed by the context viewer is built on the server, the localization substitution must also be performed on the server. includes sample code where messages are read in localized form.

1.6 Compliancy for the Citizen Context Viewer

This section explains how to develop in a compliant manner. By following these considerations, customers will also find it easier to upgrade to future versions of Cúram.

Public API

The Citizen Context Viewer has a public API which you may use in your application code. This public API will not have any components changed or removed without following Cúram standards for handling customer impact.

Identifying the API

The JavaDoc shipped is the sole means of identifying which public classes, interfaces and methods form the public API.

Outside the API

The Citizen Context Viewer also contains some public classes, interfaces and methods, which do not form part of the API.

Important: To be compliant, dependencies on any class or interface should not be made. No methods should be called other than those described in the JavaDoc.

Classes, interfaces and methods outside of the public API are subject to change or removal without notice. Unless otherwise stated in the JavaDoc, you must not place any of your own classes or interfaces in the same package as that of the Citizen Context Viewer.

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those

websites are not part of the materials for this Merative product and use of those websites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Merative, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.