Cúram 8.1.2

Case Audits Developer's Guide

Note

Before using this information and the product it supports, read the information in Notices on page 25

Edition

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

Contents

1 Developing with Case Audits

Learn how to record new algorithms for producing random samples of cases for audit. Dynamic selection queries can be used to generate a list of cases for audit. There are a number of options available for selection queries. Case data from external sources can be used to generate a list of cases for audit.

1.1 Overview

The purpose of this guide is to outline the available customization options for the Case Audits component and to provide instructions on how to implement these customizations.

This guide is intended for developers and architects that intend to implement an auditing solution by customizing Case Audits.

Prerequisites

Before reading this guide the reader should be familiar with the *Cúram Case Audits Guide*.

The reader should also be familiar with Google Guice.

Sections in this Guide

The following list describes the sections within the guide:

- **Registering a New Algorithm**
  The section describes how to add a new algorithm, which provides a new method of producing a random sample of cases for audit.
- **Utilizing Dynamic Selection Queries**
  The section describes how to use Dynamic Selection Queries to generate a list of cases for audit.
- **Configuring Selection Queries**
  The section describes the configuration options available for Selection Queries.
- **Case Audits Web Service**
  The section provides a brief overview on how case data from external sources can be used to generate a list of cases for audit.

1.2 Registering a New Algorithm

You must provide an implementation, which implements the SamplingStrategy interface to register a new algorithm. Use Guice bindings to map the algorithm to the correct algorithm implementation.

Overview

The sample algorithm that is provided with Cúram uses a starting point and an interval to determine the list of cases to be included in the case audit.

Creating a New Algorithm

An organization can add a new algorithm if the sample algorithm is not suitable, ensuring that a different method is used when a random sample of cases for audit is produced.

The following example outlines how to add an algorithm 'Every Nth Case', which adds every nth case to the list of cases to be included in the audit. N is a parameter that is specified by the audit coordinator. 1.3 Utilizing Dynamic Selection Queries on page 13 describes how to include algorithm parameters in case audit generation. The following sections describe in detail the steps required to create a new algorithm and add it to the application. The steps that are required:

- Administratively Define the New Algorithm
- Provide an Implementation for the Algorithm
- Add a Binding to the New Algorithm Implementation

Administratively Define the New Algorithm

Add a custom entry to CT_SamplingStrategy.ctx called Every Nth Case.

```
<code
          default="false"
          java_identifier="EVERYNTHCASE"
          status="ENABLED"
          value="SAMPLEVALUE"
        >
          <locale
            language="en"
            sort_order="0"
          >
            <description>Every Nth Case</description>
            <annotation/>
          </locale>
        </code>
```

Figure 1: Defining the Algorithm

Provide an Implementation for the Algorithm

The next step that is required to register a new algorithm is to provide the implementation for the algorithm.

This implementation must implement the `SamplingStrategy` Interface. The `SamplingStrategy` Interface has one method `getRandomSample`. This method takes a list of case identifiers and applies a sampling strategy to the list to generate a random case sample for audit. It accepts three parameters:

- masterList - the main list of cases that the sample is to be created from
- sampleSize - the number of cases that are to be included in the sample
- properties - a map of algorithm or configuration parameters that can be used in the filtering process.

The `SamplingStrategy` Interface can be found in the package curam.samplingstrategy.impl

```
/*
 * Copyright 2011 Curam Software Ltd.
 * All rights reserved.
 *
 * This software is the confidential and
 * proprietary information of Curam Software, Ltd.
 * ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall
 * use it only in accordance with the terms of the
 * license agreement you entered into with Curam Software.
 */
package curam.samplingstrategy.impl;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import curam.util.exception.AppException;
import curam.util.exception.InformationalException;

public class EveryNthCase implements SamplingStrategy {

  public List<Long> getRandomSample(List<Long> masterList,
    int sampleSize, Map<String, Object> properties)
    throws AppException, InformationalException {

    List<Long> randomSampleList = new ArrayList<Long>();
    Integer n = (Integer) properties.get("n");
    int index = 0;

    if (n <= masterList.size()) {

      while (randomSampleList.size() < sampleSize) {
        if (index + n < masterList.size()) {
          index = index + n;

          // If the element has been returned already,
          // try the next element until one that hasn't
          // been returned is found
          while (randomSampleList.contains(
            masterList.get(index))) {
            if (index < masterList.size() - 1) {
              index++;
            } else {
                index = 0;
            }
          }

          randomSampleList.add(masterList.get(index));
        } else {

          // Run out of elements, loop back to the
          // start of the list
          int elementsToStartOfList = masterList.size() - index;
          index = n - elementsToStartOfList;

          // If the element has been returned already,
          // try the next element until one that hasn't
```

```
            // been returned is found
            while (randomSampleList.contains(
              masterList.get(index))) {
              if (index < masterList.size() - 1) {
                index++;
              } else {
                  index = 0;
              }
            }

            randomSampleList.add(masterList.get(index));
          }
        }
      }
      return randomSampleList;
    }
}
```

Figure 2: Algorithm Implementation

Add a Binding to the New Algorithm Implementation

Guice bindings are used to map the algorithm to the correct algorithm implementation.

```
/*
 * Copyright 2011 Curam Software Ltd.
 * All rights reserved.
 *
 * This software is the confidential and proprietary
 * information of Curam Software, Ltd.
 * ("Confidential Information").  You shall not
 * disclose such Confidential Information and shall
 * use it only in accordance with the terms of the
 * license agreement you entered into with Curam Software.
 */
package curam.samplingstrategy.impl;

import com.google.inject.AbstractModule;
import com.google.inject.multibindings.MapBinder;
import curam.codetable.impl.SAMPLINGSTRATEGYEntry;

/**
 * Guice module for binding Sampling Strategies.
 *
 */
public class Module extends AbstractModule {

  @Override
  public void configure() {

    // register sampling strategies
    final MapBinder<SAMPLINGSTRATEGYEntry, SamplingStrategy>
      samplingStrategies = MapBinder.newMapBinder(binder(),
        SAMPLINGSTRATEGYEntry.class, SamplingStrategy.class);

    samplingStrategies.addBinding(
      SAMPLINGSTRATEGYEntry.EVERYNTHCASE).to(EveryNthCase.class);
  }
```

```
}
```

Figure 3: Binding the Algorithm

The new algorithm is now ready to be associated with a Case Audit Configuration in the Administration Application. For more information on Case Audit Configuration, see the *Cúram Case Audits Business Guide*.

1.3 Utilizing Dynamic Selection Queries

You can generate a random sample of cases with dynamic selection queries. Dynamic selection queries require a UIM page that allows an audit coordinator to enter selection criteria values. The case audit query management API is a public API used to run selection queries.

What is a Dynamic Selection Query?

Dynamic Selection Queries are used to generate a random sample of cases and contain the selection criteria that are used to search for and produce the list of cases. They allow the audit coordinator to enter any combination of selection criteria to be used when a list of cases is produced.

Why use a Dynamic Selection Query?

Four sample queries are provided for each of the standard case types: Integrated Case, Benefit Product Delivery, Liability Product Delivery, and Investigation Case. If these queries do not contain sufficient criteria, custom selection queries can be created.

Important: Complex queries are not suited to Dynamic Selection Queries as they do not perform. If a complex query needs to be created consider by using standard practices that are followed in the application instead of using a Dynamic Selection Query. The sample queries are implemented in this way as they are complex.

Implementing a Dynamic Selection Query

For new dynamic selection queries a UIM page must be created so that an audit coordinator can enter values for the new selection criteria. Similarly, if a new algorithm that requires parameters is to be used, then this input screen must be developed.

See Example: Implementing a Dynamic Selection Query on page 15 for an example by using the 'Every Nth Case' algorithm. The following outlines the steps that are required to use a custom Dynamic Selection Query.

- Create a UIM that contains the required selection criteria. The name of this UIM must match the name that the systems administrator enters for the Random Generation page name when selection queries are configured.
- Any other required input screens must also be developed, such as screens for entering algorithm parameters, entering the number of cases to be returned and so on. If multiple

screens are used they should be developed as a wizard. For more information on wizards, consult the *Cúram Web Client Reference Manual*.

- Create the necessary struct to cater for the selection criteria.
- Create and implement a new facade method that is responsible for generating the list of cases for audit. Note that a Case Audit Query Management API is available to help generate this list of cases.
- The systems administrator must create, validate, and publish a new Selection Query with the SQL required to retrieve the data and the selection criteria that are associated with it. The selection query must then be associated with the relevant case audit configuration.

An example of these steps is provided in Example: Implementing a Dynamic Selection Query on page 15

The Case Audit Query Management API

The Case Audit Query Management API is a public API used to run selection queries. To use this API, a new Dynamic Selection Query must be created in the systems administration application.

The Selection Query contains the SQL that is required to perform the search. For example, if a new search that contains a status is required, a dynamic selection query must be entered that contains the selection criteria page name, along with the following SQL -

SELECT caseID INTO :caseID FROM CaseHeader WHERE statusCode = :statusCode

The Case Audit Query Management API contains one main method of interest runDynamicQueryCaseSearch. This method takes two arguments a selection query identifier and a map of selection criteria (as entered by the audit coordinator). The selection query is retrieved and the selection criteria that are entered by the audit coordinator are substituted into the SQL. The query is then run against the database and a list of CaseHeader records is returned.

```
/**
   * Executes a Case Audit dynamic selection query
   * and returns a list of case header records
   * that match the criteria specified in the
   * dynamic query.
   *
   * @param selectionQueryID the unique identifier of the
 dynamic
   *         selection query
   *
   * @param parameterMap Map of all parameters
   *        in name value pairs.
   * @return A list of Case Header records that satisfy the
   *         selection criteria
   * @throws AppException
   * @throws InformationalException
   */
    public List<CaseHeader> runDynamicQueryCaseSearch(
    final long selectionQueryID,
    final HashMap<String, String> parameterMap)
    throws AppException, InformationalException
```

Figure 4: The Case Audit Query Management API

Example: Implementing a Dynamic Selection Query

Step 1: Create a UIM that contains the required selection criteria.

This screen allows the audit coordinator to enter selection criteria that relate to the dynamic query.

```
<PAGE
  PAGE_ID="exampleSelectionCriteria"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file://Curam/UIMSchema.xsd"
>

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

  <SERVER_INTERFACE
    CLASS="ExampleFacade"
    NAME="ACTION"
    OPERATION="validateCustomCriteria"
    PHASE="ACTION"
  />

  <PAGE_PARAMETER NAME="auditPlanID"/>
  <PAGE_PARAMETER NAME="queryID"/>

  <ACTION_SET
    ALIGNMENT="CENTER"
    TOP="false"
  >
    <ACTION_CONTROL
      LABEL="ActionControl.Label.Cancel"
      ALIGNMENT="LEFT"/>

    <ACTION_CONTROL
      DEFAULT="true"
      IMAGE="NextButton"
      LABEL="ActionControl.Label.Next"
      TYPE="SUBMIT"
    >
      <LINK
        SAVE_LINK="false"
        DISMISS_MODAL="false"
        PAGE_ID="exampleSelectAmount"
      >
        <CONNECT>
          <SOURCE
            NAME="PAGE"
            PROPERTY="auditPlanID"
          />
          <TARGET
            NAME="PAGE"
```

```
            PROPERTY="auditPlanID"
          />
        </CONNECT>
        <CONNECT>
          <SOURCE
            NAME="ACTION"
            PROPERTY="result$status"
          />
          <TARGET
            NAME="PAGE"
            PROPERTY="status"
          />
        </CONNECT>
        <CONNECT>
          <SOURCE
            NAME="PAGE"
            PROPERTY="queryID"
          />
          <TARGET
            NAME="PAGE"
            PROPERTY="queryID"
          />
        </CONNECT>
      </LINK>
    </ACTION_CONTROL>
  </ACTION_SET>

  <CLUSTER LABEL_WIDTH="30">
    <FIELD
      LABEL="Field.Label.Status"
      USE_BLANK="true"
      USE_DEFAULT="false">
      <CONNECT>
        <TARGET
          NAME="ACTION"
          PROPERTY="key$status"
        />
      </CONNECT>
    </FIELD>
  </CLUSTER>
  </PAGE>
```

Figure 5: UIM to allow the audit coordinator to enter selection criteria

Step 2: If required, create a screen to allow the audit coordinator enter the number of cases to audit.

```
<PAGE
  PAGE_ID="exampleSelectAmount"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file://Curam/UIMSchema.xsd"
>

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

```
  </CONNECT>
</PAGE_TITLE>

<SERVER_INTERFACE
  CLASS="ExampleFacade"
  NAME="ACTION"
  OPERATION="validateNumberOfCases"
  PHASE="ACTION"
/>

<PAGE_PARAMETER NAME="auditPlanID"/>
<PAGE_PARAMETER NAME="status"/>
<PAGE_PARAMETER NAME="queryID"/>

<CLUSTER
  DESCRIPTION="Cluster.Description.Text"
  LABEL_WIDTH="30">

  <FIELD LABEL="Field.Label.Number">
    <CONNECT>
      <TARGET
        NAME="ACTION"
        PROPERTY="key$numberOfCases"
      />
    </CONNECT>
  </FIELD>
</CLUSTER>


<ACTION_SET
  TOP="false"
>
  <ACTION_CONTROL
  LABEL="ActionControl.Label.Cancel"
  ALIGNMENT="LEFT"/>
  <ACTION_CONTROL
    DEFAULT="true"
    IMAGE="NextButton"
    LABEL="ActionControl.Label.Next"
    TYPE="SUBMIT"
  >
    <LINK
      SAVE_LINK="false"
      DISMISS_MODAL="false"
      PAGE_ID="exampleConfigureAlgorithm"
    >
      <CONNECT>
        <SOURCE
          NAME="PAGE"
          PROPERTY="auditPlanID"
        />
        <TARGET
          NAME="PAGE"
          PROPERTY="auditPlanID"
        />
      </CONNECT>
      <CONNECT>
        <SOURCE
```

```
            NAME="PAGE"
            PROPERTY="status"
          />
          <TARGET
            NAME="PAGE"
            PROPERTY="status"
          />
        </CONNECT>
        <CONNECT>
          <SOURCE
            NAME="ACTION"
            PROPERTY="result$numberOfCases"
          />
          <TARGET
            NAME="PAGE"
            PROPERTY="numberOfCases"
          />
        </CONNECT>
        <CONNECT>
          <SOURCE
            NAME="PAGE"
            PROPERTY="queryID"
          />
          <TARGET
            NAME="PAGE"
            PROPERTY="queryID"
          />
        </CONNECT>
      </LINK>
    </ACTION_CONTROL>
  </ACTION_SET>
</PAGE>
```

Figure 6: UIM to allow the audit coordinator choose how much of the generated case sample to audit

If required, create a screen to allow the audit coordinator enter algorithm parameters.

```
<PAGE
  PAGE_ID="exampleConfigureAlgorithm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file://Curam/UIMSchema.xsd"
>

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

  <SERVER_INTERFACE
    CLASS="ExampleFacade"
    NAME="ACTION"
    OPERATION="generateExampleCaseList"
    PHASE="ACTION"
  />
```

```
<PAGE_PARAMETER NAME="auditPlanID"/>
<PAGE_PARAMETER NAME="status"/>
<PAGE_PARAMETER NAME="numberOfCases"/>
<PAGE_PARAMETER NAME="queryID"/>

<CONNECT>
  <SOURCE
    NAME="PAGE"
    PROPERTY="auditPlanID"
  />
  <TARGET
    NAME="ACTION"
    PROPERTY="key$auditPlanID"
  />
</CONNECT>
<CONNECT>
  <SOURCE
    NAME="PAGE"
    PROPERTY="status"
  />
  <TARGET
    NAME="ACTION"
    PROPERTY="key$status"
  />
</CONNECT>
<CONNECT>
  <SOURCE
    NAME="PAGE"
    PROPERTY="numberOfCases"
  />
  <TARGET
    NAME="ACTION"
    PROPERTY="key$numberOfCases"
  />
</CONNECT>
<CONNECT>
  <SOURCE
    NAME="PAGE"
    PROPERTY="queryID"
  />
  <TARGET
    NAME="ACTION"
    PROPERTY="key$selectionQueryID"
  />
</CONNECT>

<CLUSTER LABEL_WIDTH="30">
  <FIELD LABEL="Field.Label.Interval">
    <CONNECT>
      <TARGET
        NAME="ACTION"
        PROPERTY="key$interval"
      />
    </CONNECT>
  </FIELD>
</CLUSTER>
```

```
  <ACTION_SET
    TOP="false"
  >
    <ACTION_CONTROL
    LABEL="ActionControl.Label.Cancel"
    ALIGNMENT="LEFT"/>
    <ACTION_CONTROL
      IMAGE="FinishButton"
      LABEL="ActionControl.Label.Finish"
      TYPE="SUBMIT"
    />
  </ACTION_SET>
</PAGE>
```

Figure 7: UIM to allow the audit coordinator to specify configurable parameters for the algorithm

Step 3: Create the necessary struct to cater for the selection criteria.

This struct contains all selection criteria available for the selection query along with any other required parameters.

ExampleSelectionCriteria

- long auditPlanID
- long selectionQueryID
- int numberOfCases
- int interval
- String status

Step 4: Create and implement a new façade method that is responsible for generating the list of cases for audit.

This method uses the Case Audit Query Management API to run the dynamic query, by using the supplied selection criteria. This dynamic query returns the list of cases that match the selection criteria. The relevant algorithm is then started to filter the case list. Finally, case audit records are created for each case in the remaining list.

```
// Inject the map
  @Inject
  private Map<SAMPLINGSTRATEGYEntry, SamplingStrategy>
    samplingStrategies;

  /**
   * Generates the sample list of cases for audit based on the
   * supplied selection criteria. This method filters the list
   * using the algorithm associated with the case type for this
   * audit plan. The number of cases returned in the list is
 also
   * restricted by the number of cases specified by the user.
   *
   * @param key The selection criteria, selection query
   *            identifier, the audit plan identifier,
   *            the number of cases to generate and any
   *            algorithm parameters.
   *
   * @throws AppException
   * @throws InformationalException
   */
  public void generateExampleCaseList(
```

```
    ExampleSelectionCriteria key)
    throws AppException, InformationalException {

    AuditPlan auditPlan = auditPlanDAO.get(key.auditPlanID);

    CaseAuditQueryManagement caseAuditQueryManagement =
      new CaseAuditQueryManagement();

    // Add all selection criteria to the map
    HashMap<String, String> parameterMap =
      new HashMap<String, String>();
    parameterMap.put(":statusCode", key.status);

    // Call the Case Audit Query Management API
    // to run the selection query
    List<curam.piwrapper.caseheader.impl.CaseHeader> caseList =
      caseAuditQueryManagement.runDynamicQueryCaseSearch(
        key.selectionQueryID, parameterMap);

    // Get the algorithm/sampling strategy configured
    // for this case type
    final SamplingStrategy samplingStrategy =
      samplingStrategies.get(
        auditPlan.getAuditCaseConfig().getAuditAlgorithm());

    List<Long> caseIDList = new ArrayList<Long>();

    for (CaseHeader caseHeader : caseList) {
      caseIDList.add(caseHeader.getID());
    }

    // Set up the algorithm parameters
    Map<String, Object> params = new TreeMap<String, Object>();
    params.put("n", new Integer(key.interval));

    // Invoke algorithm to generate case sample,
    // passing in the list of cases,
    // the number of cases to return and the algorithm parameters
    List<Long> caseIDs = samplingStrategy.getRandomSample(
      caseIDList, key.numberOfCases, params);

    curam.core.facade.intf.CaseAudit caseAuditObj =
      curam.core.facade.fact.CaseAuditFactory.newInstance();

    // for each case, create a case audit
    for (int i = 0; i < caseIDs.size(); i++) {

      CaseAuditDetails caseAuditDetails = new CaseAuditDetails();
      caseAuditDetails.dtls.auditPlanID = key.auditPlanID;
      caseAuditDetails.dtls.caseID = caseIDs.get(i);
      caseAuditObj.createCaseAudit(caseAuditDetails);
    }

  }
```

Figure 8: Generating the list of cases for audit

Step 5: The systems administrator must create, validate, and publish a new Selection Query with the SQL required to retrieve the relevant data and the selection criteria that are associated with it.

Important: Appropriate database indexing is provided for any custom dynamic selection queries. Also, if a significant case load is expected to be returned from the selection query, it would be advisable to consider by using Deferred Processing to generate the random sample of cases. For more information on Deferred Processing, see the *Cúram Server Developer Guide.*

Using Dynamic Selection Queries for Manual Case Selection

Dynamic selection queries can also be used for manual case selection. A similar process to the one described above can be used. A new UIM must be created that contains the required selection criteria.

Note, this new UIM must be created because the resulting case list must be included in the page. The UIM allows the audit coordinator to manually select from the list of cases. The name of this UIM must match the name that the systems administrator who is entered for the Manual Search page name in the Selection Query configuration. The Case Audit Query Management API can again be used to run the query.

1.4 Configuring Selection Queries

You can configure selection queries. Two types of selection query exist: dynamic queries and fixed queries. Selection queries must be published in order to be associated with a case audit configuration.

Selection Queries are used to generate a sample of cases and contain the selection criteria that are used to search for and produce the list of cases.

Dynamic Selection Queries

A dynamic selection query, when configured for audit, presents a page that contains selection criteria. An audit coordinator must enter values for the criteria which returns a case sample. The audit coordinator can enter the parameters for one criterion, or has the flexibility to enter parameters for any logical combination of parameters.

For example, to return all open cases for males that start 1-6 January, the audit coordinator would enter values as follows: case status of open; case start date range of 1-6 January; and gender of male.

Fixed Selection Queries

A fixed selection query provides a predefined set of selection criteria that is defined through the entry of an SQL statement. The fixed selection query when created by a systems administrator contains the values for the selection criteria such as case status of open and so forth.

However, an audit coordinator has the option to select this fixed query when creating a case sample, no audit coordinator entry of selection criteria is required as they are already input as part of the query.

CAUTION: There is no OOTB front-end for the fixed selection query functionality. While fixed selection queries might be valuable to some customers, any customer that utilizes the functionality would need to consider the security trade-off that it brings. Responsibility falls to the customer to:

- construct the front-end page
- ensure that no malicious content can be passed in and executed
- ensure that the SQL stored in the database is correct

Creating a Selection Query

Creating a selection query is a two step process; the first step allows the addition of the basic selection query details, such as name and the query type, along with the SQL. The second step allows the entry of the selection criteria that are associated with the selection query.

The 'Name' of the selection query is what is displayed to the administrator when configuring a case audit and to the audit coordinator when generating a random case sample, so it has a meaningful and descriptive name.

'Query' represents the type of objects that the selection query impacts. In this initial release, there is one type of object, Case. More functionality is envisaged in this area so that selection queries can be captured for any object. An example usage of this might be participants, where an agency might want to poll all employers to determine information on employee working patterns.

The 'Query Type' should be chosen depending on the type of selection query that is required, Dynamic or Fixed. If the selection query is 'Dynamic', the 'Random Generation' and 'Manual Search' page names must be entered. The page names that are entered must match the name of the custom UIM that is used to enter the selection criteria.

The SQL text is the SQL statement that is used to run the selection query. Every field on the custom selection criteria selection screen is part of the WHERE clause. Take the example that is outlined in 1.3 Utilizing Dynamic Selection Queries on page 13. A UIM is created to allow the audit coordinator to select a status, the SQL necessary for this query would be -

SELECT caseID INTO :caseID FROM CaseHeader WHERE statusCode = :statusCode

Adding Selection Criteria

The next step is to enter the attribute names and values to be used in the SQL query. These can be added by using the 'Add Criteria' link on the second page of the wizard.

For dynamic selection queries the values is used only for validation purposes. Validation of a selection query will be discussed in the next section. For fixed selection queries the values entered is the actual values used when running the query as the audit coordinator does not

have the option to enter their own values. 'Name' contains the attribute as it appears in the WHERE clause of the SQL statement. The 'Display Name' and 'Display Value' is acceptable representations of 'Name' and 'Value' that can be displayed to an audit coordinator.

Validating a Selection Query

Once a selection query is created it can then be validated. The validation performed checks that the SQL query is valid and that the statement is attempting only to read data.

Note, database integrity must be maintained so the SQL statement does not modify or remove data. To minimize this risk, the SELECT and INTO clauses are defaulted. As validation errors can be quite complex, ensure that the SQL query that is provided adheres to these guidelines.

Publishing a Selection Query

A selection query must be published to make it available to administrators to associate with a case audit configuration. This ensures that a query is validated before being made available for use to an administrator.

Finally, before the selection query can be used by an audit coordinator as part of an audit plan, it needs to be associated to a case audit configuration by the administrator.

1.5 Case Audits Web Service

A Case Audits Web Service is provided to allow case data from external sources to be audited. The Web Service can be invoked on by any capable Web Service client, including tools such as the Business Intelligence and Reporting Tools (BIRT) reporting tool.

The Web Service receives a list of case identifiers to be used in an audit along with an associated name to identify the data set, from an external source. This data is stored in the ExternalCaseAuditData and ExternalCaseAuditDataItem tables, ready to be included in an audit plan. The audit coordinator can select case data from this source. If any of the cases do not exist on the system, no data is saved and a response indicating an error has occurred is returned. The Case Audit Web Service can be found at curam.core.ws.convert.bs.impl.ExternalCaseAuditData

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.