

IBM Cúram Social Program Management
Version 7.0.0

Cúram Evidence Generator Cookbook



Note

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

Edition

This edition applies to IBM Cúram Social Program Management v7.0.0 and to all subsequent releases unless otherwise indicated in new editions.

Licensed Materials - Property of IBM.

© **Copyright IBM Corporation 2012, 2016.**

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

© Cúram Software Limited. 2011. All rights reserved.

Contents

Figures	v
--------------------------	----------

Tables	vii
-------------------------	------------

Developing with the Evidence Generator 1

Introduction	1
Purpose	1
Content Summary	1
Intended Audience	2
Prerequisites	2
Quick Overview	2
Introduction	2
Sample Component	2
Inputs and Outputs	2
Generator Inputs	2
Generator Outputs	3
Configure an Existing Product	3
Introduction	3
Step 1: Create an Evidence Directory	3
Step 2: Create and Configure Evidence Properties File	3
Step 3: Create General Properties File (general.properties)	4
Step 4: Create Product Employment Properties File (employment.properties)	4
Step 5: Configure the Module.	5
Asset Example.	5
Introduction	5
Step 1: Model Evidence Entity	5
Asset Entity and Aggregations	5
Asset Modeling Diagram	6
Step 2: Create Evidence Metadata	6
Asset Server XML	6
Asset Client EUIM	7
Step 3: Standard Evidence Configuration	8
Executing the Generator	8
Introduction	8
Generator Targets.	8
Standard Generator Targets	8
Evidence Generator Specific Targets	9
Generator Output.	9
Asset Handcrafted Code	10

Introduction	10
Asset Hook getDetailsForListDisplay	10
Customizing a Product	11
Introduction	11
Setting up a Custom Product	11
Custom evidence properties	11
Overview of Build Process and Generated Files.	12
Overriding Display Text	12
Overriding an OOTB Evidence Entity.	12
Modeling	12
Metadata	13
What is Generated	14
Adding a Brand New Custom Entity	15
What is Generated	15
Identifying Entities, Patterns and Relationships	15
Identifying Evidence Entities	15
Identifying Patterns.	15
Identifying Relationships	15
Parent-Child Relationship	15
Pre-Associated Relationship	16
Multiple Mandatory Parents Relationship	16
Related Relationship	16
Identifying Problems	16
Introduction	16
Configuration (Evidence Properties)	16
Generation Errors	16
Runtime Errors	18
Model	18
Generation Errors	18
Compilation Errors.	19
Metatypes	21
Incorrect Participant Metatype	21
Incorrect Date Metatype	22
Incorrect Comments Metatype	22
Properties	22
Generation Errors	22

Notices 25

Privacy Policy considerations	26
Trademarks	27

Figures

- | | | | | | |
|----|--------------------------------|---|----|---|----|
| 1. | Asset entity diagram | 6 | 2. | Sample custom evidence.properties | 11 |
|----|--------------------------------|---|----|---|----|

Tables

Developing with the Evidence Generator

Use the evidence generator as part of the standard Cúram build targets to dynamically create evidence entities that are based on certain criteria that are set for the evidence types. The evidence generator caters for all of the high level, repeatable evidence patterns across a number of large evidence-based solutions.

Introduction

Purpose

This document is intended to be a practical handbook for developers who are using the Cúram Evidence Generator. The document gives examples of how to use the generator to implement evidence, and provides troubleshooting tips. It is intended that the document be used in conjunction with the Cúram Evidence Generator Modeling Guide and Cúram Evidence Generator Specification document.

Content Summary

This guide presents all aspects of evidence generation from entity development to screen options. It starts with a quick overview of the generator.

Following on from this, Chapter 3 works through the required steps in configuring an existing product to use the evidence generator and Chapter 4 gives a detailed example of implementing an evidence type from start to finish using the evidence generator.

The execution of the generator is described in chapter 5. This goes through the targets that will be used by developers when generating / building evidence.

After execution of the generator, a certain amount of handcrafted code needs to be written by a custom developer. Example code is shown from in Chapter 6 relating to the earlier example.

If a solution is built in-house using the generator, and customers want to override any aspect of it, it must support customizations of the entity itself, the server functionality and the client side pages. How they go about doing this is covered in the chapter 'Customizing a Product'.

Chapter 8 looks at identifying patterns and relationships that fit into the generator as well as describing the metadata that needs to be configured to support these patterns on both the server and client side.

In order to aid developers to track down any obvious mistakes that can be made, a chapter has been dedicated to identifying and resolving problems. This details common errors that can occur due to the non-setting or non-configuration of properties and the resolution to each.

It is hoped that this cookbook will not only prove a useful guide to developers to get them up and running with evidence generation but also a useful reference for finding resolutions to common problems.

Intended Audience

This document is aimed at those intending to use the generator to develop evidence based solutions using the Cúram Enterprise Framework.

Prerequisites

To make best use of this guide, you should have a good knowledge of XML, data modeling and evidence solutions. Ideally, readers should be familiar with the Designing Cúram Evidence Solutions Guide and Cúram Evidence Developers Guide before embarking upon the exercise of evidence generation.

Quick Overview

Introduction

The evidence generator caters for all of the high level, repeatable patterns that have been identified across a number of large evidence based solutions provided by the application. These patterns are outlined in detail in the Cúram Evidence Generator Specification guide. Custom solutions may identify patterns not catered for by the generator. In these instances it will be necessary for the solution to develop the entities manually, i.e. outside of the generator. It is believed that such patterns are in the minority.

The evidence generator is run as part of the standard Cúram build targets. It iterates through every evidence folder under each component. It initially looks for one specific file, `evidence.properties`. This should define the paths to a number of files and folders required during generation. If `evidence.properties` does not exist, the generator moves onto the next folder.

Sample Component

A sample directory of the finished component will have the following details:.

1. A model directory, as usual, which would contain any model files used for the evidence entity modelling.
2. An 'evidence' directory containing 'evidence.properties'.
3. The 'evidence.properties' would then define locations for:
 - Any server, evidence metadata
 - Any integrated case, client, evidence metadata
 - Any product delivery, client, evidence metadata
 - The required properties files for common client display text

Inputs and Outputs

Generator Inputs

The following are the list of resources used by the generator as input data.

evidence.properties

This is a resource for configuring the Evidence Generator and contains all the product/component specific properties such as naming conventions, directory locations and product-wide settings. Some of these properties are also passed onto the generation itself. These properties are defined once per product.

general.properties and employment.properties

These resources are for generating the client screens. They contain generic

text labels that are used on many client screens as well as descriptions of these fields are used in the application online help. These properties are defined once per product.

Server Metadata File(e.g. Expenses.xml)

This defines your entities name and it's relationships to other evidence entities.

Client Metadata File(e.g. Expenses.euim)

This defines the client screens used to maintain your evidence entity.

Client Properties File(e.g. Expenses.properties)

This is required by your EUIM file and defines the text labels that are used, as well as descriptions of these fields used for the application online help.

and a modeled entity.

Generator Outputs

The following is the list of outputs generated by the generator.

1. Facade and Service Layer Model
2. Java Code
3. Client UIM/VIM
4. Wizard Data APPRESOURCE.dmx
5. Tab Configurations

Configure an Existing Product

Introduction

This chapter will summarize the steps involved in configuring an existing product to use the Evidence Generator. Completing these steps will mean the product will be ready for its first generatable evidence implementations.

1. Create an Evidence Directory
2. Create and Configure Evidence Properties File
3. Create General Properties File (general.properties)
4. Create Product Employment Properties File (employment.properties)
5. Configure the Module

Step 1: Create an Evidence Directory

Create directory evidence under product root directory in EJBServer. For the example, SampleEGProduct will be used as the Product name. Therefore the evidence directory would be EJBServer/components/SampleEGProduct/evidence.

Step 2: Create and Configure Evidence Properties File

Create an evidence.properties file. This file is used to configure various mandatory product parameters including locations of input files such as EUIMs and locations of output files such as generated UIMs.

Note: The location of the evidence.properties is important. This *must* be located within a directory called evidence, however this directory may be located anywhere within your component. For convenience the following location is suggested:

EJBServer/components/
SampleEGProduct/evidence/evidence.properties

The developer is then free to specify within this properties file the location of the remaining mandatory files in arbitrary locations. Again, for convenience, sub-directories under the evidence directory would be the logical choice.

The following is a sample of the product parameters required. For the complete list see *Cúram Evidence Generator Specification - Appendix A evidence.properties*

product.name=SampleEGProduct

This setting would result in the generated evidence files being copied to ../components/SampleEGProduct.

product.ejb.package=seg

Following on from the product name in the previous example, the code package name here could be seg, so generated classes would have a package structure like curam.seg.evidence

It should be noted that setting this property to evidence will result in a generated package structure of curam.evidence and not curam.evidence.evidence

product.prefix=SEG

The prefix is prepended to the name of all generated UIM pages and certain generated classes, such as the façade. In this case the façade class generated would be SEGEvidenceMaintenance

product.webclient=\${webclient.dir}/components/\${product.name}

The root directory for client product is located at webclient/components/SampleEGProduct.

Note: \${webclient.dir} is a property that is set in the Evidence Generator itself and points to the webclient/components directory. The user is free to use it or not.

Step 3: Create General Properties File (general.properties)

This file contains all generic client page properties, client message properties and online help properties for this product.

All of the keys (properties) specified in *Cúram Evidence Generator Specification - Appendix B general.properties* are mandatory. Omission of any keys is likely to break the build or cause compilation errors.

Step 4: Create Product Employment Properties File (employment.properties)

This file contains all generic employment specific client page properties, client message properties and online help properties for this product.

As with the general.properties, all of the keys (properties) specified in *Cúram Evidence Generator Specification - Appendix C employment.properties* are mandatory. Omission of any keys is likely to break the build or cause compilation errors.

Step 5: Configure the Module

The evidence generator will generate a single registrar module for all the generated evidence types. This registers the implementations of the evidence interface and the evidence comparison interface. Add the fully qualified class name to the Module Class Name initial data.

The class generated in our example would be

- `curam.seg.evidence.service.impl.SEGRegistrarModule`

Asset Example

Introduction

This chapter outlines how to implement a sample evidence type, Asset, as generated evidence. This covers the metadata and the modeling required to successfully generate the server-side and client-side artefacts for the evidence entity.

Step 1: Model Evidence Entity

This part of the process is totally removed from the evidence generator. The evidence entity is modeled in the standard way following the guidelines in Cúram Evidence Generator Modeling Guide and will be picked up in the standard Cúram build. The modeling of entities, structs and aggregations is well described in that guide, and the standards, naming conventions etc. described must be adhered to. All the metadata defined is used to support and connect to this via the service layer, façade layer or client.

Asset Entity and Aggregations

The Asset entity has the following attributes:

- Value
- Asset Type
- Start Date
- End Date

It has been decided that the screens for maintaining the Expense entity should display the employer of the case participant the record is associated with. This information is not stored on the Expense entity, it is only displayed on the screens where it is deemed useful to the caseworker as they maintain the Expense information.

With this information it has been decided that the Expense entity should have the attributes listed below (with their associated domain definition):

The primary key of the entity must be called 'evidenceID' as the generator expects this. All other attributes may be named as required. Optimistic locking is enabled on the entity. The entity should have the standard read, insert and modify operations generated automatically.

There are a number of conventions outlined in the Cúram Evidence Generator Modeling Guide that must be adhered to. These include the naming of structs and aggregations required for each entity, as well as multiplicities for the aggregations and code packages matching the `product.ejb.package` property.

Additional Modeling: As mentioned earlier, it is necessary for the employment name to be displayed on the maintenance screens for the Expense entity. A 'placeholder' for passing this information from the system to the screen is required. This placeholder is created using a RelatedEntityAttribute struct. This is simply an ordinary struct with a specific naming convention and aggregation. These conventions are outlined in the Cúram Evidence Generator Modeling Guide.

In the example above, the new struct, ExpenseRelatedEntityAttribute, is created with one attribute, employerName. The ReadExpenseEvidenceDetails struct must aggregate the ExpenseRelatedEntityAttribute struct. The multiplicity must be 1:1 and the aggregation must be named relatedEntityAttributes.

Asset Modeling Diagram

Once the entity's attributes have been defined and the necessary structs and aggregations modeled, it's possible to put it all together. Our Asset entity is modeled in Fig 4.1 below:

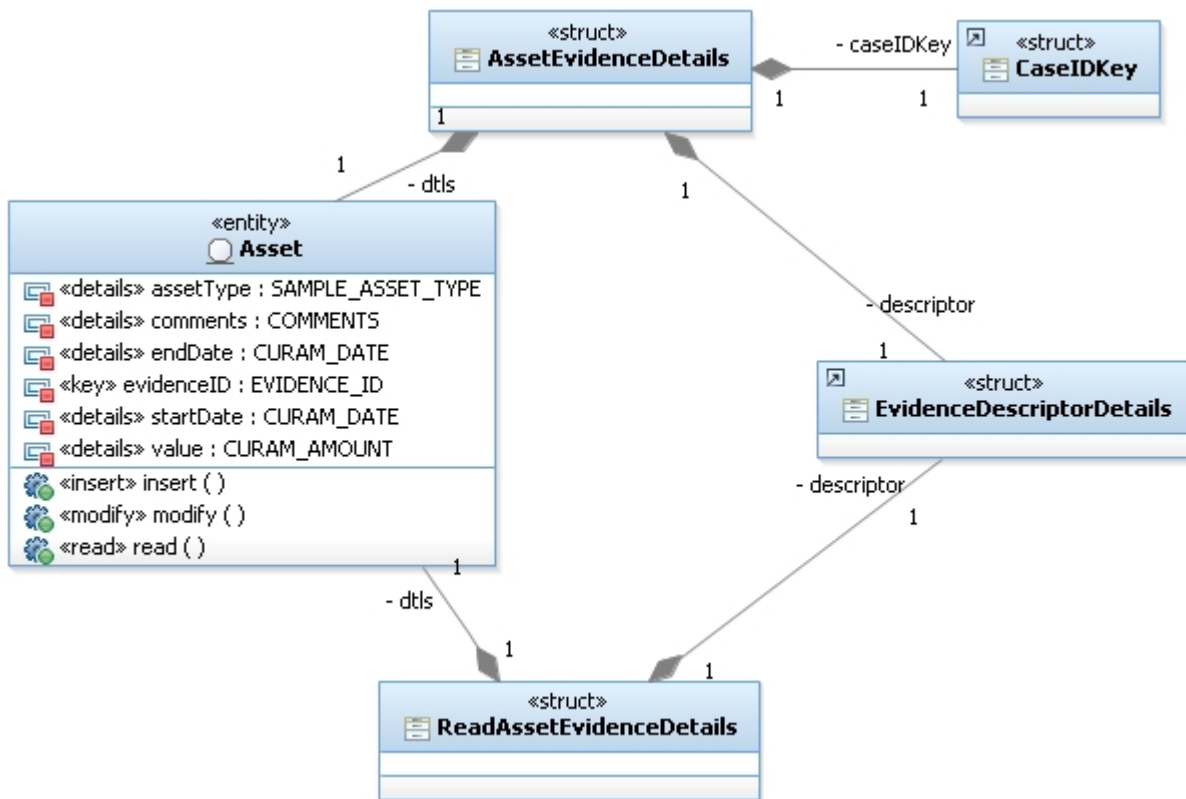


Figure 1. Asset entity diagram

Step 2: Create Evidence Metadata

The evidence generator has been configured to look in the configured directories for server XML metadata files, Integrated Case EUIM metadata files and their corresponding properties files and Product Delivery EUIM metadata files and their corresponding properties files. Each entity will have one server XML file and one pair of EUIM and properties files defining it.

Asset Server XML

The following is the server XML metadata file for Asset:

```

<EvidenceEntity>
  <Entity
    logicalName="Asset"
  >

```

```

        relateEvidenceParticipantID=""
    >
    <RelatedEntityAttributes
        exposeOperation="No"
        relatedEntityAttributes="No"
    />
    <Relationships/>
    <BusinessDates
        startDate="startDate"
        endDate="endDate"
    />
</Entity>
</EvidenceEntity>

```

Asset Client EUIM

The following is the client EUIM metadata file for Asset:

```

<Entity name="Asset" displayName="Asset">
  <UserInterface>
    <Clusters>
      <Cluster label="Cluster.Title.AssetDetails"
        numCols="2">
        <Field label="Field.Label.AssetType"
          columnName="assetType" mandatory="Yes"
          use_blank="true"/>
        <Field label="Field.Label.StartDate"
          columnName="startDate" mandatory="No"
          use_default="false"/>
        <Field label="Field.Label.AssetValue"
          columnName="value" mandatory="Yes"
          use_default="false"/>
        <Field label="Field.Label.EndDate"
          columnName="endDate" mandatory="No"
          use_default="false"/>
      </Cluster>
      <Cluster label="Cluster.Title.Comments">
        <Field columnName="comments" mandatory="No"
          metatype="COMMENTS" label="" />
      </Cluster>
    </Clusters>
  </UserInterface>
</Entity>

```

Note: EUIM is similar in nature to UIM. For example, data is described in terms of 'fields' and the layout is described in terms of 'labels', 'clusters' and 'fields'. The idea behind the introduction of 'EUIM' (Evidence UIM) was to use a format that developers would be familiar with.

The following is the associated properties file for Asset.euim:

```

Cluster.Title.AssetDetails=Asset Details

Field.Label.AssetType=Type
Field.Label.AssetType.Help=The type of the asset

Field.Label.AssetValue=Value
Field.Label.AssetValue.Help=The value of the asset

Field.Label.StartDate=Received
Field.Label.StartDate.Help=The date the asset was received

Field.Label.EndDate=Disposed
Field.Label.EndDate.Help=The date the asset was disposed

Cluster.Title.Comments=Comments
Cluster.Title.Comments.Help=Additional information

```

Step 3: Standard Evidence Configuration

There are a number of steps involved when configuring a new evidence type. Here is a checklist.

Configuration before generating Asset

- Name the Asset evidence type by adding an entry to Evidence Type Code Table.
- Optionally, create a static description for Asset evidence via a new entry in the Text Translation initial data. Link this Text Translation to a new entry in the Localizable Text initial data. This step can be deferred until later. It's only visible to the user on the New Evidence screen.
- Add a new entry in the Evidence Metadata initial data linking it to the Evidence Type and, optionally for now, the Localizable Text.
- Link the Evidence Metadata to either an integrated case or a product by adding an entry to the Admin IC Evidence Link or the Product Evidence Link initial data respectively. If the evidence is to belong to an Evidence Category (e.g. Resources) set the category attribute here.
- If the Asset Evidence Business Object Tab is to be utilised in a given Section of the application, contribute to the section definition (e.g. DefaultAppSection.sec). Without this contribution the Asset Evidence Business Object page will simply load in the current content panel.

A sample section file is generated for each product including all the evidence tabs. This sample is located at EJBServer/components/EvGen/tab/BusinessObjectTab/<product.prefix>GeneratedAppSection.sec

```
<sc:section
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sc=
    "http://www.curamsoftware.com/curam/util/client/section-config"
  id="DefaultAppSection"
>
  <sc:tab id="AssetObject"/>
</sc:section>
```

Note: There is one handcrafted implementation that **must** be completed after running the generator, as without it, some evidence screens will not be accessible. See “Asset Hook getDetailsForListDisplay” on page 10.

Executing the Generator

Introduction

This chapter looks at the evidence generator targets that the user can call on and the expected outcome.

Generator Targets

Standard Generator Targets

The evidence generator has been designed to be transparent to the developer, i.e. it integrates into the standard build targets so no additional environment variables need to be set nor do any new targets need to be called to generate evidence. Once the steps outlined in the preceding chapters have been fulfilled, the standard build targets will suffice to generate or clean the metadata driven evidence along with the standard files.

build generated: Calling this target in EJBServer will generate the evidence inf and impl layers as well as the normal server layers.

build client: Calling this target in webclient will generate and build the client screens as well as the standard client screens.

Note: As is the case with a normal build, if the build client is called prior to a build generated after model / metadata changes have been made, the client build can fail. Normally the failure would be the result of the changes made in the client UIMs / VIMs to use new features implemented on the server and then rebuilding the client without first rebuilding the server.

With evidence generation, any change made to the EUIMs / server XMLs will be automatically generated the next time the client generation occurs. This is why it is important to generate the server first if those changes affect the façade layer in any way.

build clean: The target to clean generated evidence is incorporated into the standard target and is therefore transparent to the developer. The target here is the same on both the server and the client, build clean.

Note: Customized generated code (see “Asset Handcrafted Code” on page 10) will be not be deleted.

Evidence Generator Specific Targets

As well as incorporating the evidence generator build targets into the standard targets, there are two optional targets the developer may use to provide more granular control over evidence generation. These are intended for cleaning of generated evidence without removing any of the standard generated files, thereby speeding up the development process. Both of these targets exist within the EJBServer/build.xml.

build egtools.clean: This target will clean all server related evidence files. A clean will only occur if the prerequisites have been met and will occur regardless of new EUIM and server XML files being added, or existing ones having been updated since the last build.

build egtools.client.clean: This target will clean all client related evidence files. A clean will only occur if the prerequisites have been met and will occur regardless of new EUIM and server XML files being added, or existing ones having been updated since the last build.

Generator Output

On completion of evidence generation, there will be a number of new directories in the locations specified in the evidence.properties file.

Note: All entity, service and façade level generated code is written directly to the build directory and so will not appear within your components source directory

Asset Handcrafted Code

Introduction

The generated server output contains some stubs in which custom code may be written, i.e. placeholders for customers to add their own code. These add a degree of flexibility when a generated evidence pattern is not a perfect fit for an evidence entity.

See *Cúram Evidence Generator Specification - Chapter 6, Adding Functionality* for the full list of generated stub classes and their methods.

Asset Hook `getDetailsForListDisplay`

All evidence entities **must** implement the Hook method, `getDetailsForListDisplay`. This method creates the text description for a particular Asset Business Object on the evidence workspace pages. As this is link text used on the client screens, it must be populated in order to access all the screens.

The following is the `getDetailsForListDisplay` implementation for Asset:

```
// _____  
/**  
 * Get evidence details for the list display  
 *  
 * @param key Key containing the evidenceID and evidenceType  
 *  
 * @return Evidence details to be displayed on the list page  
 */  
public EIFieldsForListDisplayDtls getDetailsForListDisplay(  
    EIEvidenceKey key)  
    throws AppException, InformationalException {  
  
    // Return object  
    EIFieldsForListDisplayDtls eiFieldsForListDisplayDtls =  
        new EIFieldsForListDisplayDtls();  
  
    // Asset entity key  
    final AssetKey assetKey = new AssetKey();  
    assetKey.evidenceID = key.evidenceID;  
  
    // Read the Asset entity to get display details  
    final AssetDtls assetDtls =  
        AssetFactory.newInstance().read(assetKey);  
  
    // Set the start / end dates  
    eiFieldsForListDisplayDtls.startDate = assetDtls.startDate;  
    eiFieldsForListDisplayDtls.endDate = assetDtls.endDate;  
  
    LocalisableString summary = new LocalisableString(  
        BIZOBJDESCRIPTIONS.BIZ_OBJ_DESC_ASSET);  
  
    summary.arg(  
        CodeTable.getItem(SAMPLEASSETTYPE.TABLENAME,  
            assetDtls.assetType));  
  
    // Format the amount for display  
    TabDetailFormatter formatterObj =  
        TabDetailFormatterFactory.newInstance();  
    AmountDetail amount = new AmountDetail();  
    amount.amount = assetDtls.value;  
    summary.arg(formatterObj.formatCurrencyAmount(amount).amount);  
  
    eiFieldsForListDisplayDtls.summary =
```

```

        summary.toClientFormattedText();
    return eiFieldsForListDisplayDtIs;
}

```

Customizing a Product

Introduction

An 'out-of-the-box'(OOTB) evidence solution is provided with some of the Cúram solutions, the customer may decide to extend and customize this OOTB evidence solution to better fit their business requirements.

Setting up a Custom Product

This section deals with setting up a custom product which will override an OOTB product.

Custom evidence properties

The OOTB product will come pre-configured with an evidence.properties file - see "Step 2: Create and Configure Evidence Properties File" on page 3 for details. For the purposes of overriding an OOTB product, the custom product will require its own, thin version of evidence.properties.

The crucial property here is 'override.product'. If this is not set, the evidence generator will assume that this evidence product is brand new. If it is set, it must be set to the 'product.name' of an existing evidence generation product. All the other properties have already been defined in *Cúram Evidence Generator Specification - Appendix A evidence.properties*.

```

# Unique name (product.name) of the OOTB product to override
override.product=SampleEGProduct

# Prefix used to specify where all metadata files are copied to
product.prefix=SEG

# Other Mandatory Properties in an Overriding Product

product.build.option=true

evidence.properties.dir
    = %SERVER_DIR%/components/custom/EvGenComponents/SEG/evidence

properties.home=${evidence.properties.dir}/properties/

server.metadata=${evidence.properties.dir}/server/metadata

caseType.integratedCase.metadata
    = ${evidence.properties.dir}/integrated/metadata

caseType.product.metadata
    = ${evidence.properties.dir}/product/metadata

```

Figure 2. Sample custom evidence.properties

Note: The location of the evidence.properties is not optional. This *must* be located in a directory named 'evidence' which resides in any subdirectory of:

EJBServer/components/custom

As the custom directory may eventually contain many of these overridden products (and evidence directories), it is recommended that some sort of naming scheme be devised. e.g.

```
EJBServer/components/custom  
/EvGenComponents/<ProductName>/evidence
```

Overview of Build Process and Generated Files

The evidence generator build process looks for evidence subdirectories in all the components listed in the `SERVER_COMPONENT_ORDER`. For each of these the first step is to gather the product's metadata (and display properties) to the build directory. Next, a search across the custom directory is performed to find any evidence.properties that overrides the queued product. It is at this point in the build that any overriding gets done. This is done by gathering the customized metadata (and display properties) and *copying them over* (not merging them with) the queued product's metadata in the build directory. The product's evidence is then generated from this super-set of metadata.

It is worth noting that most artefacts generated by an OOTB product are not modifiable. Nor are they placed under source control. The only artefacts that are modifiable, are the handcrafted Java™ classes provided for customizable hook points called to throughout the non-modifiable generated codebase. These are only generated where they previously did not exist. Thereafter, they must be maintained under source control.

Therefore, by overwriting the metadata before the build, all the generated custom artefacts get generated as if they belonged to the OOTB product (i.e. to the OOTB product's directories). The only exception to this is these handcrafted classes which will be described in more detail later on.

Overriding Display Text

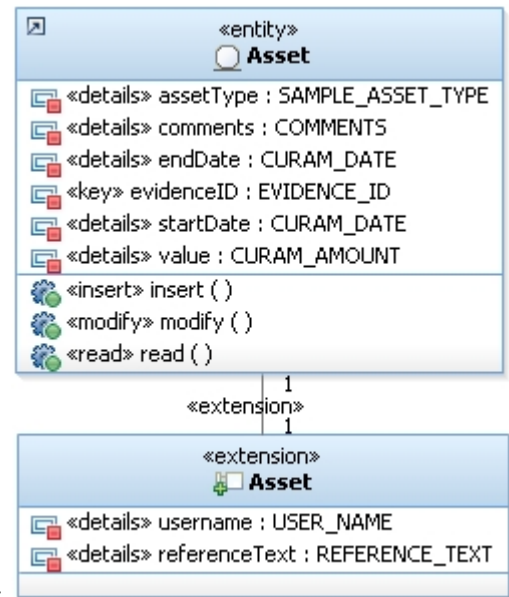
Display text is defined in the properties files associated with an EUIM, the general properties file and the employment properties file. These can all be overridden in the custom directory.

Overriding an OOTB Evidence Entity

For this example it is assumed that the Expense entity has been provided as part of a Cúram evidence solution. The customer has decided that the entity does not provide the fields required to fully meet the business requirements. To meet their requirements, the customer has decided to add two additional attributes to the entity. The first attribute is the username of the user who creates / modifies the record. The second is the number of children that the case participant has (if any).

Modeling

In keeping with the guidelines set out in the Cúram Evidence Generator Modeling Guide (Section 2.1.5) and the Cúram Server Modeling Guide (Section 30.2), an Extension class will be created and this class will be linked to the provided Expense entity.



The modeled extension is shown below:

Metadata

The metadata for a customized entity is almost identical to the standard metadata. It is captured in two files, <Entity-Name>.xml and <Entity-Name>.euim. To begin customization of an entity, it is recommended to copy all of the OOTB entity's metadata and make any modifications as appropriate.

Server-side Metadata: As the changes to Asset only relate to additional fields, the server-side metadata should be identical to the metadata of the entity that is being overridden with one exception. An additional node, `Override`, is required. This additional node specifies whether or not the entity is brand new and which custom handcrafted classes need to be generated. For more information, see the *Cûram Evidence Generator Specification - Appendix E.1.1 Override Node*.

The following is the custom server XML metadata file for Asset:

```

<EvidenceEntity>
  <Entity logicalName="Asset"
    relateEvidenceParticipantID="">
    <Override newEntity="No" customize="No" hook="Yes"
      relatedAttribute="No" validation="No" />
    <RelatedEntityAttributes exposeOperation="No"
      relatedEntityAttributes="No" />
    <Relationships/>
    <BusinessDates
      startDate="startDate"
      endDate="endDate"
    />
  </Entity>
</EvidenceEntity>
  
```

Client-side Metadata: The client-side metadata should be identical to the metadata of the entity that is being overridden except for the inclusion of any additional fields where appropriate. In the presented example, the developer will want to include the 'reference text' field on the screen as the user will be populating this themselves. The developer will not want to expose the username on the screen.

Note: Similar to any other customization within the application, it is not possible to remove any attributes from an entity.

The following is the custom client EUIM metadata file for Asset:

```
<Entity name="Asset" displayName="Asset">
  <UserInterface>
    <Clusters>
      <Cluster label="Cluster.Title.AssetDetails"
        numCols="2">
        <Field label="Field.Label.AssetType"
          columnName="assetType" mandatory="Yes"
          use_blank="true"/>
        <Field label="Field.Label.StartDate"
          columnName="startDate" mandatory="No"
          use_default="false"/>
        <Field label="Field.Label.ReferenceText"
          columnName="referenceText" mandatory="No"
          use_default="false"/>
        <Field label="Field.Label.AssetValue"
          columnName="value" mandatory="Yes"
          use_default="false"/>
        <Field label="Field.Label.EndDate"
          columnName="endDate" mandatory="No"
          use_default="false"/>
      </Cluster>
      <Cluster label="Cluster.Title.Comments">
        <Field columnName="comments" mandatory="No"
          metatype="COMMENTS" label=""/>
      </Cluster>
    </Clusters>
  </UserInterface>
</Entity>
```

The following is the associated properties file for Asset.euim:

```
Cluster.Title.AssetDetails=Asset Details
Field.Label.ReferenceText=Reference Name
Field.Label.ReferenceText.Help=Reference Name to help the user
differentiate similar records.
Field.Label.AssetType=Type
Field.Label.AssetType.Help=The type of the asset

Field.Label.AssetValue=Value
Field.Label.AssetValue.Help=The value of the asset

Field.Label.StartDate=Received
Field.Label.StartDate.Help=The date the asset was received

Field.Label.EndDate=Disposed
Field.Label.EndDate.Help=The date the asset was disposed

Cluster.Title.Comments=Comments
Cluster.Title.Comments.Help=Additional information
```

What is Generated

As stated earlier, everything except the handcrafted code is generated as it would be if the entity had been defined OOTB.

In the case of a custom extension of an OOTB entity, these handcrafted implementations will already exist. The generator creates handcrafted classes in the custom source package which are modeled using the replace superclass option, where the superclass is the existing OOTB implementation. These will only contain method stubs each of which will, by default, begin by calling the superclass implementation.

In the current example, the handcrafted preCreate function will need to be updated by the developer to assign the value of the username attribute to the creation

struct. Also, the handcrafted `validateDetails` function may need to be updated to ensure the 'reference text' field is not left blank.

Adding a Brand New Custom Entity

Adding a brand new custom entity to a custom evidence product that overrides an OOTB product is exactly the same as developing one in any other product with one exception. The `Override` node must be used and the `newEntity` attribute must be set to `Yes`.

Note: The codepath is the same as it would be in the OOTB product.

What is Generated

In this case, there will be no OOTB implementation of the handcrafted code. In order to keep the code as straightforward as possible, a dummy OOTB implementation of these is generated inside the build source directory underneath the OOTB's code package. Also, the derived custom version is generated into the custom source directory and should be placed under source control.

Identifying Entities, Patterns and Relationships

Identifying Evidence Entities

Evidence is the data collected by an organization to facilitate the delivery of services to the organization's clients. In the application, evidence is typically used in the determination of eligibility and entitlement for clients. For the evidence generator, it is any entity which implements the standard evidence interface and is maintained using the evidence solution.

Identifying Patterns

A pattern is a piece of functionality that is used by an evidence entity. This functionality might be features on a maintenance screen or additional code specific to an entity. The evidence generator provides the functionality to specify which patterns apply to which entities via metadata captured in XML. This metadata is then read, during generation, and converted to the appropriate feature, e.g. button on a client page or callout class stub in which a developer can then implement business logic.

Identifying Relationships

Relationships in evidence are ways to describe how evidence entities interact and exist in relation to each other. The generator provides the functionality to specify these relationships. It then generates all of the associated server-side code and client page functionality to facilitate the maintenance of the relationships. The most common relationship is the Parent-Child relationship. The other relationships the generator caters for are Pre-Associations, Multiple Mandatory Parents and the Related relationship.

Parent-Child Relationship

Parent-Child is one of the most common logical relationships between evidence entities. Typically this is a one-to-many relationship where the Parent can have many children and each Child must belong to a Parent. Parent-Child relationships should be used to capture the logical relationship between two entities where the Child entity cannot live without the Parent entity and the details on the Child are logically related to the details captured on the Parent. For example, student details may be held on a Student entity and student expenses on a Student Expenses

entity. In this instance, Student Expenses could not exist without Student being present but Student could exist on its own.

Pre-Associated Relationship

Pre-Associations are non-hierarchical relationships between evidence entities which can exist independently of each other. The association is known before creating the evidence. This means the developer can access data from the associated entity at create time.

Multiple Mandatory Parents Relationship

Where an entity must simultaneously be the child of more than one parent entity, the pattern for multiple mandatory parents should be used.

Related Relationship

Related relationships are again non-hierarchical in nature and are used to relate an evidence record to a non-evidence record. The primary example of this, which has occurred in all evidence based modules built by the application, is the relation of evidence based employment records to the Core Employment record. Examples of evidence based employment entities would be Self Employment and Paid Employment. This is a key functional area in a lot of solutions hence the decision to treat it as a separate pattern.

Identifying Problems

Introduction

When running the Evidence Generator, problems can occur if any of the set-up steps outlined in the preceding chapters are carried out incorrectly.

Configuration (Evidence Properties)

Generation Errors

Evidence will not build / clean:

Symptom:

No new evidence is generated when the target is generated. No evidence is deleted when the target is clean.

Cause: product.build.option is set to false or missing.

Solution:

Set product.build.option=true if this evidence is to be generated. If it is missing from the evidence.properties, add it with this value.

Evidence not found.:

Symptom:

Error when build generated is called on EJBServer...\CEF-Core\EJBServer\components\<\$product.name>\Evidence not found.

Cause: The property product.name in evidence.properties does not match that in the codebase.

Solution:

Set product.name=correct Product Name as it appears under EJBServer/components/ <ProductName>

'<EntityName>Details' is not present in the model:

Symptom:

Error when build generated is called from EJBServer: Parameter 'dtls' (of operation...) has type <EntityName> EvidenceDetails', but '<EntityName>Details' is not present in the model.

Cause: The property product.ejb.package in evidence.properties does not match part of the CODE_PACKAGE on the model.

Solution:

Set product.ejb.package=Model CODE_PACKAGE up to first "." delimiter.
For example

```
CODE_PACKAGE = seg.evidence.entity
```

```
product.ejb.package=seg
```

No source files matching the extensions XML:

Symptom:

Error when build generated is called displayed in the XML Digester output: The source location <\$server.metadata> was found to contain no source files matching the extensions XML

Cause: server.metadata does not match physical root directory for Product's evidence directory

Solution:

Set server.metadata to point to the correct directory.

The general properties file was not found:

Symptom:

Error when build generated is called displayed in the XML Digester output: The general properties file was not found at the location \$properties.home\

Cause: properties.home does not match physical properties directory.

Solution:

Set properties.home=Directory where general.properties was created.

<\$server.metadata> was found to contain no source files matching the extensions XML:

Symptom 1:

Error when build generated is called on EJBServer: Error # The source location <\$server.metadata> was found to contain no source files matching the extensions XML

Cause 1:

The property server.metadata in evidence.properties does not point to the location of server XML files.

Solution 1:

Set server.metadata=<correct location of server metadata>;

Symptom 2:

Error when build client is called on Webclient: Error # The source location <\$server.metadata> was found to contain no source files matching the extensions XML

Cause 2:

The property product.name in evidence.properties does not match that in the codebase.

Solution 2:

Set product.name=correct Product Name as it appears under EJBServer/components/ <ProductName>

No EUIM source files:**Symptom 1:**

Error when build generated is called on EJBServer: No EUIM source files were found within the EUIM source directory
<\$caseType.integratedCase.metadata>

Cause 1:

The property caseType.integratedCase.metadata in evidence.properties does not point to the location of integrated EUIM files.

Solution 1:

Set caseType.integratedCase.metadata=<correct location of integrated metadata>

Symptom 2:

Error when build generated is called on EJBServer: No EUIM source files were found within the EUIM source directory
<\$caseType.product.metadata>

Cause 2:

The property caseType.product.metadata in evidence.properties does not point to the location of product EUIM files.

Solution:

Set caseType.product.metadata=<correct location of product metadata>

Runtime Errors**HTTP Status 404 Error Message:****Symptom:**

Page not found error when trying to access generated evidence workspace.

Cause: product.codetable is set incorrectly, i.e. not pointing at product codetable directory.

Solution:

Set product.codetable=<product_Root_CodeTable_directory>.

Model**Generation Errors****Invalid Mandatory Field:****Symptom:**

Error when build generated is called from EJBServer: The mandatory field 'dtls.<fieldName>' specified for parameter 'dtls' of operation '<EntityName>.create<EntityName>Evidence' is invalid.

Cause: The "dtls" association between the <EvidenceEntity>Details struct and the EvidenceEntity entity is missing. This association is mandatory for all evidence entities.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

Compilation Errors**<EntityName>Details' is not present in the model.:****Symptom:**

Error when build generated is called: <EntityName>Details' is not present in the model.

Cause 1:

The first element (i.e. up to first delimiter ".") in CODE_PACKAGE does not match evidence property product.ejb.package in evidence.properties.

Cause 2:

The second and third elements in CODE_PACKAGE are not evidence.entity.

Solution 1:

Set first part of CODE_PACKAGE=product.ejb.package (or vice versa)

Solution 2:

Set second part of CODE_PACKAGE=evidence. Set third part of CODE_PACKAGE=entity.

details.parEvKey cannot be resolved or is not a field:**Symptom:**

Compilation error in generated code: details.parEvKey cannot be resolved or is not a field

Cause: The "parEvKey" association between the <EvidenceEntity>Details struct and the EvidenceKey struct is missing. This association is necessary if the evidence entity in question is a child of another evidence entity.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

evidenceDetails.parEvKey cannot be resolved or is not a field:**Symptom:**

Compilation error in generated code: evidenceDetails.parEvKey cannot be resolved or is not a field

Cause: This error is also a result of the "parEvKey" association between the <EvidenceEntity>Details struct and the EvidenceKey struct missing and should be dealt with in the same manner.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

dtls.selectedParent cannot be resolved or is not a field:**Symptom:**

Compilation error in generated code: dtls.selectedParent cannot be resolved or is not a field

Cause: The "selectedParent" association between the <EvidenceEntity>Details struct and the ParentSelectDetails struct is missing. The ParentSelectDetails

is present and the association between this and the entity details struct is necessary if the entity is a child of another evidence entity.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

dtls.caseIDKey cannot be resolved or is not a field:

Symptom:

Compilation error in generated code: dtls.caseIDKey cannot be resolved or is not a field

Cause: The "caseIDKey" association between the <EvidenceEntity>Details struct and the CaseIDKey struct is missing. This association is mandatory for all evidence entities.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

evidenceDetails.caseIDKey cannot be resolved or is not a field:

Symptom:

Compilation error in generated code: evidenceDetails.caseIDKey cannot be resolved or is not a field

Cause: This error is also a result of the "caseIDKey" association between the <EvidenceEntity>Details struct and the EvidenceKey struct missing and should be dealt with in the same manner.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

readEvidenceDetails.descriptor cannot be resolved or is not a field:

Symptom:

Compilation error in generated code: readEvidenceDetails.descriptor cannot be resolved or is not a field

Cause: The "descriptor" association between the Read<EvidenceEntity>Details struct and the EvidenceDescriptorDetails struct is missing. This association is mandatory for all evidence entities.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

details.descriptor cannot be resolved or is not a field:

Symptom:

Compilation error in generated code: details.descriptor cannot be resolved or is not a field

Cause: The "descriptor" association between the <EvidenceEntity>Details struct and the EvidenceDescriptorDetails struct is missing. This association is mandatory for all evidence entities.

Solution:

Create an association as described between the two structs. See “Asset Entity and Aggregations” on page 5 for more details.

evidenceDetails.descriptor cannot be resolved or is not a field:

Symptom:

Compilation error in generated code: evidenceDetails.descriptor cannot be resolved or is not a field

Cause: This error is also a result of the "descriptor" association between the <EvidenceEntity>Details struct and the EvidenceDescriptorDetails struct missing and should be dealt with in the same manner.

Solution:

Create an association as described between the two structs. See "Asset Entity and Aggregations" on page 5 for more details.

readEvidenceDetails.dtls cannot be resolved or is not a field:

Symptom:

Compilation error in generated code: readEvidenceDetails.dtls cannot be resolved or is not a field

Cause: The "dtls" association between the Read<EvidenceEntity>Details struct and the <EvidenceEntity> entity is missing. This association is mandatory for all evidence entities.

Solution:

Create an association as described between the two structs. See "Asset Entity and Aggregations" on page 5 for more details.

readEvidenceDetails.caseParticipantDetails cannot be resolved or is not a field:

Symptom:

Compilation error in generated code:
readEvidenceDetails.caseParticipantDetails cannot be resolved or is not a field

Cause: The "caseParticipantDetails" association between the ReadCaseParticipantDetails struct and the <EvidenceEntity> entity is missing. This association is mandatory for all evidence entities.

Solution:

Create an association as described between the two structs. See "Asset Entity and Aggregations" on page 5 for more details.

Metatypes

Specifying metatypes on fields is a way to force certain additional behavior on the field, e.g. turning the stored value into a link or having a text area rather than field displayed. It is possible that a developer might incorrectly specify a metatype. Examples of the common mistakes are outlined below.

Incorrect Participant Metatype

Symptom:

The primary case participant's name does not appear on the evidence maintenance screens as a link to the case participant home page.

Cause: One of CASE_PARTICIPANT_SEARCH or PARENT_CASE_PARTICIPANT_ROLE_ID was not specified as the metatype on the field that stores the case participant role ID.

Solution:

Set the metatype of the field that stores the case participant role ID

to be one of CASE_PARTICIPANT_SEARCH or PARENT_CASE_PARTICIPANT_ROLE_ID.

Incorrect Date Metatype

Symptom:

The "start" and "end" dates on the evidence workspace screen are not being populated.

Cause: In the metadata for the fields storing the "start" and "end" dates, the metatype of START_DATE or END_DATE was not specified.

Solution:

Specify the metatype of START_DATE or END_DATE to the appropriate field.

Incorrect Comments Metatype

Symptom:

Comments field on an evidence screen has field height of one row and spans only half the screen.

Cause: In the metadata for the field storing the Comments data, the metatype of COMMENTS was not specified.

Solution:

Specify the metatype of COMMENTS to the appropriate field.

Properties

Generation Errors

The general properties file was not found at the location \$properties.home\:

Symptom:

Error when build generated is called displayed in XML Digester output:
The general properties file was not found at the location \$properties.home\

Cause: general.properties does not exist.

Solution:

If general.properties does not exist, create and set properties.home to point to it.

The employment properties file was not found at the location \$properties.home\:

Symptom:

Error when build generated is called displayed in XML Digester output:
The employment properties file was not found at the location \$properties.home\

Cause: employment.properties does not exist.

Solution:

If employment.properties does not exist, create and set properties.home to point to it.

No such property exists:

Symptom:

Error when build client is called: The text property <evidence property>

used in the file <generated evidence VIM or UIM > could not be resolved as no such property exists in the properties file <generated evidence properties file >.

Cause: The property key above is missing from either the general.properties or employment.properties file.

Solution:

See *Cúram Evidence Generator Specification - Appendix B general.properties* and *Cúram Evidence Generator Specification - Appendix C employment.properties* for mandatory property keys. The missing key should be in one of these. The generated properties file that is required to have it should give an indication whether this property is from the general or employment.properties.

Notices

This information was developed for products and services offered in the United States.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM® product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-17

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbash

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM 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 IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Privacy Policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings

can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user's name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at " Copyright and trademark information " at <http://www.ibm.com/legal/copytrade.shtml>.

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.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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 names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.



Printed in USA