

IBM Cúram Social Program Management
Version 7.0.5

Developing Evidence



Note

Before using this information and the product it supports, read the information in [“Notices” on page 142](#)

Edition

This edition applies to IBM® Cúram Social Program Management v7.0.5 and to all subsequent releases unless otherwise indicated in new editions.

Licensed Materials - Property of IBM.

© **Copyright International Business Machines Corporation 2012, 2018.**

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

Contents

List of Figures.....	iv
List of Tables.....	v
Chapter 1. Implementing evidence.....	1
Developing dynamic evidence.....	1
Evidence Generator Cookbook.....	1
Developing static evidence.....	83
Evidence patterns.....	83
The Evidence Generator.....	85
Modeling for the Evidence Generator.....	87
Developing custom evidence.....	94
Developing evidence manually.....	94
Customizing dynamic evidence.....	132
Designing an evidence solution.....	135
Modeling the evidence solutions.....	135
Designing the user interface.....	138
Defining algorithms for calculating attribution periods.....	139
Validating evidence.....	139
Extending evidence processing.....	140
External APIs.....	140
Notices.....	142
Privacy Policy considerations.....	143
Trademarks.....	143

List of Figures

1. Asset entity diagram.....6

2. Sample custom evidence.properties..... 11

3. Participant evidence sequence.....128

4. Evidence sequence diagram..... 129

5. Modify participant..... 130

List of Tables

- 1. Entities in the Administration Manager for evidence..... 25
- 2. Entities in the Case Manager for evidence..... 26
- 3. Attributes in the Evidence Metadata entity..... 26
- 4. Attributes in the Product Evidence Link entity..... 27
- 5. Attributes in the Admin IC Evidence Link entity.....28
- 6. Attributes in the Temporal Evidence Approval Link entity..... 29
- 7. Attributes in the PDC Evidence Link entity..... 30
- 8. Attributes in the Evidence Type Def entity..... 30
- 9. Attributes in the Case Config Evidence Link entity.....31
- 10. Attributes in the Admin IC Evidence Type Def Link entity..... 32
- 11. Attributes in the Evidence Descriptor Entity..... 33
- 12. Attributes in the Attributed Evidence entity.....36
- 13. Attributes in the Evidence Approval Check entity.....37
- 14. Attributes in the evidence relationship entity..... 39
- 15. Additional aggregations..... 93
- 16. Additional aggregations..... 93
- 17. Additional aggregations..... 94
- 18. Sample mapping of evidence to products.....137
- 19. Relationships between evidence types.....137
- 20. Standard information provided in views..... 138

Chapter 1. Implementing evidence

Implementing evidence can involve developing dynamic, static, and custom evidence, customizing dynamic evidence, and designing an evidence solution.

Developing dynamic evidence

To develop dynamic evidence, use the Evidence Generator both 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 and as a rapid way to develop the server side code and client side screens for evidence entities that integrate fully with the standard Cúram Evidence Solution, and be familiar with evidence types and the evidence pattern.

Evidence Generator Cookbook

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.

Developing dynamic evidence with the Evidence Generator: introduction

Use the Evidence Generator to develop dynamic evidence.

The proceeding three areas are addressed in the following pages:

- Configuring the Evidence Generator.
- Developing evidence entities that use the Evidence Generator.
- Applying patterns during development, the required metadata for each pattern, and the effect of the generated output.

Before you use the Evidence Generator, be familiar with the information in the *Developing an evidence solution* related link and the *Developing with evidence* related links.

For more information about developing dynamic evidence, see the *Modeling for the Evidence Generator* related link and the *Evidence Generator specification* related link.

Related concepts

Developing evidence manually

Custom evidence solutions can be developed with Cúram Evidence. All of the evidence server-side infrastructure artifacts are available in the `curam.core.sl.infrastructure.impl` package. The evidence metadata entity contains metadata about each evidence type. This entity must be populated before evidence maintenance can proceed. Evidence maintenance functions are available in the administration application.

Modeling for the Evidence Generator

Specific entity modeling is required when you use the Cúram Evidence Generator as the generator relies on certain, attributes, structs, and aggregations within the generated code. Use this information to learn about entity modeling that is required to use the Cúram evidence generator. The evidence generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

Evidence Generator specification

Use the Cúram Evidence Generator as a rapid way to develop the server side code and client side screens for evidence entities that integrate fully with the standard Cúram Evidence Solution.

Related information

Designing an Evidence Solution

Quick overview

Use the Evidence Generator for all the high level, repeatable patterns that are identified across various large evidence-based solutions that are provided by the application.

When to use the Evidence Generator

For more information about repeatable patterns, see the *Evidence Generator specification* related link. Use custom solutions to identify patterns that are not catered for by the generator. For patterns that are not catered for by the generator, the solution must develop the entities manually, that is, outside the generator. Such patterns are untypical.

The Evidence Generator is run as part of the standard Cúram build targets. The generator iterates through every evidence folder under each component. The generator initially targets the file `evidence.properties`. The file `evidence.properties` defines the paths to various files and folders that are required during generation. Where the file `evidence.properties` does not exist, the generator moves to the next folder.

Sample component

A sample directory of the finished component includes:

1. A model directory. The model directory contains any model files that are used for the evidence entity modeling.
2. An 'evidence' directory that contains the `evidence.properties`.

The `evidence.properties` then defines the 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.

Related concepts

[Evidence Generator specification](#)

Use the Cúram Evidence Generator as a rapid way to develop the server side code and client side screens for evidence entities that integrate fully with the standard Cúram Evidence Solution.

Generator inputs and outputs

The Evidence Generator uses five resources as input data. The Evidence Generator produces five outputs.

Generator inputs

The Evidence Generator uses the following resources as input data:

evidence.properties

The `evidence.properties` is a resource to configure the Evidence Generator. The file `evidence.properties` contains all the product and component-specific properties. For example, naming conventions, directory locations, and product-wide settings. Some of these properties are also included in the generation itself. These properties are defined once per product.

general.properties and employment.properties

The resources `general.properties` and `employment.properties` generate the client screens. The resources contain generic text labels that are used on many client screens. Descriptions of these fields are used in the application online help. These properties are defined once per product.

Server metadata file (for example, Expenses.xml)

The server metadata file defines the names of your entities and the entities' relationships to other evidence entities.

Client metadata file (for example, Expenses.euim)

This client metadata file defines the client screens that are used to maintain your evidence entity.

Client properties file (for example, Expenses.properties)

The client properties file is required by your Evidence User Interface Metadata (EUM) file. The client properties file defines the text labels that are used and the descriptions of these fields that are used for the application online help and a modeled entity.

Generator outputs

The Evidence Generator produces the outputs:

1. Facade and service layer model.
2. Java code.
3. Client UIM/VIM.
4. Wizard data APPRESOURCE.dmx.
5. Tab configurations.

Configuring an existing product

By configuring an existing product for use with the Evidence Generator, the product is ready for its first generatable evidence implementations.

About this task

To configure an existing product, five steps are required:

1. Create an evidence directory.
2. Create and configure the evidence properties file (`evidence.properties`).
3. Create the general properties file (`general.properties`).
4. Create the product employment properties file (`employment.properties`).
5. Configure the module.

Procedure

1. Create directory evidence under the product root directory in `EJBServer`. For example, `SampleEGProduct` is used as the product name. So, the evidence directory is `EJBServer/components/SampleEGProduct/evidence`.
2. Create an `evidence.properties` file. Use the file to configure various mandatory product parameters, including locations of input files, such as EUMs, and locations of output files, such as generated UIMs.

Note: The location of the `evidence.properties` is important. The location *must* be within a directory named `evidence`. However, you can locate the directory anywhere within your component. For convenience the following location is suggested:

```
EJBServer/components/  
SampleEGProduct/evidence/evidence.properties
```

Within the properties file, specify the location of the remaining mandatory files in arbitrary locations. Again, for convenience, sub directories under the evidence directory are the logical choice.

The proceeding is a sample of the product parameters required. For a complete list of product parameters, see the *evidence.properties: explanation and sample file* related link.

```
product.name=SampleEGProduct
```

This setting copies the generated evidence files to `./components/SampleEGProduct`.

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

Based on the product name in the previous example, the code package name might, for example, be `seg`. So, the format of the package structure of the generated classes is `curam.seg.evidence`.

Note: Setting the preceding property to `evidence` generates a package structure of `curam.evidence` (not `curam.evidence.evidence`).

In the prefix

```
product.prefix=SEG
```

the prefix is prepended to the name of all generated UIM pages and certain generated classes, for example, the façade. Here, the generated façade class is SEGEvidenceMaintenance.

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

The location of the root directory for client product is webclient/components/SampleEGProduct.

Note: Set the property `${webclient.dir}` in the Evidence Generator. The property points to the directory webclient/components. Using the property is optional for the user.

3. Create the general properties file (`general.properties`). The file contains all generic client page properties, client message properties, and online help properties for this product. For more information about the general properties file, see the *general properties* related link.

Note: All the keys (properties) specified in the *general properties* related link are mandatory. Omission of any keys is likely to break the build or cause compilation errors.

4. Create the product employment properties file (`employment.properties`). The file contains all generic employment that is specific to client page properties, client message properties, and online help properties for the product. For more information about the employment properties file, see the *employment.properties* related link.

Note: Like `general.properties`, all the keys (properties) specified in the *employment.properties* related link are mandatory. Omission of any keys is likely to break the build or cause compilation errors.

5. Configure the module. The Evidence Generator produces a single registrar module for all the generated evidence types, which 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. In the preceding example, the class that is generated is `curam.seg.evidence.service.impl.SEGRegistrarModule`.

Related concepts

[evidence.properties: explanation and sample file](#)

The `evidence.properties` file is used to configure the generator options.

[general.properties](#)

The `general.properties` file contains all generic label values for the product. The generic labels consist of localized label values for all common buttons, page titles, and so on. Some generic labels permit dynamic values, that is, the name of the evidence entity the page title is describing. All properties within this file must be set.

[employment.properties](#)

The `employment.properties` file contains all generic label values for the employment pages generated. The generic label values consist of localized label values for all common buttons, page titles, and so on.

Asset as generated evidence: implementing a sample evidence type

To generate asset as evidence, you must generate the server-side and client-side artefacts for the evidence entity.

Step 1: Model evidence entity

During entity modeling, the defined metadata is used to support and connect to the Evidence Generator by using the service layer, façade layer, or client.

Modeling the evidence entity is independent from the Evidence Generator. The evidence entity is modeled in the standard way and included in the standard Cúram build. For more information about evidence entity modeling, see the *Modeling for the Evidence Generator* related link.

Asset entity and aggregations

The attributes of the asset entity are:

- Value
- Asset type
- Start date
- End date

The screens for maintaining the expense entity display the employer of the case participant and the associated record. The related information is not stored on the expense entity. Instead, the information is only displayed on the screens where it is deemed that it is useful to the caseworker as the caseworker maintains the expense information.

The expense entity must include the attributes (with their associated domain definition):

- The primary key of the entity evidenceID (this is expected by the generator).
- All other attributes as required.
- Optimistic locking on the entity enabled.
- The entity with the standard read, insert, and modify operations automatically generated.

Adhere to the naming conventions, for example, the naming of structs and aggregations that are required for each entity, and multiplicities for the aggregations and code packages that match the `product.ejb.package` property. For more information about naming conventions, see the *Modeling for the Evidence Generator* related link.

Additional modeling

Displaying the employment name on the maintenance screens for the Expense entity is not necessary. However, communicating the information from the system to the screen is required. Use a 'placeholder' to communicate the information. Use a `RelatedEntityAttribute` struct to create the placeholder. A `RelatedEntityAttribute` struct is an ordinary struct with a specific naming convention and aggregation. You must adhere to the conventions that are outlined in the *Modeling for the Evidence Generator* related link.

In the preceding example, 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`.

Related concepts

Modeling for the Evidence Generator

Specific entity modeling is required when you use the Cúram Evidence Generator as the generator relies on certain, attributes, structs, and aggregations within the generated code. Use this information to learn about entity modeling that is required to use the Cúram evidence generator. The evidence generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

Asset entity diagram

You can complete the model evidence entity when the entity's attributes are defined and the necessary structs and aggregations are modeled.

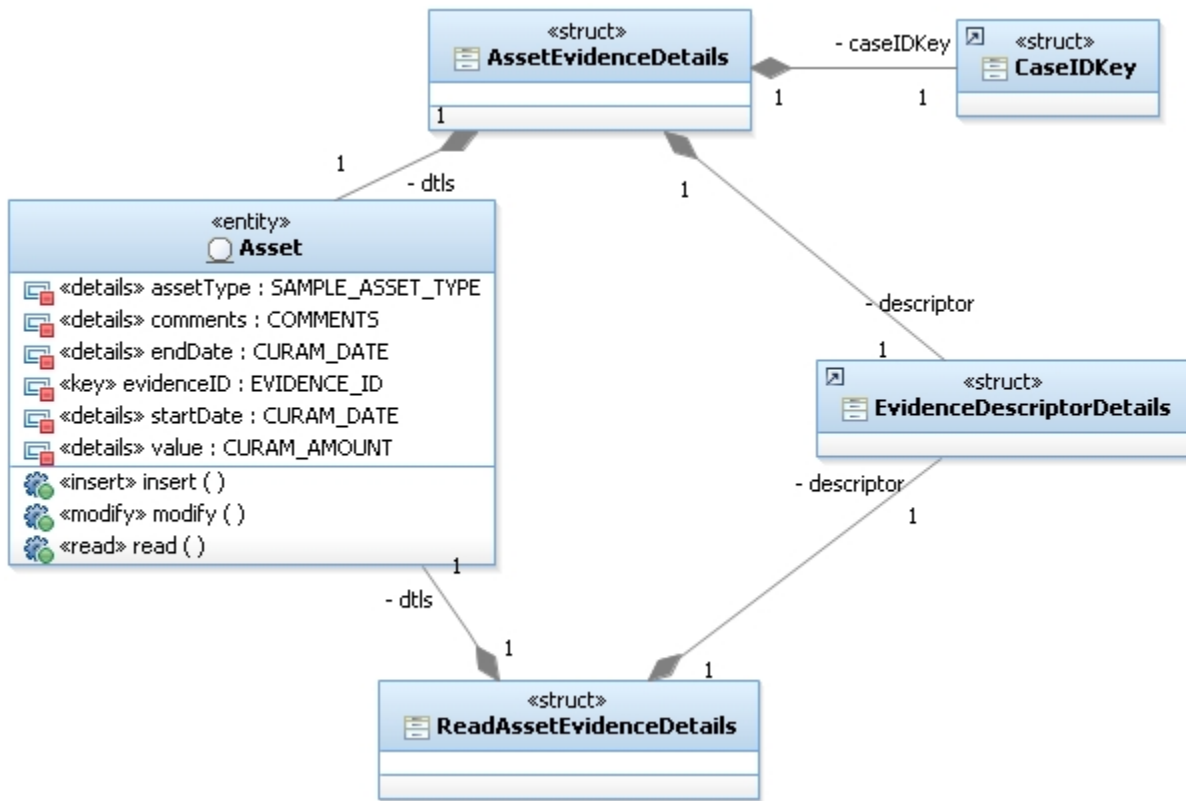


Figure 1: Asset entity diagram

Step 2: Create evidence metadata

The Evidence Generator is configured to identify specific files.

In the configured directories, the Evidence Generator is configured to identify:

- Server XML metadata files
- Integrated case EUIM metadata files and their corresponding properties files
- Product delivery EUIM metadata files and their corresponding properties files

Each entity has one server XML file and one pair of EUIM and properties files that define the entity.

Asset server XML

Specific attributes are required to generate the wizard page when you configure evidence end dating for non-dynamic evidence.

The proceeding sample 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>
```

Generating the wizard page when you configure evidence end dating for non-dynamic evidence

To generate the wizard page when you configure evidence end dating for non-dynamic evidence, the following attributes must be present in the asset server XML metadata file:

- `<AutoEndDate active="Yes"/>`
- `<BusinessDates endDate="endDate" />`

The proceeding example shows the asset server XML file after you add the attributes:

```
<EvidenceEntity>
  <Entity
    logicalName="Asset"
    relateEvidenceParticipantID=""
  >
    <AutoEndDate active="Yes"/>
    <RelatedEntityAttributes
      exposeOperation="No"
      relatedEntityAttributes="No"
    />
    <Relationships/>
    <BusinessDates
      startDate="startDate"
      endDate="endDate"
    />
  </Entity>
</EvidenceEntity>
```

To disable the wizard page generation for the evidence end dating feature:

- Remove the `<AutoEndDate active='Yes' />` attribute from the XML metadata OR
- Update the value of the active attribute value to No; for example, `<AutoEndDate active='No' />`.

Asset client Evidence UIM (EUIM)

View the client Evidence UIM (EUIM) metadata file for asset and the associated properties file for Asset.euim.

The client Evidence UIM (EUIM) metadata file for asset is:

```
<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 to UIM. For example, data is described in terms of 'fields' and the layout is described in terms of 'labels', 'clusters', and 'fields'. EUIM uses a format with which developers are familiar.

The associated properties file for Asset.euim is:

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

Specific steps are required to configure a new evidence type.

Checklist to configure a new evidence type before you generate an asset

- To name the asset evidence type, add an entry to the Evidence Type Code Table OR
- Create a static description for asset evidence by using a new entry in the Text Translation initial data. Link the Text Translation to a new entry in the Localizable Text initial data. As the step is only visible to the user on the New Evidence screen, you can defer the step until later.
- Add an entry in the Evidence Metadata initial data linking it to the Evidence Type and, optionally for now, to 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, for example, Resources, set the category attribute here.
- If the asset Evidence Business Object Tab is to be used in a section of the application, contribute to the section definition, for example, file `DefaultAppSection.sec`. Without this contribution, the asset Evidence Business Object page loads in the current content panel only.

A sample section file is generated for each product, including all the evidence tabs. The location of the sample is `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: One handcrafted implementation **must** be completed after the generator is run. If the handcrafted implementation is not completed after the generator is run, the user cannot access some evidence screens. For more information, see the *Asset handcrafted code: asset hook `getDetailsForListDisplay`* related link.

Related concepts

Asset handcrafted code: asset hook `getDetailsForListDisplay`

All evidence entities must implement the asset hook method `getDetailsForListDisplay`.

Executing the Evidence Generator

Users can call on Evidence Generator targets and expect specific outcomes.

Evidence Generator standard targets

The Evidence Generator builds targets into the standard targets.

The Evidence Generator has a clear design, that is, the generator integrates the standard build targets so that:

- No extra environment variables are required.
- No new targets are required to generate evidence.

When the preceding steps are complete, the standard build targets suffice to generate or clean:

- The metadata driven-evidence.
- The standard files.

build generated

By calling the build generated target, the EJBServer generates:

- The evidence inf and impl layers.
- The normal server layers.

build client

By calling the build client target in web client, the EJBServer generates and builds:

- The client screens
- The standard client screens

Note: Like a normal build, if the build client is called before a build that is generated after changes to the model or metadata, the client build might fail. Typically, the failure is due to changes in the client UIMs or VIMs to use new features that are implemented on the server and then rebuilding the client without first rebuilding the server.

With evidence generation, any changes to the EUIMs or server XMLs are automatically generated the next time that the client is generated. So, if the EUIM or server XML changes affect the façade layer in any way, you must first generate the server.

build clean

The target to clean generated evidence is incorporated into the standard target so that the target is clear. The target is the same on the server and the client, build clean.

Note: Customized generated code is not deleted. For more information, see the *Asset handcrafted code: asset hook getDetailsForListDisplay* related link.

Related concepts

Asset handcrafted code: asset hook getDetailsForListDisplay

All evidence entities must implement the asset hook method `getDetailsForListDisplay`.

Evidence Generator specific targets

Use the specific targets `build egtools.clean` and `build egtools.client.clean` to provide more granular control over evidence generation.

By using the specific targets `build egtools.clean` and `build egtools.client.clean`, you can speed up the development process because:

- The specific targets clean the generated evidence.
- The specific targets do not remove any of the standard generated files.

`build egtools.clean` and `build egtools.client.clean` are located within `EJBServer/build.xml`.

build egtools.clean

The `build egtools.clean` target cleans all server-related evidence files. The clean is only applied if the prerequisites are met. The clean is applied whether:

- New EUIM and server XML files are added OR
- Existing EUIM and server XML files were updated since the last build.

build egtools.client.clean

The build `egtools.client.clean` target cleans all client-related evidence files. The clean is only applied if the prerequisites are met. The clean is applied whether:

- New EUIM and server XML files are added OR
- Existing EUIM and server XML files were updated since the last build.

Generator output

When evidence generation is complete, new directories are added in the locations that are specified in the `evidence.properties` file.

Note: As all entity, service, and façade level-generated code is written directly to the build directory, the code is not displayed within your components source directory.

Asset handcrafted code: asset hook `getDetailsForListDisplay`

All evidence entities must implement the asset hook method `getDetailsForListDisplay`.

Custom code can be written in some stubs that are generated by the server output, that is, placeholders for customers to add their own code. The placeholders provide flexibility when a generated evidence pattern is not an exact match for an evidence entity. For more information about extending the function of a generated entity, see the *Adding functionality* related link.

Asset hook `getDetailsForListDisplay`

Implementing the asset hook method `getDetailsForListDisplay` is mandatory for all evidence entities. By using `getDetailsForListDisplay`, text descriptions are created for a particular asset business object on the evidence workspace pages. As the link text is used on the client screens, the link text must be populated to access all screens.

The `getDetailsForListDisplay` implementation for asset is:

```
//-----  
/**  
 * 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.getOneItem(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 eiFieldsForListDisplayDtls;
}

```

Related concepts

Adding functionality

There are a number of extension classes that can be coded, with the generator providing a default skeleton implementation for each in your source code directory. Additionally each of these classes is automatically modeled by the generator, so all follow the standard factory, interface, implementation pattern used in the application.

Customizing a product

While a default evidence solution is provided with some of the Cúram solutions, the customer can extend and customize the default evidence solution to match the customer's business requirements.

Custom evidence properties

The default product is configured with an `evidence.properties` file. For more information about creating and configuring an `evidence.properties` file, see the *Configuring an existing product* related link. To override a default product, the custom product requires its own, thin version of `evidence.properties`.

Note:

The `override.product` property must be set to `product.name`. Otherwise, the evidence generator treats the evidence product as new. For more information about evidence properties, see the *evidence.properties: explanation and sample file* related link.

```

# 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 `evidence.properties` must be located in a directory that is named `evidence` within any subdirectory of:

`EJBServer/components/custom`

As the custom directory can contain many of the overridden products and the evidence directories, use a naming scheme. For example:

```

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

```

Build process and generated files: an overview

The evidence generator build process identifies evidence sub directories in all the components that are listed in the `SERVER_COMPONENT_ORDER`. During the build process:

1. The product's metadata and display properties are gathered to the build directory.
2. A search of the custom directory finds any `evidence.properties` that override the queued product.

Where overriding in the build is required, the customized metadata, and the display properties, are gathered and copied over the queued product's metadata in the build directory. The customized metadata, and the display properties, are not merged. The product's evidence is then generated from the super-set of metadata.

Note: Most artefacts generated by a default product are not modifiable. Likewise, most artefacts generated by a default product are added to source control.

The only artefacts that are modifiable are the handcrafted Java™ classes that are provided for customizable hook points that are called throughout the non-modifiable generated codebase. The handcrafted Java classes are only generated where they did not exist. Then, the handcrafted Java classes must be maintained as part of source control.

Therefore, by overwriting the metadata before the build all the generated custom artefacts are generated as if they belonged to the default product, that is, the product's directories. The only exception is handcrafted Java classes.

Overriding display text

Display text is defined in the properties files that are associated with:

- An EUIM
- The general properties file
- The employment properties file

The preceding files can be overridden in the custom directory.

Related concepts

[evidence.properties: explanation and sample file](#)

The `evidence.properties` file is used to configure the generator options.

Related tasks

[Configuring an existing product](#)

By configuring an existing product for use with the Evidence Generator, the product is ready for its first generatable evidence implementations.

Overriding a default evidence entity: example

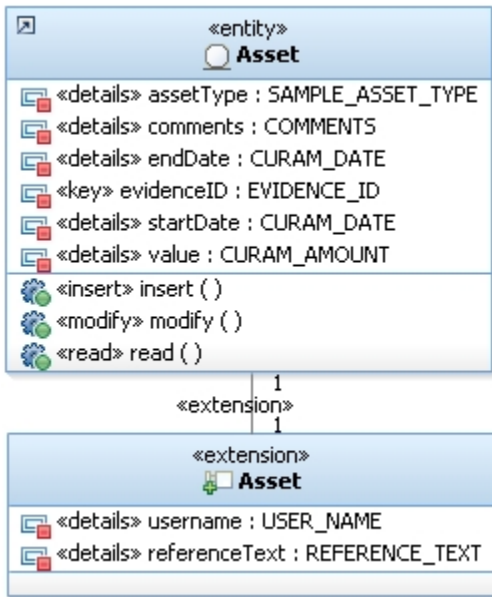
To meet business requirements, customers can override the default evidence entity by, for example, editing the server-side metadata and client-side metadata.

In the proceeding example, it is assumed that the expense entity was provided as part of a Cúram evidence solution. The customer decided that the entity does not provide the fields that are required to fully meet the business requirements. To meet their requirements, the customer added two extra attributes to the entity:

1. The user name of the user who creates or modifies the record.
2. The number, if any, of children that the case participant has.

Modeling

By conforming with the requisite guidelines, an extension class is created and the class is linked to the provided expense entity. For more information about modeling guidelines, see the *Modeling for the Evidence Generator* related link and the *Cúram modeling reference* related link.



Metadata

The metadata for a customized entity is almost identical to the standard metadata. The metadata for a customized entity is captured in two files:

- <Entity-Name>.xml
- <Entity-Name>.euim

To start customizing an entity, copy all the default entity's metadata and then make the required changes. The two types of metadata are:

- Server-side metadata
- Client-side metadata

Server-side metadata

The asset changes apply only to extra fields. So, with one exception the server-side metadata is identical to the metadata of the entity that you are overriding. The exception is that an extra node **Override** is required. The extra node specifies:

- Whether the entity is new
- The custom handcrafted classes to generate.

For more information about overriding nodes, see the *Server metadata: document structure* related link.

The proceeding 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

Except for including any extra required fields, the client-side metadata is identical to the metadata of the entity that you are overriding. In the proceeding example, you must include the **reference text** field on the user interface so the user can populate the field. Do not display the user name on the user interface.

Note: You cannot remove any attributes from an entity.

The proceeding 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 proceeding 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
```

Generated output

Other than the handcrafted code, everything else is generated in the same way it is when the default entity is defined.

For a custom extension for a default entity, handcrafted implementations pre-exist. The generator creates handcrafted classes in the custom source package. Where the superclass is the existing default implementation, the handcrafted classes are modeled by using the replace superclass option. The superclass contains method stubs only. By default, each of the method stubs begins by calling the superclass implementation.

In the preceding example, you must update the handcrafted preCreate function to assign the value of the user name attribute to the creation struct. Also, you must update the handcrafted validateDetails function to ensure the **reference text** field is not left blank.

Related concepts

[Modeling for the Evidence Generator](#)

Specific entity modeling is required when you use the Cúram Evidence Generator as the generator relies on certain, attributes, structs, and aggregations within the generated code. Use this information to learn about entity modeling that is required to use the Cúram evidence generator. The evidence generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

[Server metadata: document structure](#)

The server-side metadata is provided as a well-formed XML document, named <Entity Name>.xml.

Related information

[Cúram modeling reference](#)

Adding a new custom entity

To add a new custom entity to a custom evidence product that overrides a default product, develop the entity in the same way you develop an entity in any other product but with one exception.

The exception is:

- Use the Override node.
- Set the newEntity to **Yes**.

Note: Use the same codepath as in the default product.

Generated output

In the preceding example, you do not implement the default handcrafted code. To ensure that the code is as simple as possible, a copy of the default handcrafted code is generated inside the build source directory that is under the default's code package. Ensure that the derived custom version is:

- Generated into the custom source directory
- Added to source control

Identifying entities, patterns and relationships

You can use four types of evidence relationships: parent-child, pre-associated, multiple mandatory parents, and related relationships.

Identifying entities

Evidence is data that is collected by an organization to facilitate the delivery of services to the organization's clients. In the application, evidence is typically used to determine clients' eligibility and entitlement. For the Evidence Generator, evidence is:

- Any entity that implements the standard evidence interface AND
- Maintained by the evidence solution.

Identifying patterns

A pattern is any function the evidence entity uses. Examples of functions are:

- Features within a maintenance screen
- Extra code that is specific to an entity

By using metadata that is captured in XML, a function of the Evidence Generator is to specify the patterns that apply to specific entities. During evidence generation, the metadata is read and converted to the appropriate feature. Examples are:

- A button on a client page.
- A callout class stub where you can then implement business logic.

Identifying relationships

In evidence, relationships describe how evidence entities interact and exist in relation to each other. Use a function in the generator to specify the relationships between evidence entities. Then, the generator produces the associated server-side code and client page functions to facilitate the maintenance of the relationships. You can use four types of evidence relationships: parent-child, pre-associated, multiple mandatory parents, and related relationships.

Parent-Child relationships

Parent-Child is one of the most common logical relationships between evidence entities. Typically, a parent-child relationship is a one-to-many relationship where:

- The parent can have many children AND
- Each child must belong to a parent.

Use parent-child relationships 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.

An example of a parent-child relationship is where:

- Student details are stored in a student entity AND
- Student expenses are stored in a student expenses entity.

In this example, student expenses cannot exist without the student entity, but the student entity can exist on its own.

Pre-Associated relationships

Pre-Associated relationships are non-hierarchical relationships between evidence entities that can exist independently of each other. Before you create the evidence, you must know the association between evidence entities so that you can access data from the associated entity as you create the evidence.

Multiple mandatory parents relationships

Use the multiple mandatory parents relationship pattern where an entity must simultaneously be the child of more than one parent entity.

Related relationships

Related relationships are non-hierarchical. Use related relationships to associate an evidence record to a non-evidence record. A primary example is the relation of evidence-based employment records to the core employment record. That relationship is found in all evidence-based modules that are built by the application.

Examples of evidence-based employment entities are:

- Self-employment
- Paid employment

Such examples are a key functional area typical of solutions. For this reason, evidence-based employment entities are categorized as a separate pattern.

Configuration errors: generation

The Evidence Generator produces seven types of generation errors that are associated with configuration (evidence properties).

Generation errors

The seven types of generation errors that are associated with configuration (evidence properties) are:

1. Evidence will not build or evidence will not clean.

2. Evidence not found.
3. '<EntityName>Details' is not present in the model.
4. No source files match the extensions XML.
5. The general properties file was not found.
6. <\$server.metadata> was found to contain no source files that match the extensions XML.
7. No EUIM source files.

See the following explanation of the symptom, cause, and solution for each generation error.

1. Evidence will not build or Evidence will not clean

Symptom

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

Cause

The `product.build.option` is:

- Set to **False** OR
- Missing.

Solution

If the evidence is to be generated, set `product.build.option=true`. If `product.build.option=true` is missing from the `evidence.properties`, add `product.build.option=true` to `evidence.properties`.

2. 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 the property in the codebase.

Solution

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

3. '<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 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
```

4. No source files match 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 that match the extensions XML.'

Cause

`server.metadata` does not match the physical root directory for the product's evidence directory.

Solution

Set `server.metadata` to point to the correct directory.

5. The general properties file was not found**Symptom**

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

Cause

`properties.home` does not match the physical properties directory.

Solution

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

6. <\$server.metadata> was found to contain no source files that match the extensions XML**Symptom 1**

Error when build generated is called on `EJBServer:Error#`. The source location `<$server.metadata>` contains no source files that match 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>` contains no source files that match the extensions XML.

Cause 2

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

Solution 2

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

7. 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>.

Configuration error: runtime

The Evidence Generator produces one runtime error that is associated with configuration (evidence properties): HTTP Status 404 error message.

HTTP Status 404 Error Message

Symptom

A page not found error when the page tries to access the generated evidence workspace.

Cause

`product.codetable` is set incorrectly, that is, not pointing at the product codetable directory.

Solution

Set `product.codetable=<product_Root_CodeTable_directory>`.

Model errors: generation

The Evidence Generator produces one generation model error.

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. The association is mandatory for all evidence entities.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

Related concepts

[Step 1: Model evidence entity](#)

During entity modeling, the defined metadata is used to support and connect to the Evidence Generator by using the service layer, façade layer, or client.

Model errors: compilation

The Evidence Generator produces 11 types of compilation model errors.

Compilation errors

The 11 types of compilation model errors are:

1. '`<EntityName>Details`' is not present in the model.
2. `details.parEvKey` cannot be resolved or is not a field.
3. `evidenceDetails.parEvKey` cannot be resolved or is not a field.
4. `dtls.selectedParent` cannot be resolved or is not a field.
5. `dtls.caseIDKey` cannot be resolved or is not a field.
6. `evidenceDetails.caseIDKey` cannot be resolved or is not a field.
7. `readEvidenceDetails.descriptor` cannot be resolved or is not a field.
8. `details.descriptor` cannot be resolved or is not a field.
9. `evidenceDetails.descriptor` cannot be resolved or is not a field.
10. `readEvidenceDetails.dtls` cannot be resolved or is not a field.
11. `readEvidenceDetails.caseParticipantDetails` cannot be resolved or is not a field.

See the following explanation of the symptom, cause, and solution for each compilation error.

1. '`<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, that is, up to the 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 the other way around.

Solution 2

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

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

A compilation error in generated code evidenceDetails.parEvKey cannot be resolved or is not a field.

Cause

The "parEvKey" association between the <EvidenceEntity>Details struct and the EvidenceKey struct is missing.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

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

A compilation error in generated code evidenceDetails.parEvKey cannot be resolved or is not a field.

Cause

The "parEvKey" association between the <EvidenceEntity>Details struct and the EvidenceKey struct is missing.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

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

A compilation error in the 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 ParentSelectDetails and the entity details struct is required if the entity is a child of another evidence entity.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

5. dtls.caseIDKey cannot be resolved or is not a field**Symptom**

A compilation error in the 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 between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

6. evidenceDetails.caseIDKey cannot be resolved or is not a field**Symptom**

A compilation error in the generated code `evidenceDetails.caseIDKey` cannot be resolved or is not a field.

Cause

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

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

7. readEvidenceDetails.descriptor cannot be resolved or is not a field**Symptom**

A compilation error in the 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. The association is mandatory for all evidence entities.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

8. details.descriptor cannot be resolved or is not a field**Symptom**

A compilation error in the 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. The association is mandatory for all evidence entities.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

9. evidenceDetails.descriptor cannot be resolved or is not a field**Symptom**

A compilation error in the generated code `evidenceDetails.descriptor` cannot be resolved or is not a field.

Cause

The "descriptor" association between the `<EvidenceEntity>Details` struct and the `EvidenceDescriptorDetails` struct is missing.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

10. readEvidenceDetails.dtls cannot be resolved or is not a field**Symptom**

A compilation error in the 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. The association is mandatory for all evidence entities.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

11. readEvidenceDetails.caseParticipantDetails cannot be resolved or is not a field**Symptom**

A compilation error in the 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. The association is mandatory for all evidence entities.

Solution

Create an association between the two structs. For more information about creating an association between two structs, see the *Step 1: Model evidence entity* related link.

Related concepts

[Step 1: Model evidence entity](#)

During entity modeling, the defined metadata is used to support and connect to the Evidence Generator by using the service layer, façade layer, or client.

Metatype errors: incorrect participant, date, and comments

Specify metatypes on fields to force extra behavior on the field. If you incorrectly specify a metatype, the error typically relates to the metatype participant, date, or comments.

Examples of metatype uses:

- Turn the stored value in a field into a link.
- Display a text area rather than field.

See the following explanation of the symptom, cause, and solution for common metatype errors.

Incorrect participant metatype**Symptom**

On the evidence maintenance screens, the primary case participant's name does not display as a link to the case participant home page.

Cause

The `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 either:

- `CASE_PARTICIPANT_SEARCH` OR
- `PARENT_CASE_PARTICIPANT_ROLE_ID`.

Incorrect date metatype**Symptom**

The "start" and "end" dates on the evidence workspace screen do not populate.

Cause

In the metadata for the fields that store 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

The comments field in an evidence screen has a field height of one row and displays on half the screen only.

Cause

In the metadata for the field that stores the comments data, the metatype of COMMENTS was not specified.

Solution

Specify the metatype of COMMENTS to the appropriate field.

Property errors: generation

Three property errors are common during generation.

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

Symptom

Error when build generated is called displayed in XML Digestor 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 `general.properties`.

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

Symptom

Error when build generated is called displayed in XML Digestor 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 `employment.properties`.

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 is missing from either the `general.properties` file or the `employment.properties` file.

Solution

The missing key is likely in the `general.properties` file or the `employment.properties` file. View the generated properties file. The generated properties file that is required to contain the missing property key might indicate whether the property is from the general or employment properties. For more information about mandatory property keys, see the *general properties* related link and the *employment.properties* related link.

Related concepts

general.properties

The `general.properties` file contains all generic label values for the product. The generic labels consist of localized label values for all common buttons, page titles, and so on. Some generic labels

permit dynamic values, that is, the name of the evidence entity the page title is describing. All properties within this file must be set.

employment.properties

The `employment.properties` file contains all generic label values for the employment pages generated. The generic label values consist of localized label values for all common buttons, page titles, and so on.

Evidence types

Evidence types represent the events and circumstances that need to be captured for case assessment purposes.

Examples

Examples of evidence types include income, income usage, address, employment, bank details, and so on. An evidence record is a set of saved information that is entered for an evidence type. For example, an employment evidence record might include details about a person's job.

Evidence objects

Evidence objects are collections of evidence records that show how a piece of information varies over time. For example, a person's employment evidence changes twice over the years. This evidence object for employment evidence consists of three records: one record for the initial capture and one for each change in circumstances.

Several evidence objects can be associated with each evidence type. For example, if a person has two jobs, two different evidence objects are stored. The objects, plus the records for that evidence object, store all necessary information.

Evidence corrections and changes in circumstance

Evidence maintenance supports two styles of evidence changes: evidence corrections and changes in circumstance.

Evidence correction

An evidence correction is the replacement of an existing evidence record with a new evidence record to correct an incorrect piece of data. For example, a person might input their date of birth as part of a submitting an application online. When the caseworker interviews the client and verifies the date of birth, the caseworker finds that the customer made a mistake when originally entering the information. The caseworker corrects the date of birth evidence by overwriting the existing date of birth with the correct one. All corrections that are made to evidence can be viewed for historical purposes. Caseworkers can see when the change was made, who changed it, and the original value.

Changes in circumstance

A change in circumstance is when data in an evidence record changes over time due to changes in circumstance. For example, evidence that captures a weekly income amount for a person can vary over time. When the income amount goes up or down, the caseworker can record when the income change took effect.

Such an evidence pattern supports a succession of changes in circumstance to the same evidence object. For example, the set of changes to the income amount, each in succession, collectively represent the changes to the income amount evidence object.

Read-only evidence types

Evidence from a trusted source, such as medical records from a hospital, might need to be protected from modification by a user. An administrator can protect evidence records by making the evidence type read-only.

Evidence records of a read-only evidence type cannot be modified or created by a user, but system processes can create or broker read-only evidence around the application. The worker can manually delete read-only evidence records.

The evidence pattern

The architectural overview of the evidence pattern consists of evidence entities, the evidence controller, and the evidence user interface.

Evidence entities

The evidence framework includes entities in the core reference model for storing evidence information. Most of the entities are part of the Case Manager because evidence is maintained at the case level. There are extra entities in the administration manager that link evidence to products and one entity for setting up evidence approval checks.

Evidence controller

Use the evidence controller to maintain evidence. At each step in the evidence process, the evidence controller manages the step.

Evidence user interface (UI)

Evidence screens and evidence views are provided with the evidence framework. The screens and views provide consistency for capturing, viewing, updating, validating, and activating evidence. The screens and view are reusable and customizable.

Evidence entities

For evidence, the Administration Manager contains eight entities and the Case Manager contains four entities.

Table 1: Entities in the Administration Manager for evidence	
Entity	Description
Evidence Metadata	Stores configuration data about non-dynamic evidence types.
Product Evidence Link	Links evidence types to products.
Admin IC Evidence Link	Links evidence types to integrated cases.
Temporal Evidence Approval Link	Stores configuration of evidence approvals for evidence types.
PDC Evidence Link	Stores evidence types that are associated with a participant data case type.
Evidence Type Def	Defines a dynamic evidence type. This entity has an EvidenceTypeVersionDef child entity that contains metadata information.
Case Config Evidence Link	Stores evidence types that are configured and are, therefore, available on a case. When evidence is added to a case, only evidence that is configured against the case type is available. This entity enables evidence types to be configured against case types. This entity does not support the configuration of evidence for product deliveries and integrated cases. Product delivery and integrated case types use separate entities for associating evidence.
Admin IC Evidence Type Def Link	Defines a link between integrated case types and dynamic evidence types.

<i>Table 2: Entities in the Case Manager for evidence</i>	
Entity	Description
Evidence Descriptor	Centralizes the maintenance of all evidence records. It links the custom evidence entities to the case header.
Attributed Evidence	Stores information about the time period during which an active evidence record is used to determine case eligibility.
Evidence Relationship	Manages relationships between custom business entities. The relationships are normally referred to as parent-child relationships and can exist for several reasons. For example, in one scenario, the parent is the main business entity. The child provides further details that are included in a separate entity because potentially more than one instance of the entity exists.
Evidence Descriptor Approval Request	Indicates whether an evidence record is waiting for approval from a supervisor before the evidence is either activated or canceled.

Evidence metadata

Evidence metadata is maintained as part of the application's administration function.

The evidence metadata entity provides configuration data about an evidence type. The configuration data includes the names of the application pages to be used for displaying and editing evidence.

<i>Table 3: Attributes in the Evidence Metadata entity</i>	
Attribute	Description
Evidence Metadata ID	Unique identifier of the evidence metadata record. Use the primary key to identify the evidence metadata record in the system.
Evidence Type	Type of evidence that the evidence metadata record relates to, that is, a code table value. Use the code table value to identify the evidence to which the configuration data refers.
Effective From	Date from which the configuration data applies to the evidence type. The system determines the correct evidence pages to use for an evidence type based on the effective From date of the evidence metadata record.
View Page Name	Application page name that is used to display evidence of the View Page Name type.
View Snapshot Page Name	Application page name that is used to display the evidence snapshot of the View Snapshot Page Name type. Evidence snapshot is only available for participant evidence.
Create Page Name	Application page name that is used for creating evidence of the Create Page Name type.

<i>Table 3: Attributes in the Evidence Metadata entity (continued)</i>	
Attribute	Description
Modify Page Name	Application page name that is used to edit evidence of Modify Page Name type.
Business Object Page Name	Application page name that is used to view evidence of the Business Object Page Name type from the business object perspective.
Business Object Issues Page Name	Application page name to be used for viewing business object issues for evidence of Business Object Issues Page Name type.
Business Object Verifications Page Name	Application page name that is used for viewing business object verifications for evidence of Business Object Verifications Page Name type.
History Record Page Name	Application page name that is used for the history record for evidence of History Record Page Name type.
Workspace Page Name	Application page name that is used to list evidence of Workspace Page Name type.
Localized Description ID	Unique identifier of the localized description record. The record contains localized description for evidence of Localized Description ID type.
Participant Data Indicator	Indicator for participant data. The indicator is set if the evidence type pertains to participant data.
Participant Data Applies To All Indicator	Indicator that applies only to participant evidence types. The evidence types indicate whether this data is relevant for all case members or just the primary client.
Record Status	Status of the evidence metadata record. The status is active unless the evidence metadata record is deleted. Where the evidence metadata case is deleted, the status is canceled.

Product Evidence Link

Use the Product Evidence Link entity to link evidence metadata to products.

By having a link entity between the evidence metadata and the product entities, the relationships between evidence types and products can be reciprocal. The Product Evidence Link entity specifies the list of evidence types that are used by a product or, conversely, the list of products that use a particular evidence type.

<i>Table 4: Attributes in the Product Evidence Link entity</i>	
Attribute	Description
Product Evidence Link ID	Unique identifier of the product evidence link record, the primary key that identifies the product evidence link record in the system.

<i>Table 4: Attributes in the Product Evidence Link entity (continued)</i>	
Attribute	Description
Product ID	Identifier of the product record that is linked to the evidence metadata record.
Evidence Metadata ID	Identifier of the evidence metadata record that is linked to the product record. The Evidence Metadata ID sets up a relationship between the evidence type that is recorded for the evidence metadata and the product.
Data Linked To Product Indicator	Indicates whether data is captured at the product level rather than the integrated case level.
Shareable Indicator	Indicates whether the evidence type can be shared by the product. If set to true, then the Shareable Indicator evidence type can be used in the Evidence Broker to configure sharing between case types.
Category	Indicates the Category code for the evidence in the context of this product.
Quick Link Indicator	Indicates whether the Quick Link Indicator evidence type appears in the quick link list for adding evidence.
Sort Order	Indicates the sort order for the evidence type. The sort order is an integer value that is used when the system is ordering evidence types. If multiple types have the same sort order, the alphabetic ordering of the type code is used.

Admin IC Evidence Link

Use the Admin IC Evidence Link entity to link evidence metadata to integrated cases.

By using a link entity between the evidence metadata and an integrated case, you can identify the evidence that is stored on the integrated case.

<i>Table 5: Attributes in the Admin IC Evidence Link entity</i>	
Attribute	Description
Admin IC Evidence Link ID	Unique identifier of the Admin IC Evidence Link ID record, the primary key that is used to identify the Admin IC Evidence Link record in the system.
Evidence Metadata ID	Identifier of the evidence metadata record that is linked to the admin integrated case record. The link sets up a relationship between the evidence type that is recorded for the evidence metadata and the integrated case.
Admin Integrated Case ID	Identifier of the admin integrated case record that is linked to the evidence metadata record.

<i>Table 5: Attributes in the Admin IC Evidence Link entity (continued)</i>	
Attribute	Description
Shareable Indicator	Indicates whether the evidence type can be shared by the integrated case. If set to true, then Shareable Indicator can be used in the Evidence Broker to configure sharing between case types.
Category	Indicates the category code for the evidence in the context of the integrated case.
Quick Link Indicator	Indicates whether the evidence type appears in the quick link list for adding evidence.
Sort Order	Indicates the sort order for the evidence type. The sort order is an integer value that is used when the system is ordering evidence types. If multiple types have the same sort order, the alphabetic ordering of the type code is used.

Temporal Evidence Approval Link

The Temporal Evidence Approval Link entity stores the configuration of evidence approvals for evidence types.

<i>Table 6: Attributes in the Temporal Evidence Approval Link entity</i>	
Attribute	Description
temporalEvApprovalCheckID	Unique identifier of the temporal evidence approval check record.
organisationUnitID	Identifier of the organization unit to which the evidence approval check percentage applies, which is relevant only for evidence approval checks that are configured for an organization unit.
userName	Indicates the user name of the user to whom the evidence approval check percentage applies, which is relevant only for evidence approval checks that are configured for a user.
positionID	Identifier of the position to which the evidence approval check percentage applies, which is relevant only for evidence approval checks that are set up for a position.
evidenceType	Indicates the type of evidence to which the evidence approval check percentage applies.
approvalType	Indicates the type of evidence approval check, for example, organization, position, user, or evidence type.
percentage	Indicates the percentage of evidence changes for which manual approval is required.
comments	Indicates extra notes or comments that are entered by a user.

<i>Table 6: Attributes in the Temporal Evidence Approval Link entity (continued)</i>	
Attribute	Description
recordStatus	Indicates the status of the temporal evidence approval check record. The values are either Active or Canceled. Logical deletion is supported.

PDC Evidence Link

The PDC Evidence Link entity stores evidence types that are associated with a participant data case type.

<i>Table 7: Attributes in the PDC Evidence Link entity</i>	
Attribute	Description
pdcevidenceLinkID	Unique reference number of the participant data case evidence link record.
evidenceMetadataID	Identifier of the evidence metadata record that is linked to the participant data case record. The participant data case record sets up a relationship between the evidence type that is recorded for the evidence metadata and the participant data case.
shareableInd	Indicates whether the evidence type can be shared by the participant data case. If the value is set to true, then this evidence type can be used in the Evidence Broker to configure sharing between case types.
category	Indicates category code for the evidence in the context of the participant data case.
quickLinkInd	Indicates whether the quickLinkInd evidence type is displayed in the quick link list for adding evidence.
sortOrder	Indicates that the sort order is an integer value that is used when evidence types are ordered. Where multiple types have the same sort order, the alphabetic ordering of the type code is used.
pdctypeID	Participant data case type identifier that is a foreign key to the participant data case type link entity.

Evidence Type Def

The Evidence Type Def entity defines a dynamic evidence type. The entity includes a EvidenceTypeVersionDef child entity that contains metadata information.

The following table describes each of the attributes in the Evidence Type Def entity:

<i>Table 8: Attributes in the Evidence Type Def entity</i>	
Attribute	Description
evidenceTypeDefID	Unique identifier of the evidence type def record.

Table 8: Attributes in the Evidence Type Def entity (continued)

Attribute	Description
evidenceTypeCode	Indicates the evidence type, which is a value on the evidence type codetable. The code value is defined as the base CODETABLE_CODE domain rather than the specific codetable domain. The reason is that it might refer to values that are added dynamically at runtime rather than just values that are present at compile time.
description	Provides a general description of the description evidence type.
logicalName	Indicates the logical name for logicalName evidence type. The evidence type cannot contain white space and it must be unique across evidence types. The evidence type is used to generate page URLs in the user interface.
securityGroupName	Indicates the security group to which the securityGroupName evidence type def belongs.
evidenceTypeDefinition	Allows for a more detailed description of the evidence type to be provided.
recordStatus	Indicates the status of the evidence type def record. The values are Active or Canceled. Logical deletion is supported.
autoEndDateIndOpt	Indicates, by using a flag, whether the Auto End Date Wizard for the evidence type is enabled, which is an optional attribute that was added in Version 7.0.1.0.
maintenancePattern	Determines how the Evidence Broker handles the maintenancePattern evidence type.

Case Config Evidence Link

The Case Config Evidence Link entity stores evidence types that are configured and are, therefore, available on a case.

When evidence is added to a case, only evidence that is configured against the case type is available. Use the Case Config Evidence Link entity to configure evidence entities against case types. The Case Config Evidence Link entity does not support configuring evidence for product deliveries and integrated cases. Product delivery and integrated case types use separate entities for associating evidence.

Table 9: Attributes in the Case Config Evidence Link entity

Attribute	Description
caseConfigEvidenceLinkID	Unique identifier for the case configuration evidence link record.
caseConfigurationID	Unique identifier for the case configuration record.
evidenceTypeID	Unique reference to the underlying evidence configuration.

<i>Table 9: Attributes in the Case Config Evidence Link entity (continued)</i>	
Attribute	Description
category	Indicates the evidence category for the related evidence type in the context of the case type.
caseType	Indicates the type of case for which the evidence link was created.
recordStatus	Indicates the status of the case evidence configuration record, for example, Active or Canceled.
sortOrder	Indicates a numeric value that is used to order evidence types. Where multiple types have the same sort order, the alphabetic ordering of the type code is used.
shareableInd	Indicates whether the evidence type can be shared by the participant data case. If the value is set to <code>true</code> , then this evidence type can be used in the Evidence Broker to configure sharing between case types.
quickLinkInd	Indicates whether the quickLinkInd evidence type is displayed in the quick link list for adding evidence.
evidenceNature	Indicates the nature of the evidence that is being linked, for example, static or dynamic evidence.
visibleToCitizenInd	Dictates whether this evidence type is displayed to a citizen in the Citizen Account.

Admin IC Evidence Type Def Link

The Admin IC Evidence Type Def Link entity defines a link between integrated case types and dynamic evidence types.

<i>Table 10: Attributes in the Admin IC Evidence Type Def Link entity</i>	
Attribute	Description
shareableInd	Indicates whether the evidence type can be shared by the participant data case. If the value is set to <code>true</code> , then this evidence type can be used in the Evidence Broker to configure sharing between case types.
adminICEvidenceTypeDefLinkID	Unique identifier of the Admin IC Evidence Type Def Link record.
evidenceTypeID	Identifies the associated evidence type def record.
adminIntegratedCaseID	Identifies the integrated case configuration to which the evidence type def configuration is being associated.
quickLinkInd	If the value is set to <code>true</code> , then the associated evidence is displayed on the preferred list for addition to the integration case.

Table 10: Attributes in the Admin IC Evidence Type Def Link entity (continued)	
Attribute	Description
sortOrder	Indicates the order that evidence is displayed in after a caseworker creates the evidence through the dashboard.
category	Indicates the category that the evidence belongs to, for example, Medical or Household.

Evidence descriptor

The Evidence Descriptor ID entity contains the information that is required to maintain a piece of evidence in the system, for example, the evidence record status and the received date.

The role of the Evidence Descriptor ID entity is similar to that of the case header. However, the case header entity contains the information that is standard to maintaining a case in the system.

The evidence descriptor entity stores a summary of common evidence information in a single database table. Storing a summary of common evidence in a single database table simplifies the process of retrieving evidence information. The system can then read a summary of all evidence by using the table instead of reading the individual evidence type database tables.

Including common evidence information in the evidence descriptor entity also reduces repetition as it not required to model the information in each of the evidence entities. For example, information that is related to the status of an evidence record is stored as part of the evidence descriptor entity. So, the need for including a status code in every custom evidence entity is eliminated.

The following table describes each of the attributes in the evidence descriptor entity:

Table 11: Attributes in the Evidence Descriptor Entity	
Attribute	Description
Evidence Descriptor ID	Unique identifier of the evidence descriptor record. Evidence Descriptor ID is the primary key that is used to identify the evidence descriptor record in the system.
Case ID	Identifier of the case that is associated with the evidence. Evidence can be associated with either an integrated case or a product delivery case. When associated with an integrated case, it can be shared across all product delivery cases within the integrated case.
Participant ID	Identifier of the participant to whom the evidence relates. Participant ID can be the primary client of the case or a member of the integrated case. By recording a Participant ID for each evidence record, evidence can be maintained from the participant perspective. For example, a person's evidence can be identified and canceled or transferred to an alternative case.

Table 11: Attributes in the Evidence Descriptor Entity (continued)

Attribute	Description
Status Code	Status of the evidence descriptor record. When an evidence record is inserted, its status is in edit. In edit, evidence becomes active when a user selects to activate it by applying evidence changes. If a user modifies an active evidence record, an additional evidence record with an in edit status is created. This ensures that active evidence remains intact until the evidence changes are fully ready for activation. When updates to pending evidence are activated, the status of the active evidence becomes superseded and its in edit evidence becomes active.
Received Date	Date the organization received the evidence. Received Date is a user entered field.
Correction Set ID	Identifier that is used to track all corrections to a piece of evidence. When evidence is captured, it is assigned a correction set ID. That same piece of evidence can be corrected resulting in any number of new evidence records. However, the Correction Set ID for each of these records is carried through, grouping evidence in a correction set.
Effective From	Date from which a piece of evidence is effective. The Effective From date is stored to support a change of circumstance to a piece of evidence over time.
Succession ID	Identifier of the set of evidence records that collectively represents a real world item of evidence and the changes that are made to it over time.
Related ID	Identifier of the evidence record that is related to the evidence descriptor record. Each time evidence is captured, an evidence descriptor record and an evidence record are created. The evidence descriptor record contains all the information that is described in this table. The evidence record contains all the business information that is modeled for the evidence entity. The Related ID and the evidence type link the two records.
Evidence Type	Type of evidence that the evidence descriptor record relates to. For example, "income evidence", "income usage evidence". Evidence Type is a code table value. The Evidence Type and the Related ID link the evidence descriptor record to its evidence record.
Pending Removal Indicator	Indicator that is used to flag an active evidence record for logical deletion. The Pending Removal Indicator is set when active evidence record is selected for deletion by the user.

Table 11: Attributes in the Evidence Descriptor Entity (continued)

Attribute	Description
Pending Update Indicator	Indicator that is used to flag pending updates for an active evidence record. The Pending Update Indicator is set when an active evidence record is updated that results in a new evidence record. The new evidence record shares the Correction Set ID as the active evidence record, but has a status of in edit. If the Pending Update Indicator is flagged, the system supersedes the active evidence record when the in edit version is activated.
New Indicator	Indicator that is used to flag a new instance of evidence. When new evidence is created, the New Indicator is automatically flagged. The indicator is unset when the evidence is activated. When unset, it cannot be set again, as the evidence is no longer considered new.
Approval Requested Indicator	Indicator that is used to flag evidence that requires approval from a supervisor. When a user activates evidence, the system checks if approval is required. If approval is required, the Approval Requested Indicator is flagged until the supervisor approves or rejects the evidence.
Shared Instance ID	Unique identifier that is common to all evidence records shared from the same initial piece of evidence.
Shared Indicator	Indicates that the Shared Indicator evidence record is shared from another case.
Change Received Date	Indicates the date on which evidence changes were received. When a user modifies an evidence record, the date defaults to the current date on the system, that is, the date on which the modifications are saved.
Evidence Activation Date	Date on which evidence is activated. The system sets the date when evidence changes are applied.
Shared Unchanged Indicator	Indicates that the shared evidence record was not changed since the evidence record was accepted onto the case and before it was applied. The Shared Unchanged Indicator is used by the evidence broker when the evidence broker is determining whether to rebroadcast the evidence back to the original source case.
Source Case ID	Unique identifier of the case from which the evidence was shared.
External Source Case Indicator	Indicates whether the source case is from an external system.
Change Reason	Indicates the reason for correcting the record.

Attributed evidence

Attribution periods refer to the time period in which evidence is used in case eligibility and entitlement determination. The **Attributed Evidence ID** entity is used to store information that relates to the effective time period for an active evidence record.

Most evidence has business dates that are associated with it. However, often the dates are not directly used for case eligibility and entitlement determination. For example, an employment has a start and end date. However, the business requirement might be, for example, to consider the piece of evidence for case eligibility and entitlement on month-aligned or quarter-aligned dates. While business start and end dates such as an employment start and end date influence the attribution dates, the business start and end dates are not explicitly used by the case eligibility and entitlement determination process.

By storing the attributed evidence information in an entity separate from the custom evidence that might contain business dates, users can maintain business dates without having to understand how or whether the dates affect case eligibility and entitlement. It is the system, rather than the user, that performs the calculations to determine when active evidence is effective.

Attributed evidence records are not created for integrated cases. Each attributed evidence record pertains to a particular product delivery case. When evidence is activated at the integrated case level, the system creates an attributed evidence record for each product delivery case that shares the evidence record. For example, an integrated case can have three product delivery cases within it that share the income evidence for a household. Any income evidence record activated results in three separate attributed evidence records for each of the product delivery cases.

The system checks for product delivery cases within an integrated case before it creates any attributed evidence records. If there are no product delivery cases within the integrated case at the time an evidence record is activated, then the system does create attributed evidence records for the active evidence. Later, when product delivery cases are added to the integrated case, the system creates the necessary attributed evidence records.

The following table describes each of the attributes in the attributed evidence entity:

<i>Table 12: Attributes in the Attributed Evidence entity</i>	
Attribute	Description
Attributed Evidence ID	Unique identifier of the attributed evidence record. Attributed Evidence ID is the primary key that is used to identify the attributed evidence record in the system.
Evidence Descriptor ID	Identifies the related evidence descriptor record. During case eligibility and entitlement determination process, the system uses the Evidence Descriptor ID to retrieve the evidence information to be used in the eligibility and entitlement determination.
Case ID	Identifies the related product delivery case. The Case ID links the attribution period for an active evidence record to the product delivery case that uses this evidence for eligibility and entitlement determination processing.
Attributed From Date	Start date of the period over which evidence is used in case eligibility and entitlement determination. When eligibility and entitlement for a product delivery case is checked, the system retrieves evidence where the attribution periods match the eligibility and entitlement period.

<i>Table 12: Attributes in the Attributed Evidence entity (continued)</i>	
Attribute	Description
Attributed To Date	End date of the period over which evidence is used in case eligibility and entitlement determination.

Approval request and evidence approval check

Evidence includes approval checks. Approval checks provide an extra step in the evidence change process to ensure that the changes are correct.

When an evidence approval check applies to an evidence change, the case supervisor must approve or reject the change.

The evidence descriptor approval request entity and the evidence approval check entity are used to store information that relates to evidence approval checks. The evidence descriptor approval request entity is linked to the evidence descriptor entity. The evidence descriptor approval request entity is used to indicate whether an evidence descriptor record is pending approval. The Evidence Approval Check ID entity is part of the administration manager. The Evidence Approval Check ID entity is used to maintain configuration details for approval check functionality.

The following table describes each of the attributes in the evidence approval check entity:

<i>Table 13: Attributes in the Evidence Approval Check entity</i>	
Attribute	Description
Evidence Approval Check ID	Unique identifier of the evidence approval check record. The Evidence Approval Check ID is the primary key that is used to identify the evidence approval check record in the system.
Organization Unit ID	Identifier of the organization unit to which the evidence approval check percentage applies. Organization Unit ID is relevant only for evidence approval checks that are set up for an organization unit.
Username	Username of the user to whom the evidence approval check percentage applies. Username is relevant only for evidence approval checks that are set up for a user.
Position ID	Identifier of the position to which the evidence approval check percentage applies. Position ID relevant only for evidence approval checks that set up for a position.
Evidence Type	Type of evidence to which the evidence approval check percentage applies.
Approval Type	Type of evidence approval check, that is, organization, position, user, and evidence type, or evidence type, or both.

Table 13: Attributes in the Evidence Approval Check entity (continued)

Attribute	Description
Percentage	Percentage of evidence changes that the case supervisor must approve or reject. The Percentage is applied based on the evidence approval type. For example, if the approval type is an evidence type, such as income evidence, then 60 percent of all income evidence must be approved by a case supervisor before it is activated.
Comments	User comments on the evidence approval check record.
Record Status	Status of the evidence approval check record. The Record Status status is active unless the evidence metadata record was deleted, in which case the status is canceled.

Custom evidence entities

The main task in designing an evidence solution is to model the custom evidence entities. Evidence entities are used to store information for a specific evidence type.

It is also common for evidence entities to be related to each other. Examples of custom evidence entities are income and income usage. In the following example, the evidence types are related and the income evidence is the parent of income evidence usage.

Evidence types

Evidence types represent the events and circumstances that determine client eligibility and entitlement for benefits.

Not all evidence types directly influence eligibility. Some evidence information is captured for informational purposes.

Each evidence type is added as a code value to the evidence type code table. When added, an evidence type is used to link the information that is stored on the evidence descriptor table with the information that is stored on an evidence entity table. When the system checks case eligibility, the system reads the evidence type and related ID for an evidence descriptor record to locate its related evidence record.

The evidence metadata entity also includes the evidence type attribute. The evidence type attribute links the evidence entity with one or more products. For example, evidence metadata can be created for the income evidence type and linked to the income support product. When a user enters evidence for an income support case, that user can enter income evidence.

When two evidence records are related to each other, the evidence records evidence types are stored in the evidence relationship record. Evidence relationships are covered in the next section.

Evidence relationships

There are five common types of logical relationships.

Evidence entities can naturally relate to each other. The common types of logical relationships are categorized as:

- The evidence exists in isolation. The evidence has no relationship, or dependency, on any other evidence. The evidence is a stand-alone evidence and, so, contains no child evidence records.
- An evidence entity has a specific parent evidence entity and the parent evidence entity is a mandatory relationship. The child evidence cannot exist without having a parent evidence record of this type.
- An evidence entity has a single parent evidence entity. The parent evidence can be one of many parent entities, but it can contain only one parent evidence. In other words, a child evidence record of this type cannot exist without having a parent evidence record, but the type of the parent evidence can vary.

- An evidence entity has a single parent evidence entity. The parent evidence can be one of many parent entities, but the parent entity does not always have a parent evidence. The parent entity can exist in isolation and, so, the parent-child relationship is optional between the two evidence entities.
- An evidence entity has more than one type of child evidence.

An evidence entity can fulfill more than one relationship role. For example, an evidence entity can be the parent of another evidence entity and the grandparent of an extra evidence entity. In a grandparent-grandchild relationship, the 'middle' evidence entity is both a parent (of the grandchild), and a child (of the grandparent).

The evidence framework includes the evidence relationship entity. The evidence relationship entity is used to store the relationship information between evidence records. The following table describes each of the attributes in the evidence relationship entity.

Table 14: Attributes in the evidence relationship entity	
Attribute	Description
Evidence Relationship ID	Unique identifier of the evidence relationship record. Evidence Relationship ID is the primary key that is used to identify the evidence relationship record in the system.
Parent ID	Identifier of the parent evidence descriptor record. The system reads the parent evidence descriptor record to retrieve the parent evidence record.
Parent Type	Evidence type of the parent evidence record. For example, income evidence. The Parent ID and Parent Type are used to identify the parent evidence record in the evidence relationship.
Child ID	Identifier of the child evidence descriptor record. The system reads the child evidence descriptor record to retrieve the child evidence record.
Child Type	Evidence type of the child evidence record. For example, income usage evidence. The Child ID and the Child Type are used to identify the child evidence record in the evidence relationship.

The system uses relationship information to filter the evidence that is displayed in the evidence subtabs that are within the evidence object tab. When the system is accessing a child evidence object subtab, the system searches for the child evidence records that are related to the parent evidence record. There are any number of evidence records for the child evidence type, but only those evidence records that are in relationships with the parent evidence record are displayed in the subtab.

For example, a user can access the Asset Ownership evidence subtab for a parent Current Asset record evidence object tab. The system reads the evidence relationship table to retrieve only the Asset Ownership evidence records that are related to the parent Current Asset evidence record.

For more information about the evidence object tab, see the *The evidence User Interface (UI)* related link.

Related concepts

[The evidence User Interface \(UI\)](#)

Screens and views are provided with the evidence framework to allow users to capture and maintain evidence in a consistent manner.

The evidence controller

The evidence controller is responsible for most of business processing that is required to maintain evidence.

The evidence controller balances the common infrastructure that is applied across all evidence types for maintaining evidence and any parts of evidence maintenance that were customized to meet business requirements.

Common logic is provided in the evidence controller for enacting the steps in the processes that form part of the overall evidence pattern, which prevents repeating the logic across all evidence types in a custom evidence solution.

To provide a balance, the evidence controller also orchestrates the logic specific to an evidence type. The evidence controller contains methods that call evidence interface methods for the evidence types. Therefore, each custom evidence entity must implement this interface to take part in the evidence pattern.

Evidence hooks and registrar

Evidence hooks provide extension points where customized business logic can be added to an evidence processing. The registrar process works along with evidence hooks.

Evidence hooks

When evidence is being removed, the evidence controller calls an evidence hook where extended functionality can be added.

Evidence registrar

The purpose of the evidence registrar is to permit the business logic to be customized on a per product basis. Each product can register with an evidence subpattern its own hook. When a product is registered, the evidence controller enacts the extended processing for the process specific to that product.

List evidence

The list evidence process presents the user with relevant information about an evidence in the evidence list.

There are a few different list methods. One list method provides a view of all the evidence objects of a type. There are separate methods to provide lists of active objects of all types, and in edit objects of all types.

Insert evidence

The insert evidence process is used to capture evidence information for an evidence type. The result is a new evidence record with an in edit status.

Step 1

Insert a new evidence record that specifies the evidence type and pass control to the evidence controller. A user that wants to insert new evidence is presented with an insert screen that is unique to the evidence type.

Step 2

When the evidence controller creates an evidence descriptor record, the evidence descriptor record includes five characteristics for the participant to whom the evidence applies:

- The correction set ID
- The succession ID
- The status (in edit)

- The case ID
- The participant ID

For information about the evidence descriptor entity, see the *Evidence descriptor* related link.

Step 3

Step 3 occurs only if the new evidence record is a child of a parent evidence record. The evidence controller creates an evidence relationship record to acknowledge the relationship between the parent evidence record and its new child.

Step 4

Step 4 is the insertion of an entry into the evidence change history table. This is the first entry in the evidence change history as it captures the actual creation of the evidence.

Step 5

The final step is to callout to an evidence hook. The hook enacts any extra steps that are required to insert a new evidence record based on business requirements for an evidence type.

Related concepts

Evidence descriptor

The `EvidenceDescriptor` ID entity contains the information that is required to maintain a piece of evidence in the system, for example, the evidence record status and the received date.

Modify evidence

The modify evidence process allows a user to update evidence information for an active or in edit evidence record.

As with the insert evidence process, the modify evidence process specifies the evidence type and passes control to the evidence controller. The evidence controller retrieves evidence information for the evidence record from both the custom evidence entity table and the evidence descriptor table. The information is displayed to the user who wants to modify it. While most of the information retrieved from the custom evidence entity table is modifiable, the information retrieved from the evidence descriptor table cannot be modified. The exception is the evidence received date, change received date, and effective date.

When the user saves the evidence changes, the evidence controller validates the evidence that can result in warnings, errors, or both. The evidence solution provides two validations to support the approval check process that are called during an enactment of the modify evidence process. One validation is used to warn users that their modifications are being made to a piece of evidence that is awaiting approval. The second validation is used to stop a user from changing evidence that is approved and is ready for activation.

The modify evidence process continues in one of two directions. If the changes apply to active evidence, the evidence controller inserts a new evidence record that contains the modified evidence. The evidence controller labels the modified evidence as either an evidence correction or an evidence succession. For more information about evidence correction, see the *Evidence correction and succession* related link. Alternatively, if the changes apply to in edit evidence, the existing evidence record is updated and no new evidence record is created.

The evidence controller then adds an entry to the evidence changes history table. This entry captures information about the modifications that are made to the evidence record. The evidence controller completes the process of modifying evidence by calling out to an evidence hook. The hook enacts any additional steps that are required to modify the evidence based on business requirements.

Related concepts

Evidence correction and succession

The evidence pattern supports two types of evidence change: evidence correction and evidence succession.

Evidence correction and succession

The evidence pattern supports two types of evidence change: evidence correction and evidence succession.

An evidence correction is the replacement of an existing evidence record with a new evidence record to correct an incorrect piece of data. For example, an active bank account evidence record that contains an incorrect bank account number can be updated such that the new bank account number supersedes the incorrect one.

An evidence succession is the set of evidence records that collectively represents a piece of evidence as it changes over time. For example, a bank account evidence record can include a bank account balance. This bank account balance is likely to change over time and the succession of bank account balances collectively represent the changes to the bank account.

The evidence controller uses the correction ID, succession ID, and effective date attributes to manage evidence changes.

A correction set ID and succession ID are assigned to all new evidence records. The correction set ID is used to track corrections that are made to evidence; the succession ID is used to track changes in circumstance.

When a user is updating an active evidence record, the user can modify the effective date of change or else leave it the same. The effective date of change is the field that determines whether a modification to an active evidence record is a succession or a correction.

When a user is modifying evidence, if no change is made to the effective date of change field, the modification is a correction. For all evidence corrections, the system assigns the in-edit evidence record the same correction ID as the active evidence record. This ensures that the evidence corrections supersede the existing active evidence. Also, it allows for all evidence corrections to be tracked in a single evidence change history.

If the effective date of change is changed as part of modifying evidence, the modification is a change during the lifetime of the evidence and as such is a succession. To monitor a succession of updates that are made to an active evidence record, the system assigns each in edit evidence record the same succession ID, but a different correction set ID. When activated, the succession of updates does not supersede any existing active evidence.

Important: The effective date of change can be updated only for active evidence records. The evidence pattern provides validation that prevents a user from modifying the effective date of change for in edit evidence. If the user enters an incorrect effective date of change when the user is updating active evidence, the user must discard the incorrect in edit record and restart the update process.

View evidence

The view evidence process displays evidence information for an evidence record. The view evidence process is initiated when a user selects to view an evidence record in the evidence list.

The evidence controller retrieves evidence information for the evidence record from both the custom evidence entity table and the evidence descriptor table. The evidence controller also retrieves the name of the user responsible for the last modification from the evidence change history table. The evidence information is presented to the user on the view evidence screen unique to the evidence type.

Remove evidence

The remove evidence process marks an active evidence record as pending removal.

Note: Enacting the remove evidence process does not remove an active evidence record. The evidence record remains active after it is flagged as pending removal. To initiate the pending removal, the apply evidence changes process must be run.

The remove evidence process involves two steps:

1. Specifying the evidence ID to the evidence controller. The evidence controller retrieves the evidence record and sets the active evidence to pending removal. While the evidence record status remains active, the evidence record's pending removal indicator is flagged. An entry is also added to the evidence changes history table.
2. Specifying the evidence type to the evidence controller. The evidence controller calls out to an evidence hook. This hook enacts any additional steps that are required to mark the evidence as pending removal based on business requirements.

Apply evidence changes

The apply evidence changes process serves two purposes: to activate new and updated evidence, and to remove active evidence that is flagged as pending removal.

A user can start the apply evidence change process in three different ways:

- The user applies all outstanding changes.
- The user applies only their own changes.
- The user selects the specific changes that apply from the complete list of pending changes.

Both the calculate attribution period and the submit for approval process are called as part of applying evidence changes. The purpose of the calculate attribution period process is to calculate and store the period during which the newly activated evidence is used in eligibility and entitlement determination. The purpose of the submit for approval process is to determine whether an evidence change requires approval from the case supervisor and to start the processing that is approved.

At a high level, the process of applying evidence changes can be divided into stages. In the first stage, the evidence controller validates the pending evidence changes. In the second stage, the evidence controller determines whether the evidence changes require approval from the case supervisor. In the third stage, the evidence controller activates the in-edit evidence and calculates the attribution periods for the newly activated evidence. In the fourth stage, it cancels any active evidence that is pending removal. In the final stage, the eligibility and entitlement engine is called.

1. Validating evidence changes

During the first stage of applying evidence changes, the evidence controller validates the pending evidence changes.

The evidence controller uses the following three steps to validate the pending evidence changes:

1. The evidence controller calls out to a hook that checks for evidence requirements at the case level such as the minimum set of evidence records that must exist for a case. This hook can call custom validations that apply at the case level rather than at the evidence type level.
2. The evidence controller then calls all validations that are associated with applying evidence changes for the specific evidence type.
3. If any of the validations fail, an exception is thrown and the user must make the appropriate updates before the user tries to apply the changes again.

2. Checking whether evidence requires approval

During the second stage of applying evidence changes, the evidence controller checks whether the pending evidence changes require approval from the case supervisor.

To determine whether the pending evidence changes require approval from the case supervisor, the evidence controller performs five steps:

1. The evidence controller checks if manual approvals are already outstanding for the evidence by checking whether the approval status is submitted. The evidence controller does not add the evidence to the list of pending updates because it still requires approval. However, the evidence is not be sent for manual approval a second time because the case supervisor is already informed.
2. The evidence controller checks if the evidence was previously rejected. If so, the evidence controller submits the evidence for approval.

3. The evidence controller checks if the evidence was previously approved. If so, the evidence does not require approval and is thus added to the list of pending updates (ready for the next steps that are required to apply evidence changes).
4. The evidence controller checks if the evidence was previously automatically approved. If so, the evidence does not require manual approval again, so the evidence controller adds it to the list of pending updates.
5. For all other evidence, the evidence controller calls the Check for Evidence Approval API that reads the evidence approval checks table and determines whether the evidence must be manually approved. Evidence that requires approval is submitted for approval; evidence not requiring approval is added to the list of pending updates.

To set an evidence's approval status to submitted, the evidence controller performs five steps:

1. The evidence controller creates an approval request record and an evidence descriptor approval request record with the current approval request indicator set to true.
2. The evidence controller updates any previous evidence descriptor approval request records for the same evidence descriptor record by setting the current approval request indicator to false.
3. The evidence controller updates the evidence descriptor record by setting the approval request indicator to true and the approval status to submitted.
4. The evidence controller adds an entry to the evidence change history to acknowledge that the evidence is submitted for approval.
5. The evidence controller enacts the evidence approval workflow. For more information about the evidence approval workflow, see the *Submit evidence for approval workflow* related link.

Related concepts

Submit evidence for approval workflow

When the case supervisor approves or rejects a manual activity, the workflow splits and continues in one of two directions.

3. Activating evidence and calculating attribution periods

During the third stage of applying evidence changes, the evidence controller activates in edit evidence and calculates the attribution periods for the newly activated evidence.

To activate in edit evidence and calculate the attribution periods for the newly activated evidence, the evidence controller performs six steps:

1. The evidence controller changes the status of the new and updated evidence records from in edit to active and populates the evidence activation date with the current date on the system. It also searches for existing active evidence records with the same correction set ID as the newly active evidence records. If found, the evidence controller changes the status of the existing active evidence records to superseded.
2. To create attribution periods for the newly active evidence, the evidence controller initiates the calculate attribution period process by calling out to a hook. This hook retrieves the list of case IDs that require an attribution period for the active evidence. If the evidence is maintained for a stand-alone product delivery, only one case ID is returned. If evidence is maintained at the integrated case level, each product delivery case that shares the evidence must have its own attribution period. So, the case IDs for each of these product deliveries are returned.
3. The evidence controller creates a new attribution period for each of the case IDs.
4. The evidence controller searches for existing active evidence records that have the same succession ID as the newly activated evidence records. If found, the evidence controller reattributes all evidence records in the succession.
5. The evidence controller continues applying evidence changes to in edit evidence by calling out to another hook. This hook enacts any additional steps that are required to activate the in edit evidence.
6. The evidence controller adds an entry to the evidence change history table for each evidence record that is activated.

4. Removing active evidence

The fourth stage of applying evidence changes is to apply pending removal changes to active evidence.

To apply pending removal changes to active evidence, activate in edit evidence and calculate the attribution periods for the newly activated evidence, the evidence controller performs four steps:

1. The evidence controller applies the evidence changes to active evidence that is pending removal by changing the status of this evidence to canceled.
2. The evidence controller searches for existing active evidence records that have the same succession ID as the newly canceled evidence records. If found, the evidence controller reattributes all evidence records in the succession.
3. The evidence controller calls out to a hook that enacts any additional steps that are required to cancel the active evidence.
4. The evidence controller adds an entry to the evidence change history table for each evidence record that is canceled.

5. Assessing evidence changes

The last step in applying changes is to assess affected product delivery cases.

The evidence controller calls the eligibility and entitlement engine by using an eligibility and entitlement determination period that consists of the earliest attributed **From** and latest attributed **To** dates for all applied evidence.

Calculating attribution periods: additional functionality

The evidence framework provides additional functionality for calculating attribution periods. The additional functionality includes support for simulating the activation of in-edit evidence and also includes the automatic calculation of attribution periods for new product delivery cases.

Simulating the activation of in-edit evidence

As part of the manual check eligibility process, users can check eligibility by using in-edit evidence. The system simulates the activation of the in-edit evidence records by calculating virtual attribution periods for the in-edit evidence records. The system also virtually supersedes the existing active evidence records. The result is that the user is able to see the eligibility results that might be achieved by applying evidence changes to all in-edit evidence.

Automatic calculation of attribution periods

The evidence framework includes functionality that automatically reenacts the calculate attribution period process for existing active evidence to create attribution periods for the new product delivery cases. The functionality occurs when these product delivery cases are submitted.

When evidence is activated, the evidence controller creates an attribution period for each product delivery case within an integrated case that shares the evidence.

Note: Additional product delivery cases can get added to the integrated case after the evidence was activated and these new product deliveries require attribution periods for their active evidence.

Submit evidence for approval workflow

When the case supervisor approves or rejects a manual activity, the workflow splits and continues in one of two directions.

The first activity in this workflow is a manual activity. The purpose of this activity is to send a task to the case supervisor with instructions to approve or reject a piece of evidence on a case. The task includes links to the approve and reject evidence pages. The manual activity is completed when the case supervisor approves or rejects the activity.

The workflow splits and continues in one of two directions based on the outcome of the manual activity. If the evidence is approved, the next activity is the evidence approval activity; if rejected, the next activity is the evidence rejection activity. Both of these activities are route activities whose purpose is to send a

notification to the user who selected to activate the evidence. The notification informs the user of the evidence approval outcome and includes a link to the relevant evidence list screen.

Participant evidence integration

Participant data is also regarded as evidence.

A concern's date of birth, for example, is regarded as evidence. Even though such data is maintained from the Participant Manager, the date must be accessible to all cases that are required to use it as evidence. The following entities are integrated into the default application for the evidence solution.

- Address
- Alternate ID
- Alternate Name
- Bank Account
- Citizenship
- Concern Role
- Concern Role Relationship
- Education
- Employer
- Employment
- Employment Working Hour
- Foreign Residency
- Person
- Prospect Employer
- Prospect Person

Modifications to these entities automatically applies to all cases using the data and triggers eligibility and entitlement re-determination of all cases using the data.

For more information about participant evidence integration, see the *Participant evidence integration* related link.

Related concepts

Participant evidence integration

Evidence is the term that is used for data in the calculation of eligibility and entitlement. Participant data is also regarded as evidence, for example, a concern's date of birth.

Evidence generation

Evidence entities, and the relationships between them, fall into a relatively small number of high level patterns.

As the maintenance overhead on evidence code can be quite considerable, especially if the modules being maintained are large, the idea of generating evidence artefacts was initiated. The evidence generator takes input data about the entity, its relationships to other entities as well as meta-data about how it will be maintained on the client and generates the server-side code and client-side UIM, VIM files, and the associated properties and help.

For more information about the evidence generator, see the related links for *Evidence Generator specification*, *Developing dynamic evidence*, and *Modeling for the Evidence Generator*.

Related concepts

Evidence Generator specification

Use the Cúram Evidence Generator as a rapid way to develop the server side code and client side screens for evidence entities that integrate fully with the standard Cúram Evidence Solution.

Evidence Generator Cookbook

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.

Modeling for the Evidence Generator

Specific entity modeling is required when you use the Cúram Evidence Generator as the generator relies on certain, attributes, structs, and aggregations within the generated code. Use this information to learn about entity modeling that is required to use the Cúram evidence generator. The evidence generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

Evidence period calculation

The evidence period calculation algorithm completes the dates for the evidence record and its intended period of validity.

The following elements can be contained in an evidence record period:

- A start date and an end date for the evidence record.
- An effective date for the evidence record, when successive end dates are used.
- A case creation date when no other period dates exist for the evidence record.

The period for the evidence record can depend on the type of evidence that is entered and how it is configured. Some evidence types can require a change of state and other evidence types successive periods for the evidence to be recorded.

The following five steps are completed by the algorithm to establish the system-recorded period for the evidence until the valid start or effective date is established for the evidence record:

1. The evidence period start date is set to the evidence effective date.
2. The evidence period start date is set to the evidence business start date when the start date cannot be determined by the previous step.
3. The evidence period start date is set to the case start date when the start date cannot be determined by the previous steps.
4. If there is a succession to the evidence record, then the evidence period end date is set to the day before the effective date of the succeeding record.
5. The evidence period end date is set to the evidence business end date when the end date cannot be determined by the previous step.

The evidence period calculation algorithm uses the `EvidenceController` class and the `getPeriodForEvidenceRecord` method.

In certain instances, customers might not want to use the default logic. For information about using a customization hook point to override the business start date, see the *Customizing the evidence start and end dates* related link.

Related tasks

Customizing the evidence start and end dates

Where a business start date or end date is not configured on an evidence record, organizations can use a customization hook point to override the start or end dates.

Customizing the evidence start and end dates

Where a business start date or end date is not configured on an evidence record, organizations can use a customization hook point to override the start or end dates.

About this task

For customers who do want to use the default logic where the case creation date is used when no business start date is defined, a customization hook point is available. The following three use cases are examples of where displaying a case start date in the evidence period is not correct:

- An organization might be required to manage the full history of the Names evidence by using a person's date of birth. Currently, the default Names evidence does not contain a business Start Date. As a result, when a person is registered in the system, Names evidence displays the Case Start Date in the **Person Home Evidence List** page. Organizations might require the flexibility to change the date that is displayed on the Names evidence to reflect the participant's date of birth. Consequently, the organization can easily manage changes in a person's name before the person was registered in the system.
- A child evidence that does not contain a business start date. When the caseworker views the **In-Edit** or **Active Evidence** list for the child evidence, the period that is displayed to the caseworker is the Case Start Date.
- An evidence that is related to another entity, for example, an existing evidence type that is named Paid Employment, with no business start and end date. However, the evidence type that is linked to the Employment entity does not contain associated business dates. So, the Paid Employment evidence displays the Case Start Date instead of the employment start dates.

Where a business start date is not configured, the business start date can be overridden by using the EvidencePeriodHook. When the EvidencePeriodHook is used, anywhere that uses the method `getPeriodForEvidenceRecord` returns the overridden date. For example, the **Evidence List** pages and the **Incoming Evidence List** pages.

The proceeding example illustrates how the EvidencePeriodHook works. The example consists of two sample evidences:

- An Income evidence. The evidence is a parent evidence. The Income evidence contains the business start date, the business end date, employer information, and the payment frequency.
- A Tax evidence. The Tax evidence is a child of Income evidence. The Tax evidence captures a tax amount and a tax band. The Tax evidence does not capture start and end dates. Start and end dates are captured per income.

When a caseworker views the tax records from the **Evidence List** page, the tax records display the Case Start Date even though the start of the tax payment is based on the Income evidence. For the caseworkers to easily understand when the tax records start from, the Income evidence start date can be displayed when the caseworker is viewing the list of tax records. The proceeding example shows how the Case Start Date can be overridden to be the Income evidence start date.

Procedure

1. An implementation of EvidencePeriodHook class must be provided with the `getStartDate()` and `getEndDates()` methods implemented.
2. If either the `getStartDate()` or the `getEndDates()` functions are not required in the new implementation, the function must return null to preserve the default application behavior.
3. Add a Guice binding of the new implementation to a module class that is bound to the required evidence type. The module class can be a new or existing class.

Example

The following code is a sample module class with a binding for Income-Tax:

```
public class EvidencePeriodHookTestModule extends AbstractModule {
    public EvidencePeriodHookTestModule() {
        super();
    }

    @Override
    public void configure() {
        final MapBinder<String, EvidencePeriodHook> mapBinder
            = MapBinder.newMapBinder(binder(),
                String.class, EvidencePeriodHook.class);
        mapBinder.addBinding("taxEvidenceType")
            .to(IncomeTaxEvidencePeriodHookImpls.class);
    }
}
```

```

    }
}

```

The following code is a sample hook implementation:

```

public class IncomeTaxEvidencePeriodHookImpl.class implements EvidencePeriodHook{
    /**
     * This implementation returns the income evidence business start date
     */
    @Override
    public Date getStartDate(EvidenceDescriptorDtls dtls)
        throws AppException, InformationalException {

        //Tax evidence object
        TaxEvidence taxEvidenceObj =
            TaxEvidenceFactory.newInstance();

        //Income evidence object
        IncomeEvidence incomeEvidenceObj =
            IncomeEvidenceFactory.newInstance();

        //Read tax evidence details to get income evidence ID
        EvidenceCaseKey key = new EvidenceCaseKey();
        key.caseIDKey.caseID=dtls.caseID;
        key.evidenceKey.evidenceID=dtls.relatedID;
        key.evidenceKey.evType=dtls.evidenceType;
        ReadTaxEvidenceDetails readTaxEvidenceDetails =
            taxEvidenceObj.readTaxEvidence(key);

        //Read and return start date from income evidence
        EIEvidenceKey evKey = new EIEvidenceKey();
        evKey.evidenceID=readTaxEvidenceDetails.incomeEvidenceID;
        evKey.evidenceType="incomeEvidenceType";

        return incomeEvidenceObject.getStartDate(evKey);
    }
    /**
     * This implementation returns income evidence business end date
     */
    @Override
    public Date getEndDate(EvidenceDescriptorDtls dtls) throws AppException,
        InformationalException {

        //Tax evidence object
        TaxEvidence taxEvidenceObj =
            TaxEvidenceFactory.newInstance();

        //Income evidence object
        IncomeEvidence incomeEvidenceObj =
            IncomeEvidenceFactory.newInstance();

        //Read tax evidence details to get income evidence ID
        EvidenceCaseKey key = new EvidenceCaseKey();
        key.caseIDKey.caseID=dtls.caseID;
        key.evidenceKey.evidenceID=dtls.relatedID;
        key.evidenceKey.evType=dtls.evidenceType;
        ReadTaxEvidenceDetails readTaxEvidenceDetails =
            taxEvidenceObj.readTaxEvidence(key);

        //Read and return end date from income evidence
        EIEvidenceKey evKey = new EIEvidenceKey();
        evKey.evidenceID=readTaxEvidenceDetails.incomeEvidenceID;
        evKey.evidenceType="incomeEvidenceType";

        return incomeEvidenceObject.getEndDate(evKey);
    }
}

```

The evidence User Interface (UI)

Screens and views are provided with the evidence framework to allow users to capture and maintain evidence in a consistent manner.

The following are the main screens and views in the evidence framework.


Dashboard

The dashboard view provides a summary display of evidence for a case. The dashboard groups evidence by category to help a caseworker locate individual evidence types. Further information is

available including whether there is any evidence in edit, any verifications outstanding, or any issues for each evidence type. Each category offers additional flexibility for a caseworker with the following three different views of evidence:

- All the evidence types that are configured for that category on a case.
- All evidence that is recorded for the category.
- All evidence for the category that is not recorded.

EvidenceFlow

Note:  EvidenceFlow is deprecated.

The EvidenceFlow view provides an alternative summary display and navigation through evidence on a case where each evidence type is represented by a tile. When a tile, or evidence type, is in focus, then the list of evidence objects, and the successive changes to the object over time, for that evidence type are available.

Evidence type tab

The evidence type tab displays the evidence records for a particular type of evidence. Its purpose is to allow users to maintain evidence for the evidence type.

The evidence type tab is similar to active and in-edit list views, except that it is specific to one evidence type and it shows evidence records irrespective of their status. The evidence type tab provides links to view and edit individual evidence records, and links to the standard evidence object tab view.

An evidence type tab can be opened in the following three different ways:

- Selecting an evidence type on evidence list, that is, active evidence list, in-edit evidence list.
- Navigating to a child evidence type from a parent evidence record.
- Selecting the evidence name on the Dashboard.

Evidence Object tab

A tab view is provided for each evidence object that displays the latest details for the evidence and lists the successive changes to the object over time. Any additional data that relates to the evidence object is available. If the evidence is a parent, then a list of related child evidence is displayed, one list for each child evidence type. For example, income evidence is a parent of income usage evidence. A caseworker that views income evidence can view a list of income usage evidences relating to the income evidence.

If an evidence type is a child, the parent evidence is listed. If an evidence type is a grandchild, only the child evidence is displayed and not the parent evidence. The related evidences are available to one level of relationship (parent to child is one level, child to grandchild another level).

An evidence object tab can be opened by selecting an evidence description on evidence list (active evidence list, in-edit evidence list, and other evidence lists).

View evidence

View evidence is used to view evidence information that is retrieved from the evidence descriptor table. For more information about view evidence, see the *View evidence* related link.

Insert evidence

Insert evidence permits users to input specific evidence details. For more information about insert evidence, see the *Insert evidence* related link.

Modify evidence

Modify evidence permits users to view and modify evidence details. For more information about modify evidence, see the *Modify evidence* related link.

Apply evidence changes

Apply evidence changes permits users to select the evidence records to activate or remove. For more information about apply evidence changes, see the *Apply evidence changes* related link.

Validate evidence changes

The validate evidence changes screen allows a user to validate evidence changes for an evidence type. It is a pre-test of the apply evidence changes maintenance function for a specific evidence type. As evidence changes can be applied across any number of evidence types simultaneously, it can be difficult for a user to find and correct all errors that occurred. Pre-testing allows a user to test the evidence changes for just one evidence type and correct these changes before the user applies them. This screen is accessible from all main evidence screens, that is, from in-edit evidence, active evidence, dashboard, and evidenceFlow.

Evidence approval

The user that creates the evidence approval check must enter a percentage. The figure is the percentage of evidence changes that are submitted by users in the organization that require manual approval by the case supervisor. The user must also select whether the approval check percentage applies to all evidence types or to a selected evidence type.

The evidence framework provides a number of screens for maintaining evidence approvals. Evidence approval check set-up screen is one of many screens that are provided as part of system administration for setting up evidence approval checks. It is used to set up an evidence approval check for an organization unit. There are similar screens for creating evidence approval checks for users, positions, and for evidence types.

Evidence correction history

The purpose of the evidence correction history screen is to list all corrections that are made to a specific record that was modified without an 'effective from date' specified. The evidence correction history screen is accessed by selecting the **View History** option on the evidence view page.

Evidence change history

The purpose of the evidence change history screen is to list all changes that are performed on an evidence object during its lifetime. The evidence change history page is accessed from evidence object tab.

Important: The evidence change history is not limited to changes made to a single evidence record because it is designed to recognize changes that are made to the same piece of evidence. For example, when corrections are made to active evidence, the system creates a new evidence record with these corrections. Both the existing active evidence record and the evidence record with the evidence corrections are considered the same piece of evidence. They share correction set ID. So, both evidence records are monitored in the one evidence change history.

Related concepts

View evidence

The view evidence process displays evidence information for an evidence record. The view evidence process is initiated when a user selects to view an evidence record in the evidence list.

Insert evidence

The insert evidence process is used to capture evidence information for an evidence type. The result is a new evidence record with an in edit status.

Modify evidence

The modify evidence process allows a user to update evidence information for an active or in edit evidence record.

Apply evidence changes

The apply evidence changes process serves two purposes: to activate new and updated evidence, and to remove active evidence that is flagged as pending removal.

Evidence Generator specification

Use the Cúram Evidence Generator as a rapid way to develop the server side code and client side screens for evidence entities that integrate fully with the standard Cúram Evidence Solution.

Note: The generator requires that the entity is modeled with specific options set, and that certain associated structs are created according to a naming convention and with specific aggregations. For more information, see the *Modeling for the Evidence Generator* related link.

This section provides a complete reference for the following:

- Configuring the Cúram Evidence Generator
- Developing the evidence entities that use the Cúram Evidence Generator

This section also describes the patterns that can be applied at development time, the meta data required for each, and how it affects the generated output.

Related concepts

Modeling for the Evidence Generator

Specific entity modeling is required when you use the Cúram Evidence Generator as the generator relies on certain, attributes, structs, and aggregations within the generated code. Use this information to learn about entity modeling that is required to use the Cúram evidence generator. The evidence generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

Input sources for the generator

The generator relies on specific input sources to produce its outputs.

Configuration

The generator is configured by using a components `evidence.properties` file. The file provides options for, for example, setting the code package of generated code, the location of generated files, and so on.

Standard properties and message files

Each component must also provide a general message, and the following two standard properties files:

general.properties

The file provides many of the standard properties that are needed by the generator, such as various page and list titles and standard action control links. It helps to ensure a consistent experience across all evidence types under the generators control.

employment.properties

The file provides many of the standard properties that are needed by the generator when it is linked with core employments, including various page and list titles and standard action control links. The file helps to ensure a consistent experience across all evidence types under the generators control.

Ent<product.prefix>GeneralError.xml

This file provides many of the error messages that the generated code attempts to throw under certain circumstances.

Entity metadata

Each entity that is generated requires its own metadata files to be provided. These describe various features of the entity that is generated, and are roughly separated into two distinct sections.

Server side

The server side metadata is used to define various things, including the relationships between various entities, the participant the record relates to, the business dates of the entity, and any cached database operations.

Client side

The client side metadata is more concerned with the layout of the generated screens, and the text labels and descriptions that appear on them.

Outputs from the generator

The Cúram Evidence Generator produces the code, screens, and configuration files that are required for evidence types to fully integrate with the standard Cúram Evidence Solution. No further coding is required.

The generator also produces skeleton implementations of various extension points in the code to permit simple customization of the generated evidence. For example, for validations, and both pre- and post-processing for the standard Create, Read and Update methods.

Modeling

For each entity that is handled by the generator, a service layer class and various extension classes are modeled. The modeling ensures that all generated code can be accessed by using the normal application interface-factory method. In addition, a facade class is generated per component to provide access to all the Create, Read, Update, and Delete operations for those entities.

Code implementation

Implementation code is generated for all of the modeled classes that are created by the generator, and for the entity layer. The code implementation ensures that there is no requirement to write any further code.

However, there are several extension points that are generated where custom code can be easily integrated into the generated implementation code. The extension points are useful for things such as validations, evidence object descriptions.

Message files

The generator also produces a message file per entity with specific error messages that are contained within.

Screens

The generator writes all the screens that are required for creating, modifying, viewing, listing, and so on, the different evidence records. The generator also resolves scripts that are required to integrate the generated screens with the standard infrastructure screens.

Wizards

When you select to create an evidence object at runtime, often related or parent objects must be selected. In this case, the generator produces all the wizard configuration and screens that are required to take the user through this process, step-by-step.

Tabs

Each entity also has a Business Object Tab that is produced to permit the user to view all details about an evidence object, such as its change history, and any related objects.

Base directory and directory structure setup

The base directory for the configuration and metadata must be named `evidence`, and the file `evidence.properties` must exist. The base directory must contain three sub-directories: `properties`, `server metadata`, and `client metadata`.

The Cúram Evidence Generator is designed to automatically find any locations where evidence must be generated by looking for a specific directory and file structure within each component in the component order. It is essential to get the structure correct.

Note: The case of letters in the directory and file names is important and must be created exactly as specified.

Properties directory

Within the evidence directory, there must be a directory that is called `properties`. The directory is the location for the `general.properties` and `employment.properties` files.

Server metadata directory

Within the evidence directory, there must be a directory that is called `server`. Within `server` is a directory that is called `metadata`. The directory is the location for your server metadata files.

Client metadata directory

Integrated case level

Within the evidence directory, there must be a directory that is called `integrated`. Within the directory, there is a directory that is called `metadata`. The directory must be the location for your client metadata files for integrated level cases.

Product delivery case level

Within the evidence directory, there must be a directory that is called `product`. Within the directory, there is a directory called `metadata`. The directory is the location for your client metadata files for product delivery level cases.

Configuration and common page properties

You can customize different aspects of the Cúram Evidence Generator. Two common page properties are `general.properties` and `employment.properties`.

Configuration

The `evidence.properties` file permits customization of different aspects of the Cúram Evidence Generator. The customization ranges from the location of generated output files to the java code package used.

For more information about the customization options, see the *evidence.properties* related link.

general.properties

The `general.properties` file is used to specify common properties that are used on many generated pages. The properties range from page titles and list column headers to labels for common actions.

For more information about the properties' options, see the *general properties* related link.

employment.properties

The `employment.properties` file is used to specify common properties that are used on generated pages that involve related generated evidence to the core employment entity.

For more information about the properties' options, see the *employment properties* related link.

Server metadata: the entity node

The server-side metadata is used to describe the relationships between entities, and several options in respect of cached methods and the participant to associate the evidence to.

For more information about the metadata format and possible values to use, see the *Server metadata: document structure* related link.

The entity node

The root node of a server metadata XML document is the Entity node. The node contains attributes for specifying the logical name of the Entity and an attribute to specify which case participant a record must be associated with.

```

<Entity logicalName="PaidEmployment"
  relateEvidenceParticipantID="employeeCPRID"
  >
  ...
</Entity>

```

Relationships

The Relationships node is used to specify information about how the current entity relates to other evidence entities, and certain core entities. There are no attributes on this node. However, four sub-patterns can be used:

- Parent-child relationships
- Multiple mandatory parents
- Pre-association relationships
- Related relationships

Parent-child relationships

The parent-child relationship pattern describes a hierarchical relationship between two evidence entities. It is the fundamental relationship in Cúram Evidence. The relationship essentially means that the child cannot be created until a parent record is created. The relationship is enforced by the navigation within the application. The pattern can be used to create multitier, that is, generational, relationships. For example, parent-child-grandchild-greatgrandchild.

Metadata entries

The metadata for describing a parent-child relationship requires listing the possible parent or child types for each entity. In the following example, a parent-child relationship exists between Paid Employment (Parent) & Employment Address (Child).

From PaidEmployment.xml (that is, the XML to describe the Paid Employment entity):

```

<Relationships>
  <Child name="EmploymentAddress" />
</Relationships>

```

From EmploymentAddress.xml (that is, the XML to describe the Employment Address entity):

```

<Relationships>
  <Parent name="PaidEmployment"/>
</Relationships>

```

Multiple mandatory parents

The multiple mandatory parents pattern is similar to the standard parent-child pattern except that more than one parent must be specified for each.

Metadata entries

To describe a multiple mandatory parents relationship, the list of parent types must be wrapped in a MandatoryParents node.

Expanding on the example from the parent-child section, from EmploymentAddress.xml:

```

<Relationships>
  <MandatoryParents>
    <Parent name="PaidEmployment"/>
    <Parent name="AnotherParentType"/>
  </MandatoryParents>
</Relationships>

```

Pre-association relationships

The pre-association pattern is used when an association exists between two entities and the user is required to select the associated record before the user creates the record that completes the association.

Metadata entries

The metadata for this pattern is simple and involves specifying the list of entity types to be chosen as a pre-association.

```
<Relationships>
  <PreAssociation to="AnotherEntityType" />
</Relationships>
```

Related relationships

The related pattern is used to relate an evidence record to a non-evidence record. The related pattern is typically achieved by storing the unique identifier of the non-evidence record as a foreign key on the evidence entity. An example might be to relate a Paid/Self-Employment evidence record to a core Employment record. The related pattern is done by storing the unique identifier of the core Employment record on the Paid/Self employment entity.

A feature of the related pattern is that it is necessary to specify a list of case participant roles to be able to list the related records that might be selected.

Metadata entries

Taking the example of a PaidEmployment evidence type, the following metadata would be used to allow the user to choose from a list of core Employments relating to case participants of types PRIMARY or MEMBER

```
<Relationships>
  <Related to="Employment">
    <ParticipantType type="PRIMARY"/>
    <ParticipantType type="MEMBER"/>
  </Related>
</Relationships>
```

Related concepts

Server metadata: document structure

The server-side metadata is provided as a well-formed XML document, named <Entity Name>.xml.

The Business Dates pattern and override

The Business Dates pattern is used to specify which, if any, of the date fields on the entity correspond to the business start and end dates of that entity. These dates are then used in the calculation of the period for which the evidence object applies.

The generator returns these dates from the `getStartDate()` and `getEndDate()` methods that are defined on the `EvidenceInterface`.

Metadata entries

Using the `BusinessDates` node, either the `startDate`, `endDate` or both can be specified as follows:

```
<Entity logicalName="PaidEmployment" ... >
  <BusinessDates startDate="employmentStartDate"
    endDate="terminationDate"/>
</Entity>
```

Override

Use the `Override` node to override a default entity that is provided with application modules or to add a custom entity to a default product.

For more information, see the *Overriding a default evidence entity: example* related link.

Related concepts

Overriding a default evidence entity: example

To meet business requirements, customers can override the default evidence entity by, for example, editing the server-side metadata and client-side metadata.

The Cached Operation pattern and metadata entries

Use the `Cached Operation` pattern to cache SQL operations in the generated entity layer class.

The generator uses a caching pattern to implement and manage the cache. The operation must be an SQL operation on the entity because the entity class is generated. Therefore, it is not possible to implement business logic within it.

Metadata entries

In the `PaidEmployment` example, to cache the `readDetails` method, use the following code:

```
<Entity logicalName="PaidEmployment" ... >
  <CachedOperation>
    databaseRead = "read"
    operationName = "readDetails"
    returnType =
      "curam.example.evidence.entity.struct.
        PaidEmploymentDtls"
  </CachedOperation>
</Entity>
```

Client metadata

Typically, the client-side metadata is used to describe the layout of the screens that must be generated. The client metadata code specifies how to select values for case participant fields and when to create new role types for those participants.

Entity node

The root node of a client metadata EUIM document is the `Entity` node. The node contains attributes for specifying the name of the entity and the display name for use on screens. You can also specify here whether the entity can be modified.

```
<Entity name="PaidEmployment"
  displayName="Paid Employment"
  modify="Yes"
>
...
</Entity>
```

User interface (UI)

The `UserInterface` node is the containing node for all UI elements. The node includes an attribute that you can use to specify whether the create screen for the entity must contain a `Save & New` button.

```
<UserInterface saveAndNewButton="Yes">
...
</UserInterface>
```

Cluster node

As with developing in UIM, the EUIM `Cluster` node is used to group UI elements. There are a number of attributes available for specifying the number of columns, the label, and description text. There are also three special attributes, `create`, `modify` and `view`, so that the `Cluster` can be hidden, or shown on different screens, allowing a different layout to be used on create screens versus modify ones.

The proceeding code creates one `Cluster` that is visible on create and modify pages only, and is a label that is specified by the property `Cluster.Label` in the associated properties file, and a second `Cluster` that is only shown on view pages.

```
<Clusters>
  <Cluster label="Cluster.Label" create="Yes" view="No">
    ...
  </Cluster>
  <Cluster label="Cluster2.Label" create="No" modify="No">
    ...
</Clusters>
```

```

    </Cluster>
  </Clusters>

```

Field node

The Field node is used to specify an individual field on the screen. There are many attributes that can be used to control the behavior of this node, including the database columnName it is associated with and the use of blank or default values in codetable fields.

An extra attribute, *metatype*, can also be used to control the behavior of the field.

For more information about the available meta types, see the *Meta types* related link.

Using an example of an entity attribute that is called *employmentType*, that is a codetable of possible employment types, the proceeding code produces a field on screen that started with the default value from the codetable. When set on the entities create page, the codetable cannot be modified from the entities modify page.

```

<Cluster ... >
  <Field columnName="employmentType" use_default="true"
    label="Field.EmploymentType.Label" modify="No"
  />
</Cluster>

```

Related concepts

[Meta types](#)

The Evidence Generator supports seven meta types.

Case participant fields

When you use a Field node to represent a case participant attribute on the entity, a number of further metadata entries are required.

Note: The *metatype* attribute of the Field node must be set to `CASE_PARTICIPANT_SEARCH`

There are three ways of specifying a participant on screens that are created by the generator.

- **Selecting from a drop-down list**

The metadata can be used to specify what case participant types must be included in the list.

- **Searching**

The system can be searched for an appropriate participant.

- **Registering a new representation**

A new representative can be added to the system.

CaseParticipant node

The CaseParticipant node provides extra information about the field and how the information is stored. It includes options for, among other things, telling the system to create a new case participant role for the chosen participant.

SearchType node

The SearchType node is used to specify a particular type of case participant role that must be listed in the drop-down select box.

CreateCaseParticipant node

When you select a pre-existing case participant from the drop-down list, there is the option of creating a new role for that participant, if they do not already have it. For example, you might select to populate the drop-down with all case participants of type PRIMARY, MEMBER and ALIEN, and select to create a new role for the participant of type ALIEN. Then, if the participant selected was already of type ALIEN, no new role would be created. However, if the type was either PRIMARY or MEMBER, a new role is created.

Example

The following EUIM code, provides a Case Participant field onscreen where the drop-down is populated with participants of type PRIMARY and MEMBER. As the create option is turned off, the user must search the system for a participant, or to register a new representative.

```
<Cluster ... >
  <Field columnName="myCaseParticipant"
    label="Field.MyCaseParticipant.Label"
    metatype="CASE_PARTICIPANT_SEARCH"
  >
    <CaseParticipant create="No">
      <SearchType type="PRIMARY"/>
      <SearchType type="MEMBER"/>
    </CaseParticipant>
  </Field>
</Cluster>
```

The following EUIM code, builds on the first example, and now creates a role type of MEMBER if the chosen participant does not already have that role. As well as the drop-down list, the user can now search the system for a suitable participant and can specify a new representative.

```
<Cluster ... >
  <Field columnName="myCaseParticipant"
    label="Field.MyCaseParticipant.Label"
    metatype="CASE_PARTICIPANT_SEARCH"
  >
    <CaseParticipant create="Yes">
      <CreateCaseParticipant
        participantType="Person"
        roleType="MEMBER"
      />
      <SearchType type="PRIMARY"/>
      <SearchType type="MEMBER"/>
    </CaseParticipant>
  </Field>
</Cluster>
```

Adding functionality

There are a number of extension classes that can be coded, with the generator providing a default skeleton implementation for each in your source code directory. Additionally each of these classes is automatically modeled by the generator, so all follow the standard factory, interface, implementation pattern used in the application.

Customize class

The customize class provides methods that get called at specific points within the generated service layer code. You can use this to implement your own custom logic, and modify the values that are passed to and from the screen.

Class name

The class is named `Customise<Entity Name>.java`

Package name

The class is placed in the package

```
curam.<product.package>.evidence.customise.impl
```

where `product.package` is as specified in the `evidence.properties` file.

Customize methods

The proceeding customize methods are provided.

Method	Details
preCreate	Allows custom processing to be performed before the evidence insert operation happens.

Method	Details
postCreate	Allows custom processing to be performed after the evidence insert operation happens.
preModify	Allows custom processing to be performed before the evidence modifies operation happens.
postModify	Allows custom processing to be performed after the evidence modify operation happens.
preRead	Allows custom processing to be performed before the evidence read operation happens.
postRead	Allows custom processing to be performed after the evidence read operation happens.

Hook class

The hook class provides you with access to a number of infrastructure methods from the EvidenceInterface that must be implemented for each entity. Typically, you implement these methods on the entities implementation class. However, as this is now generated, the hook class must be used instead.

Class name

The class is named `<Entity Name>Hook.java`

Package name

The class is placed in the package

`curam.<product.package>.evidence.hook.impl`

where `product.package` is as specified in the `evidence.properties` file.

Methods

The proceeding methods are provided.

Method	Details
calcAttributionDatesFor Case	Returns the attribution dates for an entity.
getDetailsForListDisplay	Returns the textual description of an evidence object.

Validate class

The validate class provides methods where custom validations can be added for an entity.

Class name

The class is named `Validate<Entity Name>.java`

Package name

The class is placed in the package

`curam.<product.package>.evidence.validation.impl`

where `product.package` is as specified in the `evidence.properties` file.

Validate methods

The proceeding methods are provided.

Method	Details
preModifyValidate	Called from within the entities preModify method.
preInsertValidate	Called from within the entities preInsert method.
validate	Called from within the entities standard validate method.

Related attributes class

When you use the related entity attributes pattern, a further class is generated that provides a method for reading these related values and returning them. The method is called during the service layers read operation. The method is also called when the create screen is being loaded so that the values can also be displayed there.

Class name

The class is named `<Entity Name>RelatedEntityAttributes.java`.

Package name

The class is placed in the package

```
curam.<product.package>.evidence.relatedattribute.impl
```

where `product.package` is as specified in the `evidence.properties` file.

Validate methods

The proceeding methods are provided.

Method	Details
getRelatedEntityAttributes	Method to read any related attributes from alternative sources.

evidence.properties: explanation and sample file

The `evidence.properties` file is used to configure the generator options.

Description of properties

The proceeding provides a full list of the properties that can be set and the function of each.

Property key	Description
product.build.option	Defaults to <code>false</code> . Must be set to <code>true</code> to build or clean evidence for this product. Otherwise, the Evidence Generator ignores evidence for this product.
product.name	Insert the product name here. It is used to specify to where all generated files are output.
product.ejb.package	Code package name that is used for all <code>impl</code> layer directories in the product for which evidence is being generated.
product.prefix	The prefix is prepended to the name of all generated UIM pages and certain generated classes, such as the façade.
product.appendAltID	Flag to determine whether the primary alternate ID is appended to all Case Participant names (on generated evidence screens). Defaults to <code>false</code> .

Property key	Description
product.component.root	Root directory that specifies where generated server files are copied to.
product.evidence.build.root	Root directory that specifies where all temp generated output is copied.
evidence.properties.dir	Location for individual products properties file, which contains all product building config information. This must end with an 'evidence' directory.
properties.home	Product properties directory. Contains properties files and localized values for product-wide client screen label values.
product.webclient	Root directory that specifies where generated client pages are output to.
server.evidence	Location of EvidenceEntities.xml output that is used for server-side and infrastructure generation.
casetype.product.evidence	Name and location of EvidenceEntities.xml output for caseType 'Product' used for client-side Product Delivery evidence screen generation.
casetype.integratedCase.evidence	Name and location of EvidenceEntities.xml output for caseType 'Integrated Case' used for client-side Integrated Case evidence screen generation.
server.metadata	Location of server XML files that describe the entity's relationships, function creation, and infrastructure generation.
caseType.integratedCase.metadata	Location of EUIM files that are used for EvidenceEntities.xml generation that is used for client-side Integrated Case evidence screen generation.
caseType.product.metadata	Location of EUIM files that are used for EvidenceEntities.xml generation that is used for client-side Product Delivery evidence screen generation.

Sample file

Four properties must be set to specific values for your product. The values are listed first in this sample file. Recommended values are provided for the subsequent files as the properties mostly relate to intermediary files produced during generation. So, in most situations, no benefit is gained by customizing the properties.

Note: No line breaks in individual properties are permitted.

```
## Values Specific to your component
product.name = <Component Name>
product.prefix = <Chosen Prefix>
product.ejb.package = <Chosen Package>
product.webclient = ${sysenv.CLIENT_DIR}/components/<Component Name>
```

```
## Recommended Values
product.build.option = true
product.appendAltID = false
product.component.root =
    ${product.components.root}/${product.name}
evidence.properties.dir =
```

```

    ${product.components.root}/${product.name}/evidence
properties.home = ${evidence.properties.dir}/properties/
product.evidence.build.root =
    ${evidence.build.root}/${product.name}
server.evidence = ${product.evidence.build.root}/model/server
casetype.product.evidence =
    ${product.evidence.build.root}/model/product/
    EvidenceEntities.xml
casetype.integratedCase.evidence =
    ${product.evidence.build.root}/model/integrated/
    EvidenceEntities.xml
server.metadata =
    ${product.components.root}/${product.name}/evidence/
    server/metadata
caseType.integratedCase.metadata =
    ${product.components.root}/${product.name}/evidence/
    integrated/metadata
caseType.product.metadata =
    ${product.components.root}/${product.name}/evidence/
    product/metadata
create.employment.link = true
create.clientlist.for.employment = false

```

general.properties

The `general.properties` file contains all generic label values for the product. The generic labels consist of localized label values for all common buttons, page titles, and so on. Some generic labels permit dynamic values, that is, the name of the evidence entity the page title is describing. All properties within this file must be set.

Note: The property keys cannot be changed, added, or removed as doing so would cause errors in the running of the evidence generator.

Dynamic properties

For dynamic properties, use a dynamic placeholder to give more meaning to the operation of the dynamic property.

Dynamic property values

Dynamic properties are properties where you can add a dynamic value to a property at generation time. The feature can be useful for page titles, menu options, and so on, or anywhere that further context is useful.

Dynamic properties are achieved by using a placeholder where you want the dynamic value to be placed during the generation of the properties file. One placeholder type is supported by the evidence generator. The placeholder type is directly related to metadata tags within the EUIM files.

Note: A dynamic placeholder can be used multiple times in a property value and or a combination of different placeholders. However, the user must be aware of the relationship between these placeholders and the actual evidence metadata that the evidence generator processes. A value is substituted into the placeholder only if the metatype tag that this placeholder maps to exists in the evidence entity metadata.

<displayName>

The name of the evidence entity as it appears on-screen. The name is not the same as the physical name appears on the table in the database, as demonstrated in the following example:

physical name = PaidEmployment

display name = Paid Employment

Examples of dynamic value usage

The proceeding uses `displayName` as an example:

Using for this example the `Page.Title.EntityWorkspace` the value for this property would be entered as follows:

`Page.Title.EntityWorkspace=<displayName> Evidence`

At build time, the correct substitutions occur when the evidence generator processes the EUIM files. Using the Paid Employment evidence entity, for instance, the following property would be generated into the appropriate .properties files.

Page.Title=Paid Employment Evidence

Page title keys

The proceeding table describes the property keys for generic page title properties.

Property key	Description
Page.Title.EntityWorkspace	Title for the main page of the evidence workspace that is generated for each evidence type that is used in the evidenceFlow control.
Page.Title.ModifyEntity	Title that is used for the generated modify pages.
Page.Title.NewEntity	Title that is used for generated create pages.
Page.Title.ViewEntity	Title that is used for generated view pages.

Help.PageDescription keys

UIM pages use a property that is called Help.PageDescription to provide help for the page. The proceeding table describes the property keys that must be set to provide help for generated pages.

Property	Description
Help.PageDescription.CreateEntity	Provides help for the generated create pages.
Help.PageDescription.List.EvidenceEntities	Provides help for all generated workspace list pages.
Help.PageDescription.ModifyEntity	Provides help for the generated modify pages.
Help.PageDescription.ViewEntity	Provides help for the generated view pages.
Help.PageDescription.List.EvidenceTypeVerifications	Provides help for the generated workspace verification pages.
Help.PageDescription.List.EvidenceTypeIssues	Provides help for the generated workspace issues pages.
Help.PageDescription.List.ChangeHistory	Provides help for the generated business object tab change history pages.

Field label keys

The proceeding table displays the field label properties and their associated descriptions that are required in the general.properties file.

By creating another property of the same name but with .Help appended, each property can include an associated help property that is specified.

Property	Description
Field.Label.New	Label that is used for the New link in the actions menu on the generated evidence workspace.
Field.Label.Validate	Label that is used for the Validate link in the actions menu on the generated evidence workspace.

Page informational keys

Page informationals are warning messages that are shown on screen in response to user actions. The properties are a special case as they take the formatting options that are used in normal application message files. So, the number of parameters cannot be changed. However, the message itself can be changed. The properties required no help.

Page.Informational.NotModifiable

The warning message is displayed on the screen when a user attempts to modify an evidence entity record that was marked as not modifiable in the EUIM metadata.

The suggested value is %1s Evidence is not modifiable.

Parameter	Description
%1s	The parameter is filled with the display name of the evidence entity in question.

Static properties

Static properties include action control label keys, field label keys, list label keys, cluster keys, business object tab keys, and wizard screen description keys.

Action control label keys

The proceeding table lists the static action control properties and the properties' associated descriptions that are required in the `general.properties` file.

Each property can have an associated help property that is specified by creating another property of the same name but with `.Help` appended.

Property	Description
ActionControl.Label.Cancel	Button label that is used on multiple pages to cancel the action within that context.
ActionControl.Label.Close	Button label that is used on multiple pages to close the dialog.
ActionControl.Label.Save	Button label that is used in Create and Modify evidence entity pages to save new evidence entity.
ActionControl.Label.View	Link label that is used on to view specific evidence entity.
ActionControl.Label.SaveAndNew	Button label that is used on Create pages to save and add a new entity of this type.
ActionControl.Label.Search	Button label that is used for a Search button on various pages.
ActionControl.Label.Yes	Button label for a Yes button.
ActionControl.Label.No	Button label for a No button.
ActionControl.Label.New	Button label for a New button.
ActionControl.Label.Details	In Page Navigation link that is used on the view modal pages for an entity.
ActionControl.Label.History	In Page Navigation link used on the view correction history modal pages for an entity

Property	Description
ActionControl.Label.Back	Label for a Back button, which is used on generated wizard screens.
ActionControl.Label.Next	Label for a Next button, which is used on generated wizard screens.
ActionControl.Label.Finish	Label for a Finish button, which is used on generated wizard screens.

Field label keys

The proceeding table displays the static field label properties and their associated descriptions that are required in the `general.properties` file.

Property	Description
Field.Label.firstName	Used for the first name field when registering a new representative.
Field.Label.secondName	Used for the second name field when registering a new representative.
Field.Label.address	Used for the address field when registering a new representative.
Field.Label.areaCode	Used for the area code field when registering a new representative.
Field.Label.phoneNumber	Used for the phone number field when registering a new representative.

List label keys

The proceeding table displays the static list label properties and their associated descriptions that are required in the `general.properties` file.

No help properties are associated.

Property	Description
List.Title.Type	Used for the type of an evidence object on an evidence create wizard screen.
List.Title.Description	Used for the description of an evidence object on an evidence create wizard screen.
List.Title.Period	Used for the period of an evidence object on an evidence create wizard screen.
List.Title.Participant	Used for the participant of an evidence object on an evidence create wizard screen.

Cluster keys

The proceeding table displays the static cluster properties and their associated descriptions that are required in the `general.properties` file.

No help properties are associated.

Property	Description
Cluster.EvidenceHeader.Modify.Title	Title for the Cluster that is used to wrap the included infrastructure evidence header VIM on an entities-generated modify screen.

Business object tab keys

The proceeding static properties are used when you generate the business object tab for each evidence type.

No help properties are associated.

Property	Description
leaf.title.Home	The title of the main navigation tab on generated Business Object Tabs for each entity type.
leaf.title.ChangeHistory	The title of the change history navigation tab on generated Business Object Tabs for each entity type.
leaf.title.Verifications	The title of the Verification navigation tab on generated Business Object Tabs for each entity type.
leaf.title.Issues	The title of the Issues navigation tab on generated Business Object Tabs for each entity type.
Submenu.Title.New	The actions menu New link for any child entities. The display name of the child entity is automatically appended to the end of the property.
Submenu.Tooltip.New	The actions menu New tooltip for any child entities. The display name of the child entity is automatically appended to the end of the property.
MenuItem.Title.Edit	
MenuItem.Tooltip.Edit	
MenuItem.Title.Delete	
MenuItem.Tooltip.Delete	
MenuItem.Title.ContinueEdititing	
MenuItem.Tooltip.ContinueEdititing	
MenuItem.Title.Discard	
MenuItem.Tooltip.Discard	
MenuItem.Title.CancelDeletion	
MenuItem.Tooltip.CancelDeletion	

Wizard screen description keys

The proceeding static properties are used as default, helpful text descriptions to users on the generated create wizard select screens.

The cluster for selecting a core employment record uses a simple text property. In contrast, the cluster for selecting a parent or pre-association record uses a separate starting and ending property, that is combined with a comma delimited list of the possible types that are being listed.

For example, if the list contained records of type Paid Employment and Self Employment, the proceeding constructed description text would apply.

Property	Description
Wizard.SelectEmployment.Description	Text description for the core Employment object list cluster on the generated create wizard pages.
Wizard.SelectEvidence.Description.Start	Start of the text description for the select evidence object cluster
Wizard.SelectEvidence.Description.End	Start of the text description for the select evidence object cluster

```
<Wizard.SelectEvidence.Description.Start> Paid Employment,
Self Employment <Wizard.SelectEvidence.Description.End>
```

Sample file for dynamic and static properties

Use the general.properties sample file as a reference for dynamic and static properties.

Note: No line breaks in individual properties are permitted.

```
###
### Dynamic Values
###

### Page Titles
Page.Title.EntityWorkspace=<displayName> Evidence
Page.Title.ModifyEntity=Edit <displayName> Evidence
Page.Title.NewEntity=New <displayName> Evidence
Page.Title.ViewEntity=View <displayName> Evidence

### Page Help Descriptions
Help.PageDescription.CreateEntity=This page allows you to create
a <displayName> evidence record.
Help.PageDescription.List.EvidenceEntities=This page allows you
to view a list of the <displayName> evidence recorded
in the system.
Help.PageDescription.ModifyEntity=This page allows you to modify
a <displayName> evidence record.
Help.PageDescription.ViewEntity=This page allows you to view a
<displayName> evidence record.
Help.PageDescription.List.EvidenceTypeVerifications=This page
allows you to view a list of the <displayName> verifications
recorded in the system.
Help.PageDescription.List.EvidenceTypeIssues=This page allows
you to view a list of the <displayName> issues recorded
in the system.
Help.PageDescription.List.ChangeHistory=This page allows you to
view the change history of a <displayName> record.

### Page Informationals
Page.Informational.NotModifiable=%1s Evidence is not modifiable

### Field Labels
```

```

Field.Label.New=New
Field.Label.New.Help=Press the New button to create a new
    <displayName> evidence record.

Field.Label.Validate=Validate
Field.Label.Validate.Help=Press the New button to create a new
    <displayName> evidence record.

###
### Static Values
###

### Action Controls

ActionControl.Label.Cancel=Cancel
ActionControl.Label.Cancel.Help=Generic help message for cancel
    actions

ActionControl.Label.Close=Close
ActionControl.Label.Close.Help=Generic help message for close
    actions

ActionControl.Label.Save=Save
ActionControl.Label.Save.Help=Generic help message for save
    actions

ActionControl.Label.SaveAndNew=Save & New
ActionControl.Label.SaveAndNew.Help=The Save & New creates
    a new record from the information entered on the page and
    resets the page allowing an additional record to be created.

ActionControl.Label.View=View
ActionControl.Label.View.Help=Generic help message for View
    actions

ActionControl.Label.Search=Search
ActionControl.Label.Search.Help=Generic help message for search
    actions

ActionControl.Label.Yes=Yes
ActionControl.Label.Yes.Help=Yes

ActionControl.Label.No=No
ActionControl.Label.No.Help=No

ActionControl.Label.New=New
ActionControl.Label.New.Help=New

ActionControl.Label.Details=Details
ActionControl.Label.Details.Help=Shows details of the current
    record.

ActionControl.Label.History=History
ActionControl.Label.History.Help=Choose this to view the
    correction history of this record.

ActionControl.Label.Back=Back
ActionControl.Label.Next=Next
ActionControl.Label.Finish=Finish

### Field Labels

Field.Label.caseParticipant=Case Participant

Field.Label.participant=Participant

Field.Label.firstName=First Name

Field.Label.secondName=Surname

Field.Label.singleName=Name

Field.Label.address=Address

Field.Label.areaCode=Phone Area Code

Field.Label.phoneNumber=Phone Number

Field.Label.singleName=Name

```

```

### List Titles

List.Title.Type=Type
List.Title.Description=Description
List.Title.Period=Period
List.Title.Participant=Participant

### Page Titles

Page.Title.NewEvidenceWizard=New Evidence
Wizard.Text.SelectEmployment=Select Employment
Wizard.Title.SelectEmployment=Select Employment
Wizard.Text.SelectEvidence=Select Evidence
Wizard.Title.SelectEvidence=Select Evidence

InPageNav.Label.Verifications=Verifications
InPageNav.Label.Verifications.Help=Select this tab to view
Verifications

InPageNav.Label.Issues=Issues
InPageNav.Label.Issues.Help=Select this tab to view Issues

InPageNav.Label.Evidence=Evidence
InPageNav.Label.Evidence.Help=Select this tab to view Evidence

### Generated Tab Properties

leaf.title.Home=Home
leaf.title.ChangeHistory=Change History
leaf.title.Verifications=Verifications
leaf.title.Issues=Issues

Submenu.Title.New=New
Submenu.Tooltip.New=New

### Miscellaneous

Cluster.EvidenceHeader.Modify.Title=Change Details
Cluster.EvidenceHeader.Modify.Title.Help=Contains header details
for the evidence record.

Wizard.SelectEmployment.Description=Please select one of the
following Employments.
Wizard.SelectEvidence.Description.Start=Please select one of
Wizard.SelectEvidence.Description.End= from the following list.

```

employment.properties

The `employment.properties` file contains all generic label values for the employment pages generated. The generic label values consist of localized label values for all common buttons, page titles, and so on.

Note: The property keys cannot be changed, added or removed. By changing, adding, or removing property keys causes errors in the running of the evidence generator.

Page titles

The proceeding table shows the page title properties and their associated descriptions that are required in the `employment.properties` file.

Property key	Description
Page.Title.Delete.Emploment	Title for the delete employment confirmation page.
Page.Title.Employment	Title for the employment list page.
Page.Title.Modify.Employment	Title for the modify employment page.
Page.Title.View.Employment	Title for the view employment page.

Field labels

The proceeding table shows the field label properties and their associated descriptions that are required in the `employment.properties` file.

Each property can have an associated help property that is specified by creating another property of the same name but with `.Help` appended.

Property key	Description
Field.StaticText.CancelEmployment	Confirmation text for removing an employment.
Field.Label.Primary	Label for the field that indicates whether this is a primary employment or not.
Field.Label.Occupation	Label for the field that specifies the occupation that is associated with the employment.
Field.Label.Employer	Label for the field that specifies the name of the employer.
Field.Label.From	Label for the field that specifies the start date of the employment.
Field.Label.To	Label for the field that specifies the end date of the employment.
Container.Label.Action	Label for the Action container field on generated pages.

Action control labels

The proceeding table shows the action control label properties and their associated descriptions that are required in the `employment.properties` file.

Each property can have an associated help property that is specified, by creating another property of the same name but with `.Help` appended.

Property key	Description
ActionControl.Label.Delete	Label for a Delete button.
ActionControl.Label.Edit	Label for an Edit button.
ActionControl.Label.Employment	Label for an Employment button.

Sample `employment.properties` file

Note: No line breaks are permitted in individual properties.

```
### Field Labels
Field.StaticText.CancelEmployment=Are you sure\
you want to delete this Employment?
Field.Label.Primary=Primary
Field.Label.Occupation=Occupation
Field.Label.Employer=Employer
Field.Label.From=From
Field.Label.To=To
Container.Label.Action=Action
### Page Titles
```

```

Page.Title.Delete.Employment=Delete Employment Details
Page.Title.Employment=Employment
Page.Title.Modify.Employment=Modify Employment Details
Page.Title.View.Employment=View Employment Details
### Action Controls
ActionControl.Label.Delete=Delete
ActionControl.Label.Edit=Edit
ActionControl.Label.Employment=Add Employment

```

General error messages

The general error message file for a component must be named `Ent<product.prefix>GeneralError.xml`. The file must be located in the components messages folder.

Note: The preceding `<product.prefix>` represents the same value as specified in the property `product.prefix` in your components `evidence.properties` file.

For more information, see the *evidence.properties: explanation and sample file* related link.

ERR_FV_CREATE_PROVIDER_DETAILS_SET_NO_NAME

The error message warns of a missing name field when other details were provided for registering a new representative.

The proceeding value is suggested:

```

The %1s Name must be entered when any of the
    %1s details are entered.

```

where the argument is the case participant field that is being specified.

ERR_FV_CREATE_PROVIDER_NAME_SET_NO_ADDRESS

The error message warns when no address is specified while the user is registering a new representative.

The proceeding value is suggested:

```

The %1s Address must be entered when the
    %1s Name is entered.

```

where the argument is the case participant field that is being specified.

ERR_FV_FIELD_MUST_BE_ENTERED_WHEN_ANOTHER_FIELD_ENTERED

The error message warns when one field is specified and another isn't when the user is registering a new representative.

The proceeding is the suggested value for the error.

```

The %1s must be entered when the %2s is entered.

```

where the argument values are the two fields in question.

ERR_FV_REMOVE_RECORD_ASSOCIATED

The error message warns when discarding an evidence record when it has an associated record.

The following is the suggested value for the error.

```

This %2s record cannot be discarded as there is an
    associated %1s record.

```

where the argument values are the types of the evidence records in question.

ERR_FV_NO_PARENT_RECORD

The error message warns of a missing parent record when creating a child record.

The proceeding is the suggested value for the error.

This %1s record cannot be discarded as the
parent %2s does not exist.

where the argument values are the two evidence types in question.

ERR_FV_PARTICIPANT_EMPTY

The error message warns when no participant was chosen or a new one specified for a case participant field.

The proceeding is the suggested value for the error.

%1s Details must be provided.

where the argument value is the participant field left empty.

ERR_XFV_MORE_THAN_ONE_PART

The error message warns when more than one option is chosen for a case participant field, that is, a registered person is chosen and a new representative is specified as well.

The proceeding is the suggested value for the error.

Only one %1s can be entered. Please search for
a registered %1s or enter details for an unregistered %1s.

where the arguments are the name of the field in question.

ERR_XFV_PHONE_NUMBER

The error message warns when an incomplete phone number is provided while the user is registering a new representative.

The proceeding is the suggested value for the error.

Phone Number must be entered when Phone Area Code is entered.

ERR_XFV_PHONE_AREA_CODE

The error message warns when an incomplete phone number is provided while registering a new representative.

The proceeding is the suggested value for the error.

Phone Area Code must be entered when Phone Number is entered.

where the arguments are the name of the field in question.

ERR_XRV_CHILD_EXISTS_FOR_PARENT_TO_DISCARD

The error message warns when an attempt is made to discard a parent record that has a child record that is associated with it.

The proceeding is the suggested value for the error.

This %1c record cannot be discarded as there is a
related %2c record. To discard the %3c record,
you must first discard/remove the
related %4c record.

where the arguments are the evidence types concerned.

ERR_FV_CASEPARTICIPANT_CHANGE

The error message warns when an attempt is made to change the case participant on an evidence record.

The proceeding is the suggested value for the error.

A participant cannot be changed for this evidence.

ERR_FV_EVIDENCE_SELECTION_REQUIRED

The error message warns when no parent or per-association record was chosen on the create new evidence wizard screens.

The proceeding is the suggested value for the error.

An Evidence record must be selected.

ERR_FV_EMPLOYMENT_SELECTION_REQUIRED

The error message warns when no employment record was selected on the create new evidence wizard screens.

The proceeding is the suggested value for the error.

An Employment record must be selected.

Related concepts

[evidence.properties: explanation and sample file](#)

The `evidence.properties` file is used to configure the generator options.

Server metadata: document structure

The server-side metadata is provided as a well-formed XML document, named `<Entity Name>.xml`.

The proceeding is the full reference for the structure of the `.xml` file.

Entity node (required)

The Entity node is the root of the metadata document. The Entity node contains the proceeding basic information about the entity.

Attribute	Mandatory	Possible values	Description
logicalName	Yes	Any valid entity name	The logical name of the entity, as it appears on the database.
relateEvidenceParticipantID	No	Any valid case participant attribute from the entity	If set, <code>relateEvidenceParticipantID</code> shows the participant to be set on the <code>EvidenceDescriptor</code> record. If left blank on a top level entity, the participant field on the descriptor is set to the primary client of the associated case. If left blank on a child entity, the generator iterates up the hierarchy (Parent, Grandparent, and so on) until a suitable participant is identified.

relateEvidenceParticipantID node (required)

Attribute	Mandatory	Possible values	Description
relatedEntityAttributes	Yes	Yes/No	relatedEntityAttributes is a Yes/No attribute. It determines whether the entity has related entity attributes. Related entity attributes are considered to be any piece of data that is required that cannot be read from the entity table directly. The result of the value 'Yes' is that an additional class is created with a method stub. You must handcraft the code required to read any related entity attributes.
exposeOperation	Yes	Yes/No	exposeOperation is a Yes/No attribute. It determines whether the business process for retrieving the related entity attributes must be exposed to a facade, so generating beans for it. The bean is also be used on the create screen.

Relationships node (required)

The node is used to specify all relationship details about the entity. Entities can have 0..n relationships of type Parent, Child, Mandatory Parents, PreAssociation or Related.

Parent node (optional)

Add a Parent node for every possible parent type that the evidence entity has.

Attribute	Mandatory	Possible values	Description
name	Yes	Any valid evidence entity name.	The logical name of the parent evidence entity.

Mandatory Parents node (optional)

Where an entity has multiple parents that must all be specified, the <Parent> elements must be wrapped in an outer <MandatoryParents> element. The proceeding illustrates how the elements must be wrapped.

```
<MandatoryParents>
  <Parent name="Parent1"/>
  <Parent name="Parent2"/>
</MandatoryParents>
```

Child node (optional)

Add a Child node for every possible child type that the evidence entity has.

Attribute	Mandatory	Possible values	Description
name	Yes	Any valid evidence entity name.	The logical name of the child evidence entity.

PreAssociation node (optional)

Add a PreAssociation node where the entity must be associated with another entity before creation so that related attributes from the associated entity can be displayed on the create screen.

Attribute	Mandatory	Possible values	Description
to	Yes	Any valid entity name.	The evidence type that the entity is associated 'to'.

BusinessDates node (optional)

The Evidence Interface now defines two methods, `getStartDate` and `getEndDate`, that return the business dates of the entity. The methods `getStartDate` and `getEndDate` are used in the period calculation.

The BusinessDates node permits you to note which date attributes of the entity must be returned from these methods.

Attribute	Mandatory	Possible values	Description
startDate	No	Any valid date attribute of the entity.	The date attribute to use as the business start date for the entity.
endDate	No	Any valid date attribute of the entity.	The date attribute to use as the business end date for the entity.

Override node (optional)

Use the Override element when a customer wants to override or extend a default entity.

The relevant metadata must be copied to the custom evidence directory and, at a minimum, the element must be added.

This element must also be added where a new entity is being added to the product that is being overridden.

Note: For more information about using the Override element, see the *Overriding a default evidence entity: example* related link.

Attribute	Mandatory	Possible values	Description
newEntity	No	Yes/No	Shows if this is a new entity or not.
customize	No	Yes/No	Set to Yes if you want to override the provided <code>Customise<Entity Name></code> class.
hook	No	Yes/No	Set to Yes if you want to override the provided <code><Entity Name>Hook</code> class.
relatedAttribute	No	Yes/No	Set to Yes if you want to override the provided <code><Entity Name>RelatedEntityAttributes</code> class.
validation	No	Yes/No	Set to Yes if you want to override the provided <code>Validate<Entity Name></code> class.

CachedOperation node (optional)

Use the `CachedOperation` node to specify a database read operation to be cached by the application. You must provide the following three values:

- The name of the operation to be cached.
- The name of the database read operation.
- The fully qualified name of the return struct.

Attribute	Mandatory	Possible values	Description
operationName	Yes	Any sensible operation name.	The name of the cached operation.
databaseRead	Yes	The name of the SQL entity read to be cached.	The name of the SQL entity read to be cached.
returnType	Yes	The qualified name of the struct that is returned by the entity read.	The qualified name of the struct that is returned by the entity read.

Related concepts

Overriding a default evidence entity: example

To meet business requirements, customers can override the default evidence entity by, for example, editing the server-side metadata and client-side metadata.

Client metadata: document structure

The client-side metadata is provided as a well-formed XML document, named `<Entity Name>.euim`, along with associated properties files that can be specified in multiple locales.

The proceeding is the full reference for the structure of the .xml file.

Entity node

The proceeding table displays the entity attributes.

Attribute	Mandatory	Possible values	Description
name	Yes	Any valid entity name.	The logical name of the entity.
displayName	Yes	Any sensible string value.	The name of the entity as it is to appear on client screens. For example, an entity might have a logical name of 'PaidEmployment', but on the client screens it is better to display the name as 'Paid Employment'.
modify	No	Yes/No	This attribute shows whether the entity must be modifiable. This attribute is defaulted to Yes.

UserInterface node

The UserInterface node is the beginning of the screen layout.

Attribute	Mandatory	Possible values	Description
saveAndNewButton	No	Yes/No	Defaults to No. If set to Yes, then a Save And New button is added to the entity's create page.

Clusters node

The Clusters node contains each individual cluster.

Cluster node

The Cluster node contains information about each field that appears in the cluster. The cluster can contain any number of Field or SkipField elements in any order.

Attribute	Mandatory	Possible values	Description
Description	No	Any valid entry from the associated properties file.	The attribute maps directly to the UIM cluster description attribute.
numCols	No	Integer value.	The attribute maps directly to the UIM cluster numCols attribute. If not specified, the value defaults to 2.
label	No	Any valid entry from the associated properties file.	The attribute maps directly to the UIM cluster TITLE attribute.
create	No	Yes/No	By default, a cluster is displayed on the create page. To stop a cluster from being displayed on the create page, set this attribute to No .
modify	No	Yes/No	By default, a cluster is displayed on the modify page. To stop a cluster from being displayed on the modify page, set this attribute to No .

Attribute	Mandatory	Possible values	Description
view	No	Yes/No	By default, a cluster is displayed on the view page. To stop a cluster from being displayed on the view page, set this attribute to No .

SkipField node

The SkipField node indicates to the generator to insert a blank UIM Field in this position. The node permits greater control over the formatting of the fields in the UIM Cluster .

Field node

The Field node contains information about the attributes of a particular field on the screen.

Attribute	Mandatory	Possible values	Description
columnName	Yes	A valid attribute name.	The entity attribute name, as it appears on the database.
label	No	Any valid entry from the associated properties file.	The attribute maps directly to the UIM field label attribute.
modify	No	Yes/No/Many	By default, a field is modifiable on the entity's modify screen. By setting this attribute to No, the field is read-only on the modify screen. Case Participant fields are slightly different as they are typically not modifiable. By setting this attribute to Yes, it can be left blank on the create screen and to set it one time on the modify screen. After it is entered one time, it is read-only on the modify screen. Alternatively, by setting this attribute to Many, it can be overwritten many times on the modify screen.
use_default	No	True/False	If specified, it maps directly to the UIM field USE_DEFAULT attribute.

Attribute	Mandatory	Possible values	Description
use_blank	No	True/False	If specified, it maps directly to the UIM field USE_BLANK attribute.
notOnEntity	No	Yes/No	If set to Yes, this attribute indicates that the field is not directly mapped to an entity field. The default is No.
metatype	No	Any metatype recognized by the Evidence Generator.	Use metatype to specify additional information about an attribute, and how it must be formatted.
mandatory	No	Yes/No	Determines whether the mandatory indicator must be set on the field in the create and modify screens. The default is No..

CaseParticipant node

The CaseParticipant node contains additional information about the field that relates to case participant information that is stored in the field.

Attribute	Mandatory	Possible values	Description
create	No	Yes/No	Determines whether a case participant is to be created.
name	No	Any sensible string value.	The name refers to how the associated attribute is named. For example, the attribute might be named 'empCaseParticipantRoleID'. In this instance, the name attribute must have a value of 'emp'.
readOnly	No	Yes/No	Determines whether the case participant is 'read only'.

Attribute	Mandatory	Possible values	Description
nsStruct	No	Yes/No	When the user searches for a previously registered participant on the system, the default is to search for the Person type. To search for other types, set this to Yes. You must model your own struct with the same structure as CaseParticipantDetails and aggregate this instead. A further attribute that is called participantType must be added, which links to a codetable of participant types that must be searchable. This displays a drop-down list of participant types beside the search button, and the correct search dialog appears based on the type that is selected from this list.
singleNameField	No	Yes/No	This attribute is used when a newly registered participant for this field must have one name only rather than a first and second name. For example, if the user is registering a school.

CreateCaseParticipant node

The CreateCaseParticipant node contains information about creating a case participant. Including this node indicates that the selected participant must be registered as the specified case participant type, if they are not already registered as such.

Attribute	Mandatory	Possible values	Description
participantType	Yes	Any valid participant type.	This field provides more meta information to the generator about the type of participant.
roleType	Yes	The Java Identifier of an entry from the CaseParticipantRoleType code table.	Specifies the role that the participant must be registered on the case with.

SearchType node

When you use a case participant field, the system can provide a pre-populated, drop down list of existing case participants that can be selected from. The SearchType node, which can be specified multiple times within a CaseParticipant node, indicates which case participant role types to include in this list.

Attribute	Mandatory	Possible values	Description
type	Yes	Any valid Java Identifier from the CaseParticipantRoleType codetable.	Displays the case participant role type to list for selection.

Meta types

The Evidence Generator supports seven meta types.

PARENT_CASE_PARTICIPANT_ROLE_ID

On Child or Grandchild evidence, when you apply the meta type PARENT_CASE_PARTICIPANT_ROLE_ID to a field the following two things occur:

- The field displays as the name of the parents' associated case participant role (for example, James Smith).
- The name of the parents' associated case participant role is a link to the **Participant** home page.

EMPLOYER_CASE_PARTICIPANT_ROLE_ID

When you apply the meta type EMPLOYER_CASE_PARTICIPANT_ROLE_ID to a field, it implies that the field is storing an Employer's participant role ID in the field. The result is that the Employer's name is displayed as a link to the **Employer** home page.

CASE_PARTICIPANT_SEARCH

When you apply the meta type CASE_PARTICIPANT_SEARCH to a field, it implies that the field is storing the case participant ID of the case participant with which the evidence record is being associated. The result is that the participant's name is displayed as a link to the **Participant** home page.

CODETABLE_CODE

When you apply the meta type CODETABLE_CODE to a field, it implies that the field is storing a codetable value that is to be displayed as part of the description string that is generated by the function `StandardEvidenceInterface::getDetailsForListDisplay`. The result is that the code that is stored in the field is replaced by the description string from the codetable.

REPRESENTATIVE_LINK

When you apply the meta type REPRESENTATIVE_LINK to a field, it implies that the field is storing an ID that can be used to link to the **Representative** home page. The result is that the representative's name appears as a link to the **Representative** home page.

COMMENTS

When you apply the meta type COMMENTS to a field, it implies that the field is storing free text. The result is that the field is the full width of the screen and three rows high.

RELATED_ENTITY_ATTRIBUTE

When you apply the meta type RELATED_ENTITY_ATTRIBUTE to a field, the system indicates to the generator that the field comes from the modeled-related attributes struct rather than from the entity itself. Fields of this type are read-only.

Participant types

The CreateCaseParticipant node in the Evidence Generator supports five participant types.

Note: Select the closest match to the participant type to be created.

- Person
- Employer
- ServiceProvider

- Union
- Unknown

Developing static evidence

The Cúram Evidence Generator requires specific entity modelling.

Use this section to identify the entity modeling that is required to use the Cúram Evidence Generator. The Evidence Generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

Evidence patterns

When you design evidence entities for large modules, all entities must fit into a relatively small number of patterns. These patterns are typically governed by how an evidence type, or entity, relates to another evidence type.

These patterns are the basis of the Evidence Generator. To use the Evidence Generator to create a new evidence type, you must analyze the relationships and behaviors of your proposed evidence type. When you identify the evidence type's relationships, typically the correct choice of pattern is clear. An evidence entity can use more than one pattern.

The characteristics of each Evidence Generator pattern are explained to help you recognize them when you analyze the requirements for your own custom evidence types.

Parent-Child patterns

Use Parent-Child patterns to capture a logical relationship between two entities that meet the criteria of a parent-child relationship.

The Parent-Child pattern is the most common pattern. Use it for entities that have a parent-child relationship.

The characteristics of a Parent-Child relationship are:

- The child entity must belong to a parent, and cannot exist without a parent.
- The parent entity can have many children.

Example of when to use Parent-Child patterns

Use the Parent-Child pattern to model the relationship between income evidence and income usage evidence. An income record for the money that is received by the client must exist before you can create an income usage record for how that money was spent. As the money received might be spent to pay a number of bills, multiple income usage records can be associated with the income record.

When not to use Parent-Child patterns

Parent-Child patterns are not suitable in these instances:

- Loosely associated evidence entities.
- The evidence entity in the child role can exist without a relationship to the parent entity.

Pre-Association patterns

A Pre-Association relationship exists between two evidence entities that can live independently of each other until they are associated with each other by a caseworker.

Like Parent-Child patterns, Pre-Association patterns are commonly used.

Example of when to use Pre-Association patterns

Use Pre-Association patterns where a logical relationship exists between two records and to facilitate a caseworker in easily establishing this relationship. Before a caseworker can enter data about the main evidence type, the caseworker is prompted to select evidence as defined by this pre-association pattern. By selecting evidence, a relationship between the two evidence records is established.

For example, a client is a member of a household. The member details are captured in a household member evidence record. Later, a child is born to the household member. A household member evidence record captures the child's birth. However, the mother-child relationship must also be captured by using a household relationship evidence record. By specifying a pre-association pattern for household relationship evidence to household member evidence, when a caseworker creates a new household relationship record to capture the mother-child relationship, the caseworker is prompted to:

1. Select the household member.
2. Enter the household relationship details.

Performing the preceding steps shows that there is a logical relationship between the member and the household relationship. The member must be selected before a relationship and related member can be created.

When not to use Pre-Association patterns

Do not use Pre-Association patterns for evidence entities that are not logically related or entities that more naturally fall into a Parent-Child relationship. To record an associative relationship between an evidence entity and a non-evidence entity, use a Related pattern not a Pre-Association pattern. For more information about related patterns, see the *Related patterns* link.

Related concepts

Related patterns

Use Related patterns to show a relationship between a new evidence entity and a record that is not an evidence entity.

Related patterns

Use Related patterns to show a relationship between a new evidence entity and a record that is not an evidence entity.

Creating the evidence entity depends on the existence of the other record. Likewise, a child record requires a parent record.

When to use Related patterns

Use Related patterns to link an evidence entity to a non-evidence entity. A common example of Related patterns is where creating a paid employment record depends on the existence of a (core) employment record. A paid employment record is an evidence entity, and an employment record is part of the data that is captured for a client in the participant manager. The relationship between a paid employment record and an employment record is typically a foreign key relationship.

Search Case Participant patterns

Use the Search Case Participant patterns where creating or maintaining your new evidence entity requires that you search for any case participant.

Example of when to use Search Case Participant patterns

Use Search Case Participant patterns to associate a case participant with the evidence that is being created. For example, it might be necessary to search for the client's employer when you are creating employment evidence as the employer's case participant role identifier might be stored on the client's employment record.

Validate CallOut patterns

Use Validation CallOut patterns to add validations to generated evidence entities.

Validations for a generated evidence entity are not created by the Evidence Generator. Instead, you create validations in a separate file. Use the Validation CallOut Pattern when you design the evidence entity to ensure that it can "call out" to the file that contains the validations.

When you use the Validate CallOut pattern, the generated validate class is only generated once. Therefore, during subsequent generation the generated validate class is not overwritten. Add this class to your own software versioning control system so that you can modify it as required.

When to use Validate CallOut patterns

You must use Validate CallOut patterns if custom validations must be added to the generated evidence entity.

Page Hierarchy patterns

Use Page Hierarchy patterns when you design an evidence entity that can be associated with many different types of records.

An evidence entity can have many different relationships. Where an evidence entity is associated with many different types of record, these types of records must be accessible from the navigation bar on the evidence maintenance screens.

When to use Page Hierarchy patterns

Use Page Hierarchy patterns to generate the page hierarchy where many evidence types can be associated with an evidence entity.

The Evidence Generator

Use the Evidence Generator to produce the necessary code and screens for a working evidence entity.

You input to the Evidence Generator the files prepared by developers. The evidence type that is produced by the Evidence Generator is consistent in look and behavior with all evidence.

The Evidence Generator consists of build scripts and Extensible Stylesheet Language Transformation (XSLT) stylesheets. The XSLT stylesheets define how the information that is provided in the XML files is transformed into other formats to create all the necessary server and client data for an evidence entity.

Benefits of using the Evidence Generator

The Evidence Generator greatly reduces the repetitive work that is required to build custom evidence entities and ensures that all evidence entities that are developed comply with the evidence standards.

Designing, developing, and maintaining custom evidence entities and the screens necessary for capturing the evidence takes time. With evidence, every custom entity must implement the evidence interface. Therefore, there is repetition in the code that is used to create and maintain evidence entities.

Using the Evidence Generator makes creating evidence entities easier and saves you time. The evidence entities are also far easier to maintain. Changes to how the entities work can be made through a single change to the Evidence Generator instead of making many individual changes to all the entities.

Inputs and outputs

To use the Evidence Generator, you must provide evidence type information in four files. When you run the Evidence Generator, it produces specific server code, client screens, and online help screens.

Inputs for the Evidence Generator

Rational® Software Architect Designer Model

You must model the evidence type in Rational Software Architect Designer. You must add new evidence entity, its attributes, and its operations to the Rational Software Architect Designer Model.

XML file

Extra server-side metadata is recorded in an XML file. This additional information can include, for example, the evidence entity's relationship with a parent or child entity.

EUIM file

Extra client-side metadata is recorded in an Evidence User Interface Metadata (EUIM) file. The information that the EUIM file produces is used by the Evidence Generator to build all the screens that are needed for maintaining an evidence entity.

Properties file

The properties file contains globalized information and online help content that is required for the evidence maintenance screens.

Outputs from the Evidence Generator**Server code**

All the necessary server code is generated for the functions:

- Creating a new evidence record.
- Reading an evidence record.
- Modifying an evidence record.
- Deleting an evidence record.

The Evidence Generator also provides list functions so that lists of the evidence records can be displayed on various pages.

Client screens

The Evidence Generator produces the evidence client screens and views, which includes tab configurations and the dmx data that is necessary for evidence wizard pages. The generated client screens use the Evidence User Interface (EUI), and, so, are consistent with the existing evidence screens.

Online help screens

The Evidence Generator produces an individual online help screen for each evidence maintenance page.

Attribution periods or validations are not generated

The Evidence Generator does not produce the attribution periods or validations of an evidence type.

Attribution periods

Attribution periods are the periods of time during which a piece of evidence is used in case assessment. The Evidence Generator does not generate the code that is used for attribution periods. Instead, you must write module-specific code that calculates:

- The attribution From date.
- The attribution To date.

Validations

Evidence validations are checks that are run on a piece of evidence to ensure that the evidence meets the business requirements that are defined for the evidence type. Handcrafting the validations is more efficient than attempting to generate them. Use the Evidence Generator to generate evidence entities to "call out" to the validations you create. For more information about Validate CallOut patterns, see the *Validate CallOut patterns* related link.

Outputs from the Evidence Generator

When you run the Evidence Generator, it produces specific server code, client screens, and online help screens.

Server code

All the necessary server code is generated for the functions:

- Creating a new evidence record.
- Reading an evidence record.
- Modifying an evidence record.
- Deleting an evidence record.

The Evidence Generator also provides list functions so that lists of the evidence records can be displayed on various pages.

Client screens

The Evidence Generator produces the evidence client screens and views, which includes tab configurations and the dmX data that is necessary for evidence wizard pages. The generated client screens use the Evidence User Interface (EUI), and, so, are consistent with the existing evidence screens.

Online help screens

The Evidence Generator produces an individual online help screen for each evidence maintenance page.

Attribution periods or validations are not generated

The Evidence Generator does not produce the attribution periods or validations of an evidence type.

Attribution periods

Attribution periods are the periods of time during which a piece of evidence is used in case assessment. The Evidence Generator does not generate the code that is used for attribution periods. Instead, you must write module-specific code that calculates:

- The attribution From date.
- The attribution To date.

Validations

Evidence validations are checks that are run on a piece of evidence to ensure that the evidence meets the business requirements that are defined for the evidence type. Handcrafting the validations is more efficient than attempting to generate them. Use the Evidence Generator to generate evidence entities to "call out" to the validations you create. For more information about Validate CallOut patterns, see the *Validate CallOut patterns* related link.

Modeling for the Evidence Generator

Specific entity modeling is required when you use the Cúram Evidence Generator as the generator relies on certain, attributes, structs, and aggregations within the generated code. Use this information to learn about entity modeling that is required to use the Cúram evidence generator. The evidence generator relies on the existence of certain attributes, structs, and aggregations within the generated code. Various modeling strategies are required for the different metadata patterns available in the generator.

To model the structs, ensure that you are familiar with the information in the following three links: *Cúram Server Developer*, *Designing an evidence solution*, and *Developing with evidence*.

Related concepts

Developing evidence manually

Custom evidence solutions can be developed with Cúram Evidence. All of the evidence server-side infrastructure artifacts are available in the `curam.core.sl.infrastructure.impl` package. The evidence metadata entity contains metadata about each evidence type. This entity must be populated before evidence maintenance can proceed. Evidence maintenance functions are available in the administration application.

Related information

[Cúram Server Developer](#)

[Designing an Evidence Solution](#)

Entity modeling: entities

In addition to the normal entity modeling, specific settings are required so that the entity can work correctly with the generated code.

Code package

The code package for the entity and its associated structs must be specified in the model. For example:

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

Note: The code package must correspond with the `product.ejb.package` property. For more information about the `product.ejb.package`, see the *Asset as generated evidence: implementing a sample evidence type* related link.

Optimistic locking

Optimistic locking must be turned on at the entity level because the evidence solution, which interacts with the entity, relies on database-controlled versioning.

Required attributes

The evidence generator relies on certain attributes to run successfully.

Key field

The key field of the entity must be named *evidenceID* because it results in fewer generated entity key structs on the server side.

Required operations

The evidence generator relies on the existence of certain operations to successfully run.

Insert

The insert operation must use the stereotype insert.

Auto ID

The Auto ID setting must be turned on for the *evidenceID* to generate the unique identifier to insert records into the database. The evidence generator is configured to expect that the Auto ID setting is turned on.

Pre-data access operation

The pre-data access operation must be set to **Yes**.

modify

The modify operation must use the stereotype modify.

Pre-data access operation

The pre-data access operation must be set to **Yes**.

Optimistic locking

Optimistic locking must be set to **Yes**.

read

The read operation must use the stereotype read.

remove

The remove operation must use the stereotype remove.

Customizing a default evidence entity

To customize a default evidence entity, create an entity extension in the custom model. For more information on creating an entity extension in the custom model, see the *Cúram Server Modeling Guide*.

Code package

The code package for the extension must be specified in the model. For example:

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

Note: The code package must correspond with the `product.ejb.package` property that is configured in the default product, prepended with the text `custom.`, as in preceding example. For more information about the `product.ejb.package`, see the *Asset as generated evidence: implementing a sample evidence type* related link.

Entity modeling: required structs

Rather than creating similar or identical structs at each layer, the evidence generator uses the structs that are created at the entity layer to pass data to the façade layer.

So, it is important for the generator that certain structs are created and named with the correct naming convention. Also, extra aggregations are required under certain conditions. For more information about the conditions that apply, see the proceeding `<EntityName>EvidenceDetails` and `Read<EntityName>EvidenceDetails` sections and the *Entity modeling: build process* related link.

<EntityName>EvidenceDetails

A struct that is named `<EntityName>EvidenceDetails` must be created. This struct must have no attributes of its own, and must include the following three aggregations:

Object	Aggregation name	Multiplicity
The entity that is being modeled	dtls	1:1
core.sl.EvidenceDescriptorDetails	Descriptor	1:1
core.sl.CaseIDKey	caseIDKey	1:1

Read<EntityName>EvidenceDetails

A struct that is named `Read<EntityName>EvidenceDetails` must be created. This struct must have no attributes of its own, and must include the following two aggregations:

Object	Aggregation name	Multiplicity
The entity that is being modeled	dtls	1:1
core.sl.EvidenceDescriptorDetails	Descriptor	1:1

Related concepts

[Entity modeling: the build process](#)

No additional modeling is required beyond the entity layer because the evidence generator infers the classes that are required that are at service and façade layer.

Entity modeling: the build process

No additional modeling is required beyond the entity layer because the evidence generator infers the classes that are required that are at service and façade layer.

For the following two reasons, the evidence generator can infer the classes that are required that are at service and façade layer:

- The evidence solution provides the necessary tools for maintaining evidence records.
- The evidence generator uses a combination of the structs you created at the entity layer and a number of structs that are provided by the evidence solution.

Service layer

During the build, the generator creates a process class for each evidence entity at the service layer level. The process class that is created has at least the following three operations:

- `create<Entity Name>`
- `read<Entity Name>`
- `modify<Entity Name>`

More functions might be created to handle the more specialized scenarios, but those functions are generated by the evidence generator. Likewise, the implemented code that is required to run these functions is generated by the evidence generator.

Service layer

At the façade layer, the evidence generator generates a single process class per product that contains all the functions that are required for evidence maintenance. For each single entity, at least the following three functions are added to this process class:

- `create<Entity Name>Evidence`
- `create<Entity Name>Evidence`
- `modify<Entity Name>Evidence`

Similar to the service layer, extra functions might be created to handle the more specialized scenarios, but those functions are generated by the evidence generator.

Note: No `list<Entity Name>Evidence` function is listed in the preceding section as the generic `listEvidence` function on the evidence facade is used instead. The generic `listEvidence` function also accounts for no `list<Entity Name>` function on the service layer.

Parent-child relationships

Where an evidence entity is taking the role of a child in a parent-child relationship, additional aggregations must be specified.

Additional aggregations: <EntityName>EvidenceDetails

The `<EntityName>EvidenceDetails` struct must now also aggregate the structs that are listed in proceeding table.

Object	Aggregation name	Multiplicity
core.sl.EvidenceKey	parEvKey	1:1

Object	Aggregation name	Multiplicity
core.sl.ParentSelectDetails	selectedParent	1:1

Multiple mandatory parent relationships

Where an evidence entity is taking the role of a child with multiple mandatory parents, additional aggregations must be specified.

Additional aggregations

<EntityName>EvidenceDetails

The <EntityName>EvidenceDetails struct must now also aggregate the following struct for each of the mandatory parent types.

Object	Aggregation name	Multiplicity
core.sl.EvidenceKey	<Parent Entity Name>ParEvKey	1:1

Note: To keep with standard Java naming practices, the first letter in the preceding aggregation name must be lowercase.

Read<EntityName>EvidenceDetails

The Read<EntityName>EvidenceDetails struct must now also aggregate the following struct for each of the mandatory parent types.

Object	Aggregation Name	Multiplicity
core.sl.ParentEvidenceLink	<Parent Entity Name>ParentEvidenceLink	1:1

Note: To keep with standard Java naming practices, the first letter in the preceding aggregation name must be lowercase.

Pre-association relationships

When you use the pre-association pattern, specific additional modeling is required.

Additional aggregations

<EntityName>EvidenceDetails

The <EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Object	Aggregation name	Multiplicity
core.sl.EvidenceKey	preAssocKey	1:1

Case participant attributes

When you add a case participant attribute to the entity, further aggregations are required to permit the details be added correctly.

Additional aggregations

<EntityName>EvidenceDetails

The <EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Object	Aggregation name	Multiplicity
core.sl.CaseParticipantDetails	caseParticipantDetails	1:1

Read<EntityName>EvidenceDetails

The Read<EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Object	Aggregation name	Multiplicity
core.sl.ReadCaseParticipantDetailss	caseParticipantDetails	1:1

Additional case participant attributes

In certain circumstances, a business requirement might be to have a case participant, other than the primary case participant, stored as a piece of evidence data. For example, a piece of evidence named 'Medical Report'. In such a case, two requirements apply:

- It is necessary to store the ID of the person for whom the medical report was commissioned.
- It is necessary to store the ID of the medical practitioner who compiled the report.

Other examples of case participants are Education Faculties, Unions, or Employers.

You can flag an attribute, by using metadata, as being a special 'case participant' attribute. This means that this attribute stores the role ID of the case participant. You must provide the name attribute in the CaseParticipant element of the EUIM metadata, and use this name when aggregating the structs.

Additional aggregations

To facilitate the generator in its handling of this special flag, the two required structs must aggregate additional structs.

<EntityName>EvidenceDetails

The <EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Object	Aggregation name	Multiplicity
core.sl.CaseParticipantDetails	<name>CaseParticipant Details	1:1

Read<EntityName>EvidenceDetails

The Read<EntityName>EvidenceDetails struct must now also aggregate the proceeding structs:

Object	Aggregation name	Multiplicity
core.sl.ReadCaseParticipantDetails	<name>CaseParticipant Details	1:1

Related entity attributes

In certain circumstances, a business requirement might require that a field value from a related entity is available either to display or to use when the user is maintaining an entity.

For users, the availability of such a field value can be a helpful hint when users are entering information. Typically, the information that is used is from a parent evidence record. For example, displaying the remaining unallocated amount of an income record when a user wants to allocate this income against expenses.

Additional structs

Additional structs

In a scenario similar to the preceding scenario, an additional struct must be created at the entity layer to hold the related information.

<EntityName>RelatedEntityAttributesDetails

The <EntityName>RelatedEntityAttributesDetails struct must have, as attributes, any attribute that is to be shared between the entities. The attribute must be of the appropriate type.

Additional aggregations

Read<EntityName>EvidenceDetails

The Read<EntityName>EvidenceDetails struct must now also aggregate the proceeding structs.

Object	Aggregation name	Multiplicity
<EntityName>RelatedEntityAttributesDetails	relatedEntityAttributes	1:1

Non-evidence attributes

Where an entity uses the non-evidence details pattern, an extra struct must be modeled and aggregated into the standard evidence struct.

Additional struct that are required

<EntityName>NonEvidenceDetails

The <EntityName>NonEvidenceDetails struct must be modeled. The struct must hold all the extra attributes that are required for this entity.

Additional aggregations

<EntityName>EvidenceDetails

The <EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Table 15: Additional aggregations		
Object	Aggregation name	Multiplicity
<EntityName>NonEvidenceDetails	nonEvidenceDetails	1:1

Read<EntityName>EvidenceDetails

The Read<EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Table 16: Additional aggregations		
Object	Aggregation name	Multiplicity
<EntityName>NonEvidenceDetails	nonEvidenceDetails	1:1

Non-modifiable entities

Where you require that the entity cannot be modified, additional modeling is required.

Additional struct that are required

Additional aggregations

Read<EntityName>EvidenceDetails

The Read<EntityName>EvidenceDetails struct must now also aggregate the proceeding struct.

Table 17: Additional aggregations		
Object	Aggregation name	Multiplicity
curam.core.sl.infrastructure.struct.ECWarningsDetailsList	warnings	1:1

Developing custom evidence

Custom evidence solutions can be developed with Cúram evidence.

All the evidence server-side infrastructure artifacts are available in the `curam.core.sl.infrastructure.impl` package. The evidence metadata entity contains metadata about each evidence type. The entity must be populated before evidence maintenance can proceed. Evidence maintenance functions are available in the administration application.

Developing evidence manually

Custom evidence solutions can be developed with Cúram Evidence. All of the evidence server-side infrastructure artifacts are available in the `curam.core.sl.infrastructure.impl` package. The evidence metadata entity contains metadata about each evidence type. This entity must be populated before evidence maintenance can proceed. Evidence maintenance functions are available in the administration application.

Use Cúram's Evidence framework to design and implement evidence solutions. Before you design or implement evidence solution, ensure that you are familiar with the information in the *Evidence patterns* related link.

Related concepts

[Evidence patterns](#)

When you design evidence entities for large modules, all entities must fit into a relatively small number of patterns. These patterns are typically governed by how an evidence type, or entity, relates to another evidence type.

Evidence components: server-side artifacts

All the evidence server-side infrastructure artifacts are shipped in the `curam.core.sl.infrastructure.impl` package.

The key elements in the `curam.core.sl.infrastructure.impl` include the Evidence Controller Hook classes and the Evidence Interfaces. For more information, see the *Evidence Controller Hook* and the *Standard Evidence Interface* related links.

The interfaces are part of the interface hierarchy. Both the Participant Evidence Interface and the Evidence Interface extend the Standard Evidence Interface, which is the parent interface. Each evidence entity must implement the evidence interface artifacts.

Related concepts

[Evidence Controller Hook](#)

The Evidence Controller Hook is the evidence infrastructure class that contains the extension points for the evidence maintenance pattern.

[Standard Evidence Interface](#)

The Standard Evidence Interface defines the following methods, which are common to both inheriting interfaces. The interface and its associated methods are shown with the appropriate Javadoc comments.

Standard Evidence Interface

The Standard Evidence Interface defines the following methods, which are common to both inheriting interfaces. The interface and its associated methods are shown with the appropriate Javadoc comments.

```
/*
 * Copyright 2005-2006,2011 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.core.sl.infrastructure.impl;

import curam.core.sl.infrastructure.entity.struct
    .AttributedDateDetails;
import curam.core.sl.infrastructure.struct.EIEvidenceKey;
import curam.core.sl.infrastructure.struct.EIEvidenceKeyList;
import
    curam.core.sl.infrastructure.struct.EIFieldsForListDisplayDtls;
import curam.core.sl.infrastructure.struct.ValidateMode;
import curam.core.struct.CaseKey;
import curam.util.exception.AppException;
import curam.util.exception.InformationalException;
import curam.util.type.Date;

/**
 * This interface is a key component of the Curam
 * Evidence Solution. Implementations hoping to manage evidence
 * via the Evidence Solution must ensure that the
 * evidence entities contained within the solution implement the
 * Evidence Interface. By doing this, the evidence is utilizing
 * the Evidence Controller pattern whereby a lot of the common
 * business functions for maintaining evidence are contained
 * within the out-of-the-box evidence infrastructure.
 *
 * This interface is the super interface that will be
 * extended by other evidence interfaces that wish to provide
 * custom functionality for that type of evidence. The methods
 * defined on this evidence are common to any interface that
 * extends it.
 */
public interface StandardEvidenceInterface {

    // -----
    /**
     * Method for calculating case attribution dates. The
     * calculation of evidence attribution is an integral part of a
     * evidence solution as it determines the period of
     * time for which a piece of evidence is effective. The
     * implementation of this function will contain the logic that
     * derives the appropriate effective period for the evidence of
     * a particular type.
     *
     * @param caseKey
     *     Contains a case identifier
     * @param evKey
     *     Contains the evidenceID / evidenceType pairing of
     *     the evidence to be attributed
     *
     * @return Case attribution details
     */
    AttributedDateDetails calcAttributionDatesForCase(
        CaseKey caseKey, EIEvidenceKey evKey)
        throws AppException, InformationalException;

    // -----
    /**
     * Retrieves a summary of evidence details which are used to
     * populate the 'Details' column on the following evidence
     * pages:
     *
     * - All evidence workspace pages
     * - Apply changes page
     */
}
```

```

* - Apply user changes page
* - Approve page
* - Reject page
*
* @param key
*         Contains an evidenceID / evidenceType pairing
*
* @return A summary of the evidence details to be displayed
*/
EIFieldsForListDisplayDtls getDetailsForListDisplay(
    EIEvidenceKey key)
    throws AppException, InformationalException;

// -----
/**
 * Method to get the business end date for this evidence
 * record.
 *
 * @param key
 *         Contains an evidenceID / evidenceType pairing
 *
 * @return The end date for this evidence
 */
Date getEndDate(EIEvidenceKey evKey) throws AppException,
    InformationalException;

// -----
/**
 * Method to get the business start date for this evidence
 * record.
 *
 * @param key
 *         Contains an evidenceID / evidenceType pairing
 *
 * @return The start date for this evidence
 */
Date getStartDate(EIEvidenceKey evKey) throws AppException,
    InformationalException;

// -----
/**
 * Method for inserting case evidence.
 *
 * @param dtls
 *         Custom evidence details to be inserted
 * @param parentKey
 *         Contains the evidence type of the evidence being
 *         inserted
 *
 * @return Contains the evidenceID / evidenceType of the
 *         evidence inserted
 */
EIEvidenceKey insertEvidence(
    Object dtls, EIEvidenceKey parentKey)
    throws AppException, InformationalException;

// -----
/**
 * Method for inserting case evidence on modification. An
 * insert on modification takes place when the record being
 * modified is 'Active'.
 *
 * @param dtls
 *         Evidence details to be inserted
 * @param origKey
 *         Contains the evidenceID / evidenceType pairing of
 *         the evidence being modified
 * @param parentKey
 *         Contains the evidence type of the evidence to be
 *         inserted
 *
 * @return Contains the evidenceID / evidenceType of the
 *         evidence inserted
 */
EIEvidenceKey insertEvidenceOnModify(Object dtls,
    EIEvidenceKey origKey, EIEvidenceKey parentKey)
    throws AppException, InformationalException;

// -----
/**
 * Method for modifying case evidence. This function is called
 * when 'In Edit' evidence is being modified in place.

```

```

*
* @param key
*         Contains the evidenceID / evidenceType pairing of
*         the evidence to be modified
* @param dtls
*         Modified evidence details
*/
void modifyEvidence(EIEvidenceKey key, Object dtls)
    throws AppException, InformationalException;

// -----
/**
 * Method for retrieving all child evidence for a specified
 * parent
 *
 * @param key
 *         Contains a parent evidenceID / evidenceType pairing
 *
 * @return List of all child evidence (evidenceID /
 *         evidenceType pairings) for the specified parent
 */
EIEvidenceKeyList readAllByParentID(EIEvidenceKey key)
    throws AppException, InformationalException;

// -----
/**
 * Method for reading case evidence.
 *
 * @param key
 *         Contains the evidenceID / evidenceType pairing of
 *         the evidence to be read
 *
 * @return Custom evidence details
 */
Object readEvidence(EIEvidenceKey key)
    throws AppException, InformationalException;

// -----
/**
 * Method for retrieving the list of evidence to be used in
 * the validation procedure. This is based on the evidenceID /
 * evidenceType pairing passed into this function.
 *
 * If the input evidence key was that of parent evidence, then
 * this function should return the parent and its associated
 * 'Active' and 'In Edit' child evidence records, if they
 * exist.
 *
 * @param evKey
 *         Contains the evidenceID / evidenceType pairing of
 *         the evidence being "acted upon".
 *
 * @return List of evidenceID / evidenceType pairings to be
 *         used in the validation procedure
 */
EIEvidenceKeyList selectForValidation(EIEvidenceKey evKey)
    throws AppException, InformationalException;

// -----
/**
 * Method for validating evidences based on the validate mode
 * setting.
 *
 * @param evKey
 *         The evidenceID / evidenceType pairing of the
 *         evidence being "acted upon"
 * @param evKeyList
 *         The evidence hierarchy structure for the evKey
 *         parameter. If the evKey identified the parent
 *         evidence, the evKeyList may contain this parent and
 *         its relevant children for validation purposes
 *
 * @param mode
 *         The validation mode (insert, modify,
 *         validateChanges, applyChanges)
 */
void validate(EIEvidenceKey evKey, EIEvidenceKeyList evKeyList,
    ValidateMode mode)
    throws AppException, InformationalException;
}

```

Evidence Interface

The Evidence Interface and its associated methods are shown with the appropriate Javadoc comments.

```
/*
 * Copyright 2005-2007 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.core.sl.infrastructure.impl;

import curam.core.sl.infrastructure.struct
    .AttributedDateDetails;
import curam.core.struct.CaseHeaderKey;
import curam.util.exception.AppException;
import curam.util.exception.InformationalException;

/**
 * This interface extends the StandardEvidenceInterface,
 * therefore any class that implements EvidenceInterface
 * must provide its own implementations of the methods
 * defined in the standard interface. Any methods specific
 * to "classic" (i.e. not participant) evidence are to be
 * defined in this interface.
 */
public interface EvidenceInterface
    extends StandardEvidenceInterface {

    // -----
    /**
     * Transfers evidence from one case to another.
     *
     * @param details
     *     Contains the evidenceID / evidenceType pairings of
     *     the evidence to be transferred and the transferred
     * @param fromCaseKey
     *     The case from which the evidence is being
     *     transferred
     * @param toCaseKey
     *     The case to which the evidence is being
     *     transferred
     */
    void transferEvidence(EvidenceTransferDetails details,
        CaseHeaderKey fromCaseKey, CaseHeaderKey toCaseKey)
        throws AppException, InformationalException;
}
```

Participant Evidence Interface

The Participant Evidence Interface and its associated methods are shown with the appropriate Javadoc comments.

```
/*
 * Copyright 2007 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.core.sl.infrastructure.impl;

import java.util.ArrayList;

import curam.core.sl.infrastructure.struct.EIEvidenceKey;
import curam.core.sl.infrastructure.struct.EIEvidenceKeyList;
import curam.core.sl.struct.ConcernRoleIDKey;
import curam.util.exception.AppException;
import curam.util.exception.InformationalException;
```

```

/**
 * This interface extends the StandardEvidenceInterface therefore
 * any class that implements ParticipantEvidenceInterface must
 * provide its own implementations of the methods defined in the
 * standard interface. Any methods specific to participant
 * evidence be defined in this interface.
 */
public interface ParticipantEvidenceInterface
    extends StandardEvidenceInterface {

    // -----
    /**
     * Method to check if the attributes that changed during a
     * modify require reassessment to be run when they are applied.
     *
     * @param attributesChanged
     *      - A list of Strings. Each represents the name of an
     *        attribute that changed
     *
     * @return true if Reassessment required
     */
    boolean checkForReassessment(ArrayList attributesChanged)
        throws AppException, InformationalException;

    // -----
    /**
     * Method for creating the snapshot record related to a
     * participant evidence record.
     *
     * @param key
     *      Contains an evidenceID / evidenceType pairing
     *
     * @return The uniqueID and the evidence type of the Snapshot
     *         record.
     */
    EIEvidenceKey createSnapshot(EIEvidenceKey key)
        throws AppException, InformationalException;

    // -----
    /**
     * Method to compare attributes on two records of the same
     * entity type. It then returns an ArrayList of strings with
     * the names of each attribute that was different between them.
     *
     * @param key
     *      - Contains an evidenceID / evidenceType pairing
     * @param dtls
     *      - a struct of the same type as the key containing
     *        the attributes to be compared against
     *
     * @return A list of Strings. Each represents an attribute name
     *         that differed.
     */
    ArrayList getChangedAttributeList(
        EIEvidenceKey key, Object dtls)
        throws AppException, InformationalException;

    // -----
    /**
     * Method to search for records on a participant entity by
     * concernRoleID and status.
     *
     * @param key
     *      - The unique concernRoleID of the participant.
     *
     * @return A list of EIEvidenceKey objects each containing an
     *         evidenceID/evidenceType pair.
     */
    EIEvidenceKeyList readAllByConcernRoleID(ConcernRoleIDKey key)
        throws AppException, InformationalException;

    // -----
    /**
     * Method removing participant evidence. This method is called
     * when participant evidence is being canceled
     *
     * @param key
     *      - Contains an evidenceID / evidenceType pairing
     * @param dtls
     *      - Modified evidence details
     */
    void removeEvidence(EIEvidenceKey key, Object dtls)

```

```
throws ApplicationException, InformationalException;  
}
```

Adopting an interface approach enforces a pattern upon entity design and development as each entity must implement the same interface. This approach allows the IBM Cúram Social Program Management Platform to provide as much common functionality as possible so that custom implementations can concentrate more on business aspects of evidence maintenance, such as validations. Each evidence entity must implement the Evidence Interface to have access to the Evidence Controller class. This class implements the common business logic across all evidence entities and the custom business logic specific to each evidence entity.

Accessing non-modeled functions

When the Evidence Interfaces are implemented by evidence entities, the methods that are defined by these interfaces are implemented by those evidence entities.

As the methods are non-modeled, the methods exist only on the evidence entity `imp` classes. To access the non-modeled functions, you must cast from the `imp` class. For more information, see the *List evidence* related link.

For the casting mechanism to work, the factory class must extend the `imp` class rather than to the base class. For the factory class to extend the `imp` rather than to the base class, if no non-stereotyped functions are being added to the class, is to add a non-stereotyped dummy function. If a non-stereotyped dummy function is not added, a runtime error results when the casting is run.

Related concepts

[List evidence](#)

A list evidence operation involves client and server development. The list operation is used to populate an evidence workspace page.

Evidence components: client-side artifacts

The client-side infrastructure artifacts are located inside the `\webclient\components\core\Evidence Infrastructure` directory.

The `\webclient\components\core\Evidence Infrastructure` folder primarily contains `uim` and `vim` client pages. The `vim` files are typically included inside solution-specific `uim` pages to manage generic evidence details. The `vim` pages contain complete default functions for evidence maintenance.

Benefits of the .im files

The key benefit of the `.im` files is that the file can be changed to match with any enhancements that are made to the evidence maintenance solution without affecting specific implementations. So, the upgrade is seamless.

The following four files are examples of `infrastructural.vim` files:

- `Evidence_createHeader.vim`
- `Evidence_modifyHeader.vim`
- `Evidence_viewHeader.vim`
- `Evidence_viewHeaderForModal.vim`

Managing infrastructural attributes

The proceeding artifacts are used to manage the infrastructural attributes of evidence maintenance and must be included in the create, modify, and view evidence pages. The following three files are further examples of `vim` files to include.

- `Evidence_typeWorkspace.vim`
- `Evidence_workspaceInEditHighLevelView.vim`
- `Evidence_workspaceActiveHighLevelView.vim`

The preceding artifacts are used to populate evidence workspaces. An evidence workspace is a central location for managing evidence. The preceding vim files are included by the workspace .uim pages.

The proceeding three files are examples of infrastructural uim pages that provide entire evidence maintenance functions:

- Evidence_applyChanges1.uim
- Evidence_addNewEvidence.uim
- Evidence_dashboard.uim

Evidence_applyChanges1 lists all work-in-progress evidence, that is, all new and updated evidence or evidence that is pending removal. The display and action bean on the page live on the Evidence facade that is part of the centralized evidence maintenance functions.

Evidence_addNewEvidence lists all possible evidence types, which are filtered by category, and starts an appropriate create page for each.

Evidence_dashboard lists all evidence types on the case and is broken into categories. It highlights that types have In Edit evidence that is recorded and that have verifications or issues outstanding.

Note: In some cases, .vim files in the client infrastructure package are included in infrastructure pages. For instance, Evidence_dashboardView.vim is included inside the **Evidence_dashboard** page and Evidence_flowView.vim is included inside the **Evidence_flow** page.

Developing an evidence solution

Developing an evidence solution can involve various steps, such as creating, modifying, reading, and listing evidence maintenance operations, evidence attribution and reattribution, registering evidence implementations, and customizing evidence.

Administration: Evidence Metadata entity and Product Evidence Link entity

The Evidence Metadata entity contains metadata information that relates to each evidence type. The Product Evidence Link entity links evidence to a product.

Evidence Metadata entity

The Evidence Metadata entity must be populated before evidence maintenance can proceed. A number of evidence page names, including the view and modify page names, are included in the metadata. The page names are retrieved at run time by using evidence infrastructure resolve scripts and by using implementations of the Evidence Type interface on the server. The records on the Evidence Metadata entity are effective dated to facilitate pages that change over time, for example, due to legislation.

Product Evidence Link entity

In some circumstances, evidence might be stored at the Integrated Case level but only some of the evidence might apply to a product on the Integrated Case. To determine the evidence to attribute to a product, a lookup of this entity is performed as part of the attribution processing. Then, only evidence that is linked to the product is attributed.

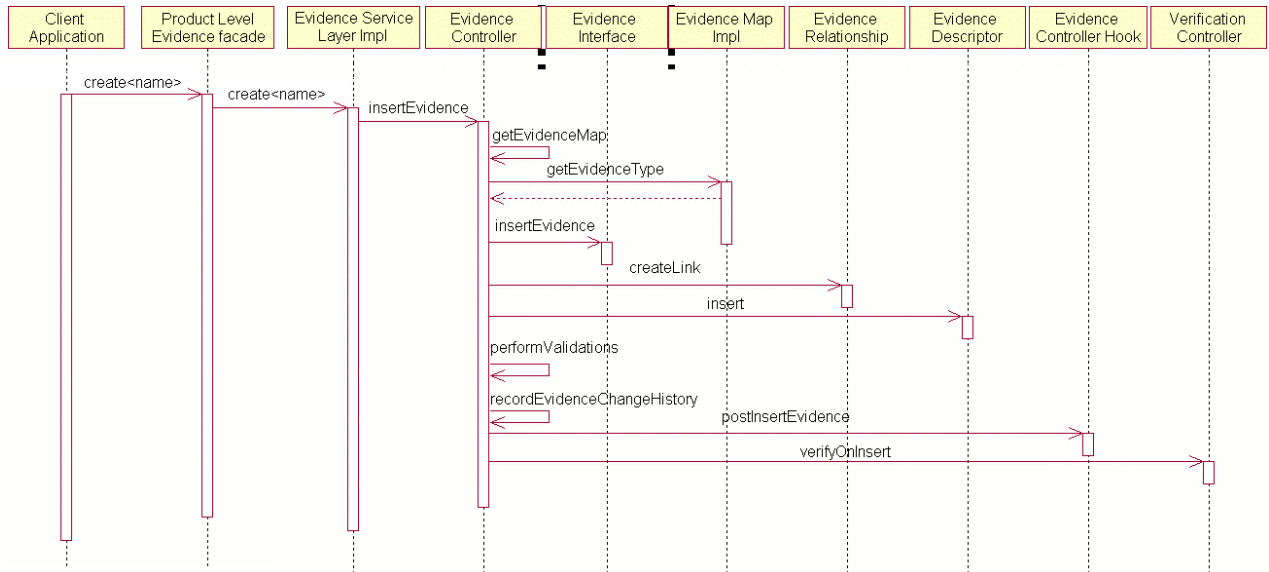
Create, modify, read, and list evidence maintenance operations

Create, modify, read, and list evidence maintenance operations are outlined by using sequence diagrams, client screen captures, and server code snippets from a sample product implementation.

Create evidence

A create evidence operation involves client and server development.

Sequence diagram for creating evidence



Client screen to be developed

The client page to be developed must include the evidence infrastructure page `Evidence_createHeader.vim`. The included `.vim` page facilitates the management of infrastructure attributes. For example, the Evidence Descriptor's `receivedDate` attribute is managed through this infrastructure page. If, in the future, more attributes that need to be managed through the create function are added to the Evidence Descriptor entity, then these attributes might be mapped through this infrastructure page. So, the operation requires just a once-off infrastructure change rather than many changes to custom artifacts.

Server methods to implement

The `SEGEvidenceMaintenance.createAssetEvidence` facade operation calls the evidence service layer implementation.

```

// -----
/**
 * Creates an Asset evidence record.
 *
 * @param dtls Details of the new evidence record to be created.
 *
 * @return The details of the created record.
 */
public ReturnEvidenceDetails createAssetEvidence(
    AssetEvidenceDetails dtls)
    throws AppException, InformationalException {

    // set the informational manager for the transaction
    TransactionInfo.setInformationalManager();

    // Asset evidence manipulation object
    Asset evidenceObj = AssetFactory.newInstance();

    // return object
    ReturnEvidenceDetails createdEvidenceDetails =
        new ReturnEvidenceDetails();

    // create the Asset record and populate the return details
    createdEvidenceDetails =
        evidenceObj.createAssetEvidence(dtls);

    createdEvidenceDetails.warnings =
        EvidenceControllerFactory.newInstance().getWarnings();
}
  
```

```

    return createdEvidenceDetails;
}

```

These overloaded `Asset.createAssetEvidence` service layer operations call the Evidence Controller infrastructure function for inserting evidence.

```

// -----
/**
 * Creates an Asset record.
 *
 * @param dtls Contains Asset evidence record creation details.
 *
 * @return the new evidence ID and warnings.
 */
public ReturnEvidenceDetails createAssetEvidence(
    AssetEvidenceDetails dtls)
    throws AppException, InformationalException {

    return createAssetEvidence(dtls, null, null, false);
}

// -----
/**
 * Creates a Asset record.
 *
 * @param dtls Contains Asset evidence record creation details.
 *
 * @param sourceEvidenceDescriptorDtls If this function is called
 * during evidence sharing, this parameter will be non-null and
 * it represents the header of the evidence record being shared
 * (i.e. the source evidence record)
 *
 * @param targetCase If this function is called during evidence
 * sharing, this parameter will be non-null and it represents the
 * case the evidence is being shared with.
 *
 * @param sharingInd A flag to determine if the function is
 * called in evidence sharing mode. If false, the function is
 * being called as part of a regular create.
 *
 * @return the new evidence ID and warnings.
 */
public ReturnEvidenceDetails createAssetEvidence(
    AssetEvidenceDetails dtls,
    EvidenceDescriptorDtls sourceEvidenceDescriptorDtls,
    CaseHeaderDtls targetCase, boolean sharingInd)
    throws AppException, InformationalException {

    // validate the mandatory fields
    validateMandatoryDetails(dtls);

    EvidenceControllerInterface evidenceControllerObj =
        (EvidenceControllerInterface)
            EvidenceControllerFactory.newInstance();
    EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls =
        new EvidenceDescriptorInsertDtls();

    ReturnEvidenceDetails createdEvidence =
        new ReturnEvidenceDetails();

    if (sharingInd) {
        EvidenceDescriptorDtls sharedDescriptorDtls =
            evidenceControllerObj.shareEvidence(
                sourceEvidenceDescriptorDtls,
                targetCase);

        // Return the evidence ID and warnings
        createdEvidence.evidenceKey.evidenceID =
            sharedDescriptorDtls.relatedID;
        createdEvidence.evidenceKey.evType =
            sharedDescriptorDtls.evidenceType;
    } else {
        // As there is no participant associated with this evidence
        // we must retrieve the case participant to set the evidence
        // descriptor participant.
        CaseHeaderKey caseHeaderKey = new CaseHeaderKey();

```

```

caseHeaderKey.caseID = dtls.caseIDKey.caseID;
evidenceDescriptorInsertDtls.participantID =
    CaseHeaderFactory.newInstance().readCaseParticipantDetails(
        caseHeaderKey).concernRoleID;

// Evidence descriptor details
evidenceDescriptorInsertDtls.caseID = dtls.caseIDKey.caseID;
evidenceDescriptorInsertDtls.evidenceType =
    CASEEVIDENCE.ASSET;
evidenceDescriptorInsertDtls.receivedDate =
    dtls.descriptor.receivedDate;

// Upon creation, the change reason should be Initial
evidenceDescriptorInsertDtls.changeReason =
    EVIDENCECHANGEREASON.INITIAL;

// Evidence Interface details
EIEvidenceInsertDtls eiEvidenceInsertDtls =
    new EIEvidenceInsertDtls();
eiEvidenceInsertDtls.descriptor.assign(
    evidenceDescriptorInsertDtls);
eiEvidenceInsertDtls.evidenceObject = dtls.dtls;

// Insert the evidence
EIEvidenceKey eiEvidenceKey =
    evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);

// Return the evidence ID and warnings
createdEvidence.evidenceKey.evidenceID =
    eiEvidenceKey.evidenceID;
createdEvidence.evidenceKey.evType =
    eiEvidenceKey.evidenceType;
createdEvidence.warnings =
    evidenceControllerObj.getWarnings();
}

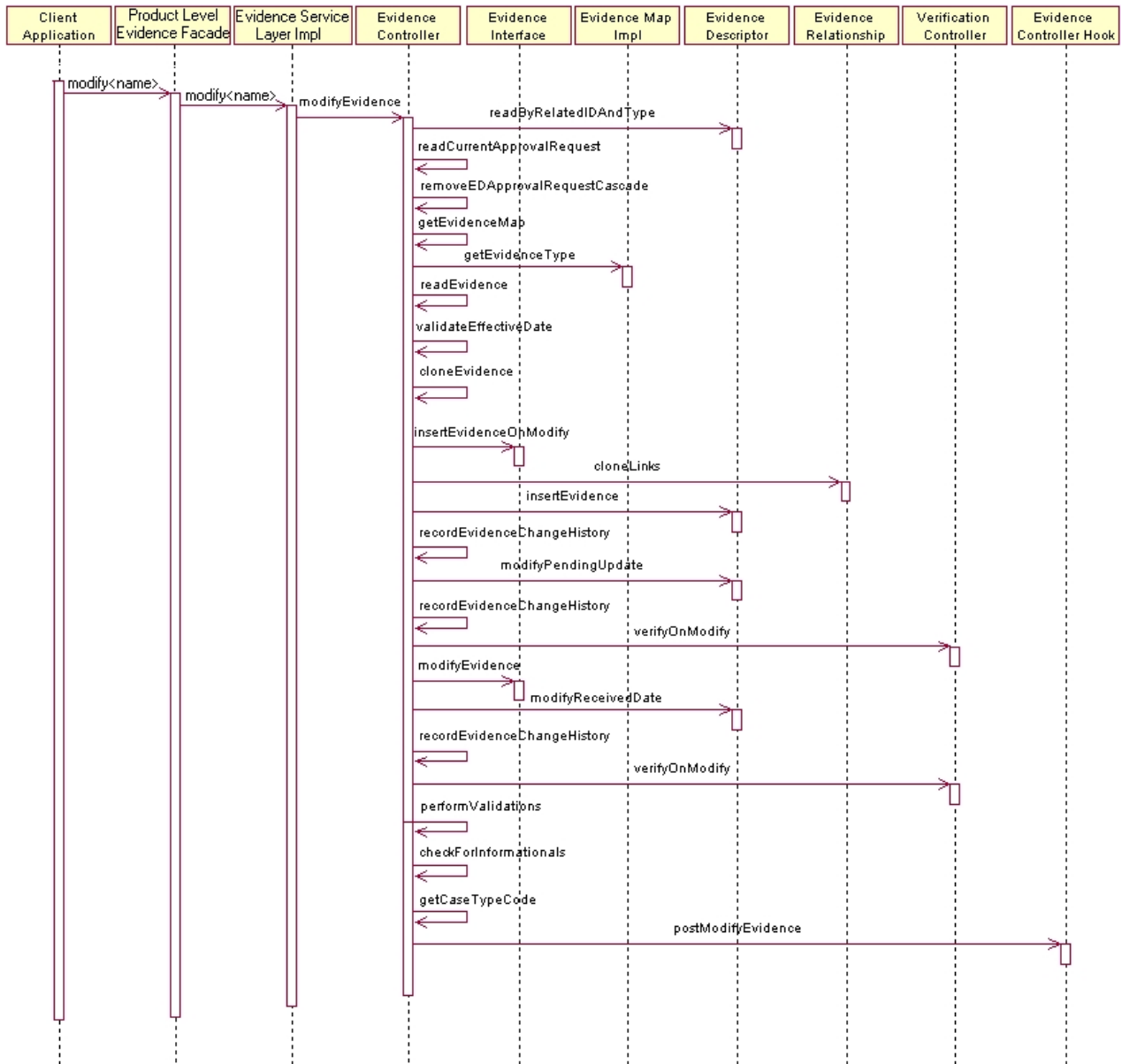
return createdEvidence;
}

```

Modify evidence

A modify evidence operation involves client and server development.

Sequence diagram for modifying evidence



Client screen to develop

The client page to be developed must include the evidence infrastructure page `Evidence_modifyHeader1.vim`. The **included.vim** page facilitates the viewing or modification or both of some infrastructure attributes. For example, the received date can be viewed or modified by using `this.vim`. Also, the change reason and the effective date of change can be set on the edited record. If, in the future, more attributes that need to be managed through the modify function are added to the Evidence Descriptor entity, then these attributes might be mapped through this infrastructure page. So, the operation requires just a once-off infrastructure change rather than many changes to custom artifacts.

The inclusion of `Evidence_modifyHeader1.vim` facilitates the following three types of evidence modification.

- Editing evidence in place

Editing evidence in place refers to the modification of incorrect data on a piece of evidence that is not yet activated. In this scenario, if the effective date is modified an error is thrown that informs the user that the date can be modified only when the user is updating an active record.

- Evidence correction

An evidence correction occurs when a piece of data on an active evidence record is modified that results in superseding the current active record. In this scenario, the effective date field must not be modified because it results in a creating new record in the succession.

- Evidence succession

If the user modifies the effective date when the user is updating a piece of active evidence, the user is specifying a new record in the succession set, that is, the new record has the same successionID as the active record. So, the active record is copied and made effective from the effective date that is specified by the user and the update is applied to this record.

Note: The activation of newly created records in a succession causes the reattribution of records in that succession set.

Server methods to implement

The `SEGEvidenceMaintenance.modifyAssetEvidence` facade operation calls the evidence service layer implementation.

```
// -----  
/**  
 * Modifies an Asset evidence record.  
 *  
 * @param details The modified evidence details.  
 *  
 * @return The details of the modified evidence record.  
 */  
public ReturnEvidenceDetails modifyAssetEvidence(  
    AssetEvidenceDetails dtls)  
    throws AppException, InformationalException {  
  
    // set the informational manager for the transaction  
    TransactionInfo.setInformationalManager();  
  
    // Asset evidence manipulation object  
    Asset evidenceObj = AssetFactory.newInstance();  
  
    // return object  
    ReturnEvidenceDetails modifiedEvidenceDetails =  
        new ReturnEvidenceDetails();  
  
    // modify the Asset record and populate the return details  
    modifiedEvidenceDetails =  
        evidenceObj.modifyAssetEvidence(dtls);  
  
    modifiedEvidenceDetails.warnings =  
        EvidenceControllerFactory.newInstance().getWarnings();  
  
    return modifiedEvidenceDetails;  
}
```

The `Asset.modifyAssetEvidence` service layer operation calls the Evidence Controller infrastructure function for modifying evidence.

```
// -----  
/**  
 * Modifies an Asset record.  
 *  
 * @param dtls Contains Asset evidence record modification  
 *             details.  
 *  
 * @return The modified evidence ID and warnings.  
 */  
public ReturnEvidenceDetails modifyAssetEvidence(  
    AssetEvidenceDetails details)  
    throws AppException, InformationalException {  
  
    // validate the mandatory fields
```

```

validateMandatoryDetails(details);

// EvidenceController business object
EvidenceControllerInterface evidenceControllerObj =
    (EvidenceControllerInterface)
        EvidenceControllerFactory.newInstance();

EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();

//
// Call the EvidenceController to modify the evidence
//

eiEvidenceKey.evidenceID = details.dtls.evidenceID;
eiEvidenceKey.evidenceType = CASEEVIDENCE.ASSET;

// Create the evidence interface modification struct and assign
// the details
EIEvidenceModifyDtls eiEvidenceModifyDtls =
    new EIEvidenceModifyDtls();
eiEvidenceModifyDtls.descriptor.receivedDate =
    details.descriptor.receivedDate;
eiEvidenceModifyDtls.descriptor.versionNo =
    details.descriptor.versionNo;
eiEvidenceModifyDtls.descriptor.effectiveFrom =
    details.descriptor.effectiveFrom;
eiEvidenceModifyDtls.descriptor.changeReceivedDate =
    details.descriptor.changeReceivedDate;
eiEvidenceModifyDtls.descriptor.changeReason =
    details.descriptor.changeReason;
eiEvidenceModifyDtls.evidenceObject = details.dtls;

evidenceControllerObj.modifyEvidence(
    eiEvidenceKey, eiEvidenceModifyDtls);

//
// Return details from the modify operation
//

ReturnEvidenceDetails returnEvidenceDetails =
    new ReturnEvidenceDetails();
returnEvidenceDetails.evidenceKey.evidenceID =
    eiEvidenceKey.evidenceID;
returnEvidenceDetails.evidenceKey.evType =
    eiEvidenceKey.evidenceType;
returnEvidenceDetails.warnings =
    evidenceControllerObj.getWarnings();

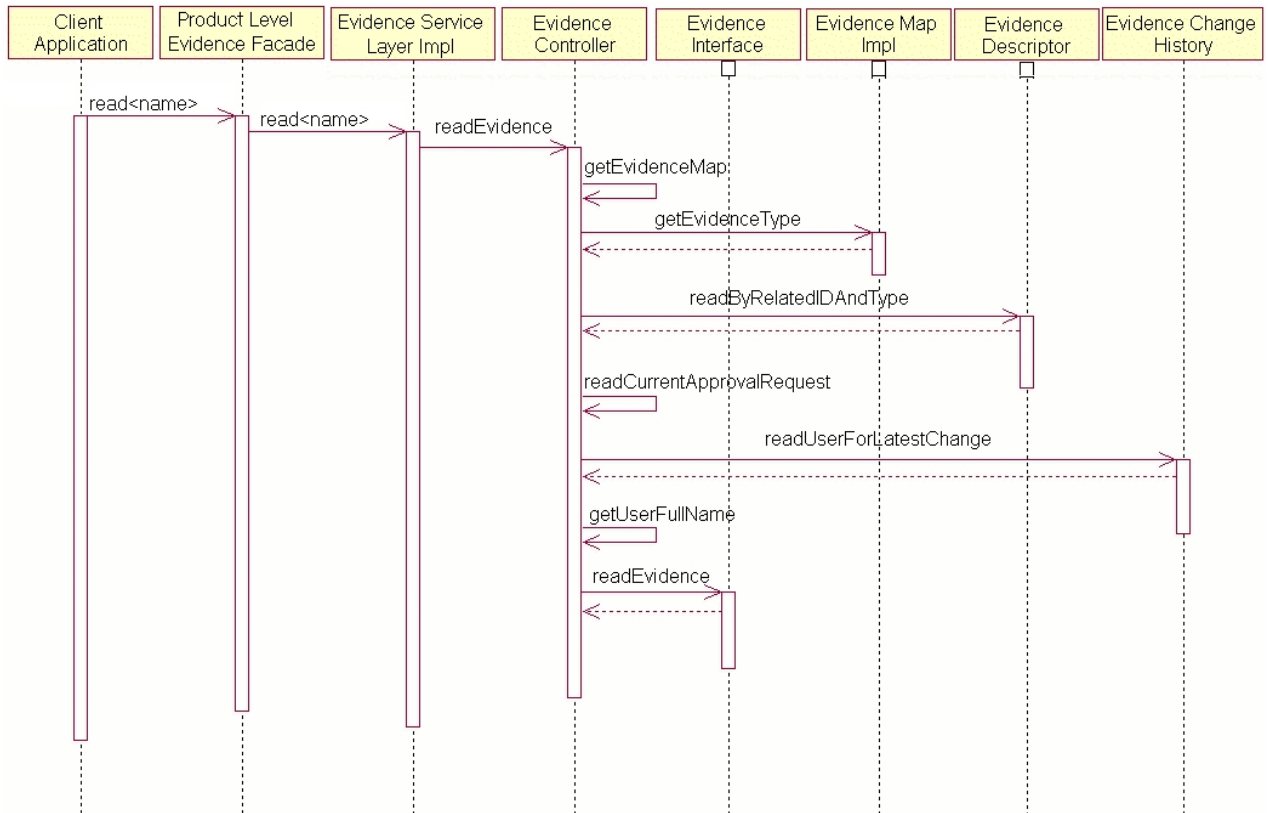
return returnEvidenceDetails;
}

```

Read evidence

A read evidence operation involves client and server development.

Sequence diagram for viewing evidence



Client screen to develop

The client page includes the evidence infrastructure page `Evidence_viewHeaderForModal.vim`. The included `.vim` facilitates the viewing of some infrastructure attributes.

Server methods to implement

The `SEGEvidenceMaintenance.readAssetEvidence` façade operation calls the evidence service layer implementation.

```

// -----
/**
 * Reads an Asset evidence record.
 *
 * @param key Identifies the evidence record to read.
 *
 * @return The details of the evidence record.
 */
public ReadAssetEvidenceDetails readAssetEvidence(
    EvidenceCaseKey key)
    throws AppException, InformationalException {

    // Asset evidence manipulation object
    Asset evidenceObj = AssetFactory.newInstance();

    // return object
    ReadAssetEvidenceDetails readEvidenceDetails =
        new ReadAssetEvidenceDetails();

    // read the Asset record and populate the return details
    readEvidenceDetails = evidenceObj.readAssetEvidence(key);

    return readEvidenceDetails;
}
  
```

```
}
```

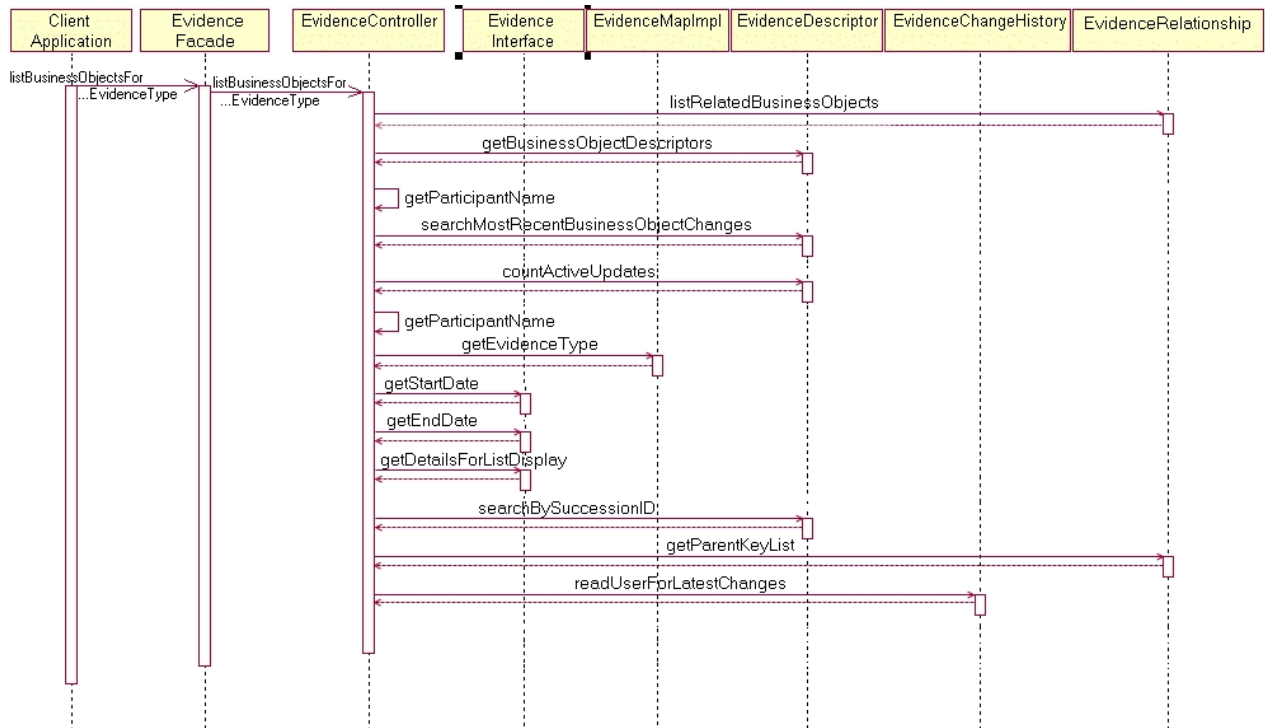
This service layer operation calls the Evidence Controller infrastructure function for reading evidence.

```
// -----  
/**  
 * Reads an Asset record.  
 *  
 * @param key contains ID of record to read.  
 *  
 * @return Asset evidence details read.  
 */  
public ReadAssetEvidenceDetails readAssetEvidence(  
    EvidenceCaseKey key)  
    throws AppException, InformationalException {  
  
    // EvidenceController business object  
    EvidenceControllerInterface evidenceControllerObj =  
        (EvidenceControllerInterface)  
            EvidenceControllerFactory.newInstance();  
  
    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();  
    eiEvidenceKey.evidenceID = key.evidenceKey.evidenceID;  
    eiEvidenceKey.evidenceType = CASEEVIDENCE.ASSET;  
  
    // Retrieve the evidence details  
    EIEvidenceReadDtls eiEvidenceReadDtls =  
        evidenceControllerObj.readEvidence(eiEvidenceKey);  
  
    // Retrieve the evidence descriptor details  
    EvidenceDescriptor evidenceDescriptorObj =  
        EvidenceDescriptorFactory.newInstance();  
  
    EvidenceDescriptorKey evidenceDescriptorKey =  
        new EvidenceDescriptorKey();  
    evidenceDescriptorKey.evidenceDescriptorID =  
        eiEvidenceReadDtls.descriptor.evidenceDescriptorID;  
  
    EvidenceDescriptorDtls evidenceDescriptorDtls =  
        evidenceDescriptorObj.read(evidenceDescriptorKey);  
  
    //  
    // Return the evidence  
    //  
  
    ReadAssetEvidenceDetails readEvidenceDetails =  
        new ReadAssetEvidenceDetails();  
    readEvidenceDetails.descriptor  
        .assign(evidenceDescriptorDtls);  
  
    readEvidenceDetails.descriptor.approvalRequestStatus =  
        eiEvidenceReadDtls.descriptor.approvalRequestStatus;  
    readEvidenceDetails.descriptor.updatedBy =  
        eiEvidenceReadDtls.descriptor.updatedBy;  
    readEvidenceDetails.descriptor.updatedDateTime =  
        eiEvidenceReadDtls.descriptor.updatedDateTime;  
  
    // assign the evidence to the return object  
    readEvidenceDetails.dtls.assign(  
        (AssetDtls)(eiEvidenceReadDtls.evidenceObject));  
  
    return readEvidenceDetails;  
}
```

List evidence

A list evidence operation involves client and server development. The list operation is used to populate an evidence workspace page.

Sequence diagram for listing evidence



Server methods to develop

Much of the data that is displayed on the workspace page is retrieved by the Evidence Descriptor entity. The description and period are retrieved by the Evidence Interface methods that must be implemented for each evidence type.

• *Asset.getDetailsForListDisplay* entity operation

The description, or summary details, is retrieved by the `getDetailsForListDisplay` Evidence Interface method that is implemented by the evidence entities. The proceeding illustrates the implementation of the `getDetailsForListDisplay` method for the Asset. This interface function is also used to retrieve summary data when the user is applying, approving, rejecting evidence and in evidence sharing, verifications, and, issues screens.

```

// -----
/**
 * Gets evidence details for the list display
 *
 * @param key Evidence key containing the evidenceID and
 * evidenceType
 *
 * @return Evidence details to be displayed on the list page
 */
public EIFieldsForListDisplayDtls getDetailsForListDisplay(
    EIEvidenceKey key)
    throws ApplicationException, 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.getOneItem(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 eiFieldsForListDisplayDtls;
}

```

(deprecated) Evidence Dashboard and EvidenceFlow

The **Evidence Dashboard** and **EvidenceFlow** are user interface constructs introduced to assist user navigation to all evidence on a case. No custom code is required in order to configure these for a custom case as these are infrastructural.

From these pages, a user can select a particular evidence type which should open the respective evidence workspace for that type of evidence. In the case of the Dashboard, this will open in a new tab, whereas the EvidenceFlow will redirect the bottom portion of the page.

The existence of 'In Edit' evidence records, outstanding verifications and outstanding issues are all highlighted graphically.

The list of evidence types on the case may be split into categories on these pages, by defining the category on the AdminICEvidenceLink table for Integrated Cases, or on the ProductEvidenceLink table for Product Deliveries.

Validations

The infrastructure facilitates the validation of work-in-progress changes. The validate page can be used either at a case level or on an individual evidence type.

The purpose of the case level validate page is to provide a means to test validations in advance of applying the changes. For some products, the full evidence set can be sizeable and results in the apply changes listing containing a considerable number of evidence changes of varying evidence types.

In that scenario, the individual evidence type validate page can make it easier to associate a validation message with the correct evidence record. The validate page allows a user to pre-test the evidence changes. The user can see the validations that fail and can fix the fails before the user applies the changes.

Two of the Evidence Interface functions that form part of the infrastructure support for evidence validation are `selectForValidations` and `validate`.

selectForValidations

The `selectForValidations` function is typically used to select all evidences that are related to or depend on the piece of evidence that is being validated. For example, modifying an amount on a parent evidence record. As part of the validation of the parent evidence, a check might need to be performed to ensure that the sum of the child evidence records does not exceed the modified parent amount.

When a user applies changes to evidence records, the Evidence Controller calls out to the `selectForValidations` interface function on the entities for each evidence record. The logic within this method retrieves all related Active and In Edit evidences within the hierarchy for validation. For instance, where the system is validating a child evidence record within a parent-child-grandchild relationship structure, both parent evidence and grandchild evidence are retrieved for the validation processing.

When processing returns to the Evidence Controller, a filter is applied to the list of evidence. The filter determines the input list and leaves only Active records, or In Edit records, depending on whether the function must validate against work-in-progress or active only evidence. The filtered list is then passed to the validate function where custom validation is applied.

The proceeding program listing displays a selectForValidations implementation that is used in the Asset demonstration.

```
// -----
/**
 * Selects all the records for validations
 *
 * @param evKey Contains an evidenceID / evidenceType pairing
 *
 * @return List of evidenceID / evidenceType pairings
 */
public EIEvidenceKeyList selectForValidation(
    EIEvidenceKey evKey)
    throws AppException, InformationalException {

    // Return object
    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();

    // Casting to impl due to calling non-modeled interface
    curam.seg.evidence.entity.intf.AssetOwnership
        assetOwnershipObj =
            (curam.seg.evidence.entity.impl.AssetOwnership)
                AssetOwnershipFactory.newInstance();

    eiEvidenceKey.evidenceID = evKey.evidenceID;
    eiEvidenceKey.evidenceType =
        CASEEVIDENCE.ASSET;

    EIEvidenceKeyList eiEvidenceKeyList =
        assetOwnershipObj.readAllByParentID(eiEvidenceKey);

    eiEvidenceKeyList.dtls.add(0, evKey);

    return eiEvidenceKeyList;
}
```

The code here, on the Asset parent entity, makes a call to the readAllByParentID interface method implementation on the child entity, Asset Ownership. The proceeding program listing displays the implementation of the readAllByParentID function on the Asset Ownership.

```
// -----
/**
 * Read all Asset Ownership records
 *
 * @param key Contains the evidenceID and evidenceType
 *
 * @return A list of evidenceID and evidenceType pairs
 */
public EIEvidenceKeyList readAllByParentID(EIEvidenceKey key)
    throws AppException, InformationalException {

    // Return object
    EIEvidenceKeyList eiEvidenceKeyList = new EIEvidenceKeyList();

    // Create the link entity object
    EvidenceRelationship evidenceRelationshipObj =
        EvidenceRelationshipFactory.newInstance();

    // parent entity key
    ParentKey parentKey = new ParentKey();
    parentKey.parentID = key.evidenceID;
    parentKey.parentType = key.evidenceType;

    // Reads all relationship details for the specified parent
    ChildKeyList childKeyList =
        evidenceRelationshipObj.searchByParent(parentKey);

    // Iterate through the link details list
    for (int i = 0; i < childKeyList.dtls.size(); i++) {

        if (childKeyList.dtls.item(i).childType.equals(
            CASEEVIDENCE.ASSETOWNERSHIP)) {

            EIEvidenceKey listEvidenceKey = new EIEvidenceKey();

```

```

        listEvidenceKey.evidenceID =
            childKeyList.dtls.item(i).childID;
        listEvidenceKey.evidenceType =
            childKeyList.dtls.item(i).childType;

        eiEvidenceKeyList.dtls.addRef(listEvidenceKey);
    }
}

return eiEvidenceKeyList;
}

```

The preceding function retrieves all child evidence keys for the specified parent. The `childID` and `childType` pairings are returned to the calling mechanism.

Evidence attribution and reattribution

Evidence attribution refers to the assignment of a time period to a piece of evidence during which that piece of evidence is used for entitlement calculations.

Attribution

The attribution period can range from a basic one-to-one mapping from the business start and end dates through to a more sophisticated algorithm that considers various factors. This custom logic calculates the attribution period. The evidence controller manages the synchronizing of the attribution period with the specified effective dates.

Note: The attribution from and to dates can be null in which case the piece of evidence is assumed effective from the case start date to the expected end date.

Example

One of the Evidence Interface functions is `calcAttributionDatesForCase` and the implementation of this function on an entity class is where the attribution **From** and **To** dates are determined for evidence on that entity.

Re-attribution

When evidence is modified as part of a succession and later activated, reattribution of the evidence records in the succession set occurs.

Example

Business Start Date: 3 May 2006 (=attribution from date)

Business End Date: 30 July 2006 (=attribution to date)

A succession record is created effective from 5 June 2006. On activation of this record, the evidence is reattributed and the proceeding attribution records are created.

- 3 May 2006 to 4 June 2006
- 5 June 2006 to 30 July 2006

Reattribution also occurs where evidence in a succession set is removed. In the proceeding example, three attribution records exist for records in the same succession set.

- 3 May 2006 to 4 June 2006
- 5 June 2006 to 30 July 2006
- 31 July 2006 to 29 Sept 2006

The evidence record associated with the second entry, that is, 5 June 2006 to 30 July 2006, is removed. So, by applying changes the proceeding reattribution is caused.

- 3 May 2006 to 4 June 2006

- 31 July 2006 to 29 Sept 2006

The attribution record 5 June 2006 to 30 July 2006 remains on the database, but is not selected by eligibility processing as the associated evidence is removed, that is, the associated evidence has a status of Canceled.

EvidenceRelationship link entity

By default, the Evidence infrastructure facilitates the linking of parent-child evidence by using the EvidenceRelationship link entity.

The proceeding table lists the structure of the EvidenceRelationship link entity.

Evidence relationship
evidenceRelationshipID
parentID
parentType
childID
childType

The EvidenceRelationship supports the relationship between any parent-child evidence and eliminates the necessity for customers to model their own link entities for managing such relationships. When evidence is being inserted, the generic EvidenceController.insertEvidence function calls to the business process EvidenceRelationship.createLink.

Where a parent type is specified, that is, passed in from the client as part of the insert, then a record is written to the EvidenceRelationship entity that links the child evidence to its parent. Also, the system calls to the business process EvidenceRelationship.cloneLinks directly after the call to the interface operation insertEvidenceOnModify. From cloneLinks, two further calls are made to cloneLinksForParent and cloneLinksForChild.

Where customers are using their own link entities to manage relationships, customers must override the Evidence Relationship business processes for creating and cloning links. The evidence type is available in the input keys of both these functions. So, responsibility can be delegated to the appropriate custom relationship processing based on the evidence type in the key.

Registering evidence implementations

The evidence maintenance pattern requires the set of evidence entities to be registered before they can be used so that the controller can access the evidence entities at runtime.

The Core Cúram Framework cannot anticipate the evidence entities to use for the evidence maintenance facility associated with a particular product implementation. So, the evidence types and their implementation must be paired at run time.

Evidence registrar module

Use Google Guice dependency injection to register the different evidence types and their implementations. To register the different evidence types and their implementations, write a new module class or add the evidence type and their implementations to an existing evidence module class. When the module class is added to the ModuleClassName table, then at runtime it is loaded and the evidence types registered.

The proceeding is an example of a Google Guice dependency injection.

```
/*
 * Copyright 2011 Cúram Software Ltd.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Cúram 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 Cúram Software.
*/

package curam.seg.evidence.service.impl;

import curam.codetable.CASEEVIDENCE;
import com.google.inject.AbstractModule;
import curam.core.impl.FactoryMethodHelper;
import java.lang.reflect.Method;
import com.google.inject.multibindings.MapBinder;
import curam.core.impl.RegistrarImpl;
import curam.core.impl.Registrar.RegistrarType;

/**
 * A module class which provides registration for all of the
 * evidence hook implementations.
 */
public class SEGRegistrarModule extends AbstractModule {

    @Override
    public void configure() {

        // Register all hook implementations which implement the
        // interface EvidenceInterface.
        MapBinder<String, Method> evidenceInterfaceMapBinder =
            MapBinder.newMapBinder(binder(), String.class,
                Method.class, new RegistrarImpl(RegistrarType.EVIDENCE));

        evidenceInterfaceMapBinder
            .addBinding(CASEEVIDENCE.ASSET)
            .toInstance(FactoryMethodHelper.getNewInstanceMethod(
                curam.seg.evidence.entity.fact.AssetFactory.class));
    }
}

```

Legacy evidence registrar

The legacy mechanism for registration of evidence entities is still supported, that is, by using the Application Properties to specify the factories to populate a hash map of the hook classes. The factory code does not change to maintain compatibility with an earlier version. However, all default legacy implementations are deprecated.

Customizing evidence maintenance

As the Evidence Controller functionality is generic to all evidence solutions, the only way to facilitate an organization's unique requirements is to provide hooks where custom logic can be located to extend the core solution. Callouts to these hooks, or extension points, are made within the Evidence Controller maintenance functions.

The Cúram infrastructure handles the maintenance of evidence, such as adding, modifying, removing, and applying changes. The infrastructure is independent of the evidence type, that is, by default all evidence types are treated the same.

Customers might need to customize the processing available for immediate use to meet project-specific needs. To facilitate the customize the processing, the `EvidenceControllerHook` interface provides a set of extension points that allows custom code to be run at points in the evidence maintenance process.

As well as adding custom code to the extension points, customers can specify 'case type' specific logic. Customers can use 'case type' specific logic to allow multiple implementations of the `EvidenceControllerHook` to be provided. Each implementation can be mapped to a 'case type' to give case type-specific customization. For example, the `postRemoveEvidence` for evidence on a Product Delivery case might be different than the `postRemoveEvidence` that is run on an Integrated Case.

Evidence Controller Hook

The Evidence Controller Hook is the evidence infrastructure class that contains the extension points for the evidence maintenance pattern.

Example

An example of a hook in the evidence infrastructure class is `postRemoveEvidence`. A call is made to this function inside the Evidence Controller `removeEvidence` operation. Where customers want to perform post-remove evidence processing, customers must override the hook with their custom version.

Providing a custom implementation of the EvidenceControllerHook

To inject a custom implementation at the provided extension points, the abstract base class `curam.core.sl.infrastructure.impl.EvidenceControllerHook` can be extended and the wanted methods can be overridden.

For most methods of the base abstract class, the implementation does nothing, but some default implementations are provided, such as for the `PreRemoveEvidence` method. The Java docs of the class can be referenced to recognize the default implementation. If required, the `super().methodName()` notation can be used to start the default implementation from an overridden method to retain the base functions.

To create a new custom `EvidenceController` hook, use the proceeding steps.

- A new process class is modeled in, for example, `CustomHook`. This process must have a 'Generalization' relationship with `EvidenceControllerHook` class (extends `EvidenceControllerHook`).
- An implementation of the newly created process is created, in which any wanted methods are overridden:

```
public class CustomHook extends curam.sample.sl.base.CustomHook {  
  
    @Override  
    public void postInsertEvidence(CaseKey caseKey,  
        EIEvidence eiEvidenceKey){  
  
    }  
}
```

- A new Module class is created, where the wanted product type is bound to the custom hook implementation. This class must extend `AbstractModule` and a configuration for this module class must be added to `MODULECLASSNAME.dmx`:

```
public class TestRegistrarModule extends AbstractModule{  
  
    @Override  
    protected void configure(){  
        MapBinder<String, Method> evidenceControllerMapBinder =  
            MapBinder.newMapBinder(binder(), String.class, Method.class,  
                new RegistrarImpl(RegistrarType.EVIDENCE_CONTROLLER_HOOK));  
  
        evidenceControllerMapBinder  
            .addBinding(PRODUCTTYPE.CUSTOMPRODUCTTYPE)  
            .toInstance(FactoryMethodHelper.getNewInstanceMethod(  
                CustomHookFactory.class));  
    }  
}
```

The preceding adds a binding of `CustomHook` implementation to `PRODUCTTYPE.CUSTOMPRODUCTTYPE` product type string. Product type is used as a key during the `EvidenceControllerHook` implementation look-up. The infrastructure compares this key to the value returned by the implementation of `CaseTypeEvidence.getCaseTypeCode()` that is specific to the evidence type that is being processed. `CaseTypeEvidence` has many implementations, and the implementation return different case type codes. Refer to the Javadoc to determine the run type of any particular implementation. The key that is used in the binding Module must match the value that is returned by `getCaseTypeCode()`, otherwise the custom hook is not picked up. For example, evidence on a Product

Delivery case uses a "productType" code that is defined in the PRODUCTDELIVERY database table. Commonly used case type codes are listed in the proceeding table.

Case name	Case type code database location
Default	CASHEADER.caseTypeCode
Integrated Case	CASHEADER.integratedCaseType
Product Delivery	PRODUCTDELIVERY.productType
Screening Case	SCREENINGCONFIGURATION.name
Assessment Delivery	ASSESSMENTCONFIGURATION.assessmentType
Investigation Delivery	INVESTIGATIONDELIVERY.investigationType

The Evidence Controller Hook Manager class manages the static initialization of the Evidence Controller Hook mapping and the retrieval of the subclass of the Evidence Controller Hook. If no subclass is found, the version of the Evidence Controller Hook class that is available for immediate use is returned.

Evidence Controller Hook Registrar and Manager

The registration of the Evidence Controller Hook class uses a similar pattern to the Evidence Registrar and the underlying Dependency Injection pattern. An Evidence Controller Hook Registrar interface is shipped as part of the evidence infrastructure.

As before, at run time, the Evidence Controller starts the Registrar's register method that performs the dependency injection of the associated custom Evidence Controller Hook. This is the class that extended the default Evidence Controller Hook and overridden the methods that are being customized. This "injector" class is located through runtime configuration where the injector class itself is referred to as the "Evidence Controller Hook Registrar".

Dependency injection

The dependency injection involves two steps. First, a custom Evidence Controller Hook Registrar, which implements the Evidence Controller Hook Registrar interface, must be located and the Registrar then started to register the customized hook class. For example, the product type and custom Evidence Controller Hook class pairing is entered into a hash map and then the class looked up by the product type when it is required. To locate the Evidence Controller Hook Registrar, its class name must be configured that uses the environment variable `curam.case.evidencecontrollerhook.registrars`.

Note: More entries need to be added to the environment variable in a comma-delimited format.

The implementation of the Registrar's register method must reference the customized Evidence Controller Hook class. By using code, rather than as configuration, provides a compile-time check that the referenced class exists. The existence of the Registrar, though, is only ascertained from the provided configuration, and can result in a runtime failure if the application is not correctly configured.

The Evidence Controller Hook Manager class manages the static initialization of the Evidence Controller Hook mapping as well as the retrieval of the subclass of the Evidence Controller Hook. If no subclass is found, the default version of the Evidence Controller Hook class is returned.

Customizing multiple participant evidence

Use the multiple participant evidence to insert multiple records, modify multiple records, or discard multiple records in a single action.

Multiple participant evidence can save time and effort when caseworkers are managing multiple clients on a case, such as adding the same address for all family members in a single operation.

Multiple participant evidence extension points

You can use six extension points for customization.

The proceeding six hook points are provided.

- Pre-create multiple participant evidence.
- Post-create multiple participant evidence.
- Pre-modify multiple participant evidence.
- Post-modify multiple participant evidence.
- Pre-discard multiple participant evidence.
- Post-discard multiple participant evidence

Implementation example

Perform the proceeding two steps to enact custom functionality.

1. Create a new class in your custom package that implements the `curam.core.sl.infrastructure.impl.MultiEvidenceHook`
2. Implement each method of the interface.

Note: The arguments supplied to the customization hook points are clones of the original. Modifications of the values are not reflected in the default flow.

```
class CustomMultiEvidenceHookImpl implements curam.core.sl.infrastructure.impl.MultiEvidenceHook
{
    /**
     * Include your custom processing in this function
     * and it will
     * be invoked before the multiple create operation.
     */
    public void preCreateMultiEvidence( final List<CaseParticipantRoleKey>
participantList)throws AppException, InformationalException
    {
        for (final CaseParticipantRoleKey item : participantList) {
            // Custom participant processing for pre create
            // multiple participant evidence
            ...
        }
    }
    // Implement all other interface methods, even if they do nothing.
    ...
}
```

Configuration example

When you create a `MultiEvidenceHook` implementation, perform the proceeding two steps to configure the implementation for use.

1. In your custom package,, create a new class that extends `com.google.guice.AbstractModule`.
2. Bind the custom implementation to interface that uses Guice binding.

```
public class HookModule extends AbstractModule {
    @Override public void configure()
    {
        // Bind custom multi evidence hook
        bind(MultiEvidenceHook.class).to(CustomMultiEvidenceHookImpl .class);
    }
}
```

Configuring custom filters for multiple participant evidence

You can customize multiple participant evidence to configure custom filters.

Use the multiple participant evidence maintenance filter to control the list of options that are presented to the user during multiple participant operations, specifically in the proceeding three scenarios.

1. The list of participants that are presented to the user during create operations.
2. The list of evidence that is presented to the user during modify operations.
3. The list of evidence that is presented to the user during a discard operations.

Filter types

You can use two types of filters: global filters and evidence type filters.

1. Use global filters as general filters to apply to all evidence, removing the need to apply for every evidence type. Also, global filters ensure that the filter is applied to newly created evidence types that support multiple participant evidence.
2. Use evidence type filters as specific filters to apply at the evidence type level. Evidence type filters permit a more fine grained control over how filters are applied.

Configuring global filters

Global filters are applied to all evidence types. Global filters can be used to provide general rules that are applied across all evidence types.

Using global filters removes the need to replicate filtering rules across multiple types and removes the need to create new filters for each newly created evidence type.

Default global filters

When the system displays a multiple participant create, update or discard page, the list of items that is presented to the user is constructed from the case participants or case evidence. For more information about how these lists are constructed, see the Javadoc information of the `curam.evidence.impl.DynamicEvidenceMultiEvidenceOperations` class.

After the unfiltered list is constructed, a global filter is applied for each operation type. For more information about how each default global filter works, see the Javadoc information of the `curam.core.sl.infrastructure.impl.MultiEvidenceFiltersImpl`.

Replacing global filters

If the default global filter is not suitable for your business scenario, the default global filter can be replaced with a custom version by configuring a new global filter.

You can implement a global filter for a multiple create scenario, multiple modify scenario, and multiple discard scenario.

Global filter for multiple create

The proceeding example shows how a global create filter can be applied to all evidence types that use multiple participant evidence maintenance. The class must extend the `AbstractMultiEvidenceFiltersImpl` and implement the `evaluateParticipantForMultiCreate` operation.

The filter in the proceeding example uses three criteria.

1. Participant exists on the case for the given received date.

2. Participant is of type PRIMARY or MEMBER.
3. Participant is active.

```

public class CustomMultiEvidenceFiltersImpl extends AbstractMultiEvidenceFiltersImpl
{
    /**
     * Removes the given participant from the list presented during multiple participant
create
     * operation.
     * The participant will be removed if they are not active, current and have a
participant
     * type of PRIMARY OR MEMBER.
     *
     * @param participant
     *      a case participant who is currently included in the multiple create list.
     * @return
     *      true if the participant should be excluded from the list.
     */
    protected boolean excludeParticipantFromMultiCreate(final MultiParticipantDtls
participant)
    {
        return participant.recordStatus.equals(RECORDSTATUS.NORMAL) &&
            (participant.typeCode.equals(CASEPARTICIPANTROLETYPE.PRIMARY) ||
             participant.typeCode.equals(CASEPARTICIPANTROLETYPE.MEMBER)) && new
participant.endDate().contains(getCurrentReceivedDate());
    }
}

```

Global filter for multiple modify

The proceeding example shows how a global modify filter can be applied to all evidence types that use multiple participant evidence maintenance. The class must extend the `AbstractMultiEvidenceFiltersImpl` and implement the `evaluateParticipantForMultiModify` operation.

The filter in the proceeding example uses two criteria.

1. For the participant whose evidence the modify operation was initiated from, filter out all other evidence records belonging to this participant.
2. Filter evidence that does not exist on the case for the given received date.

```

    /**
     * Custom class to redefine the global filter for the multiple participant
maintenance
     * evidence lists.
     */
    public class CustomMultiEvidenceFiltersImpl extends AbstractMultiEvidenceFiltersImpl
    {
        /**
         * Return true if you want to filter this item from the list of evidence that can be
         * modified.
         *
         * @param evidence
         *      an evidence record that is currently included in the multiple participant
update list.
         *
         * @return true if the evidence should be excluded from the multiple participant
update
         * list.
         */
        protected boolean excludeEvidenceFromMultiModify(final MultiEvidenceDtls evidence)
        {
            // Do not exclude by default
            boolean shouldExclude = false;
            try {
                shouldExclude = evidence.participantID !=
getCurrentEvidenceDescriptorDtls().participantID
                    && !new DateRange(evidence.startDate,
evidence.endDate).contains(

```

```

        getCurrentDynamicEvidenceObject().getReceivedDate());
    } catch (AppException e) {
        // Do not exclude
    } catch (InformationalException e){
        // Do not exclude
    }
    return shouldExclude;
}
}

```

Global filter for multiple discard

The proceeding example shows how a global discard filter can be applied to all evidence types that use multiple participant evidence maintenance. The class must extend the `AbstractMultiEvidenceFiltersImpl` and implement the `evaluateParticipantForMultiDiscard` operation.

The filter in the proceeding example uses one criteria.

1. For the participant whose evidence the discard operation was initiated from, filter out all other evidence records belonging to this participant.

```

/**
 * Custom class to redefine the global filter for the multiple participant
maintenance
 * evidence lists.
 */
public class CustomMultiEvidenceFiltersImpl extends AbstractMultiEvidenceFiltersImpl
implements MultiEvidenceFilters {

    /**
can be
 * Return true if you want to filter this given item from the list of evidence that
 * discarded.
 *
 * @param evidence
update list.
 * an evidence record that is currently included in the multiple participant
 *
 * @return true if the evidence should be excluded from the multiple participant
update
 * list.
 */
protected boolean excludeEvidenceFromMultiDiscard(final MultiEvidenceDtls evidence)
{
    boolean shouldFilter = false;
    try {
        shouldFilter = evidence.participantID !=
getCurrentEvidenceDescriptorDtls().participantID;
    } catch (AppException e) {
        // Do not filter
    } catch (InformationalException e){
        // Do not filter
    }
    return shouldFilter;
}
}

```

Global filters configuration

The proceeding example shows how to configure your custom filter for use. In the example, the `CustomMultiEvidenceFiltersImpl` class is bound to the default `MultiEvidenceFiltersImpl` class that results in the custom class that is overriding the default class.

1. In your custom package, create a new class that extends `com.google.guice.AbstractModule`.

2. Bind the custom implementation to interface using Guice binding.

```
/**
 * Configure Filters for Multiple Participant Evidence Maintenance.
 */
public class FilterModule extends AbstractModule {
    @Override
    public void configure() {
        // Bind custom evidence filter
        bind(MultiEvidenceFiltersImpl.class).to(CustomMultiEvidenceFiltersImpl.class);
    }
}
```

Configuring evidence type filters

Use evidence type filters to customize specific evidence types for multiple participant update operations. A custom filter can be configured to apply to one or more evidence types.

Note: Evidence type filters replace global filters. The type-specific filter receives the full set of records that can be legitimately displayed for the operation. For example, all case participants, including canceled ones, or all evidence of the same type, regardless of whether it is canceled or end dated. Evidence type filters provide you with full control over how records are filtered. However, it is likely that you must reapply some of the global rules.

Evidence type filters are configured by mapping the evidence type code of an evidence to a custom filter.

Implementing the multiple participant evidence filter

A multiple participant evidence filter can be implemented by extending the `curam.core.sl.infrastructure.impl.AbstractMultiEvidenceFiltersImpl` abstract class.

Perform the proceeding two steps to customize the filter.

1. Implement a custom filter by extending the `AbstractMultiEvidenceFiltersImpl`.
2. Add a binding for the custom filter implementation that uses Guice binder.

Implementing the new multiple participant evidence-specific filter

The proceeding example demonstrates how to create an evidence type-specific filter. The example excludes email addresses from the multiple update list of an email address modify or discard operation where email addresses are not of the same type as the email address record selected for update.

1. Create a custom class that extends `AbstractMultiEvidenceFiltersImpl` and implements the exclude methods for modify and discard operations.

```
public class MyCustomEmailAddressMultiEvidenceFiltersImpl extends
AbstractMultiEvidenceFiltersImpl {

    @Override protected boolean excludeEvidenceFromMultiModify(final MultiEvidenceDtls
evidence){
        return excludeEmailAddressEvidence(evidence);
    }

    @Override protected boolean excludeEvidenceFromMultiDiscard(final
MultiEvidenceDtls evidence){
        return excludeEmailAddressEvidence(evidence);
    }

    /**
     * Exclude evidence from email address multiple evidence update.
     */
    protected boolean excludeEmailAddressEvidence(final MultiEvidenceDtls evidence) {
        boolean shouldExclude = false;
    }
}
```

```

// Include by default.

final EvidenceDescriptorKey evidenceDescriptorKey = new EvidenceDescriptorKey();
evidenceDescriptorKey.evidenceDescriptorID = evidence.evidenceDescriptorID;
try {
    // Re-apply the global filter rules because we have disabled them by
    // adding this type specific filter.
    boolean evidenceShouldBeConsidered = evidence.participantID !=
getCurrentEvidenceDescriptorDtls().participantID
    && !new DateRange(evidence.startDate,
evidence.endDate).contains(getCurrentDynamicEvidenceObject().getReceivedDate());

    if (evidenceShouldBeConsidered) {
        boolean shouldExclude =

((String)readDynamicEvidenceObject(evidenceDescriptorKey).getAttributeValue( PDCEmailAddress.
emailAddressTypeAttr)).equals(
(String)

getCurrentDynamicEvidenceObject().getAttributeValue(PDCEmailAddress.emailAddressTypeAttr));
    }
    catch (AppException e) {
        // Default to include
    }catch (InformationalException e){
        // Default to include
    }
    return shouldExclude;
}
}

```

2. Create a Guice module to bind the implementation. For more information about using the Guice modules with Curam, see the *Creating a Guice module* related link. Use the evidence type code to bind the implementation that uses the standard Guice map binder strategy. In the example, the evidence type code that is needed for the binding is PDC0000260, which maps to the 'Email Addresses' evidence type. You can look up the evidence type code value on the EvidenceType code table.

```

public class EvidenceFilterModule extends AbstractModule {

    @Override public void configure() {
        final MapBinder<String, MultiEvidenceFilters>
        pdcMultiEvidenceFiltersMapBinder = MapBinder.newMapBinder(binder(),
String.class, MultiEvidenceFilters.class);
        pdcMultiEvidenceFiltersMapBinder.addBinding("PDC0000260").to(
MyCustomEmailAddressMultiEvidenceFiltersImpl.class);
    }
}

```

Related concepts

[Creating a Guice module](#)

Evidence end dating feature implementation

Caseworkers create an evidence record by recording the evidence in the first page of the evidence wizard. If an administrator enables the end dating feature for the evidence type and the end dating criteria are met, a second page is displayed in the evidence wizard. On the page, caseworkers can end date previous evidence records. Administrators need to be aware of some implementation details and behavior in relation to the end dating of evidence records through the evidence wizard.

The following information supplements the configuration information that is described in the *Enabling the End Dating of Previous Evidence When Creating Evidence* topic. Also, the following information supplements the procedural information that is described in the *Applying end dating in the creation of evidence records* topic. For more information, see the related links.

Navigating to the end dating evidence option in the evidence wizard

In the evidence wizard, to navigate from the first page where an evidence record is created to the second page where evidence records can be end dated, caseworkers must click **Save and Next**. Note the following points:

- If the create evidence transaction fails, the transaction is rolled back, no record is committed to the database, and the appropriate validation error is displayed to the caseworker on the same evidence record creation page. The caseworker is not redirected to the next wizard page.
- If the create evidence transaction is successful, the evidence is created and committed to the database, and the caseworker is directed to the second page of the wizard. Therefore, as the create and end date processes are separated as end-to-end transactions, the caseworker cannot navigate back to the previous evidence record creation page.

Completing the evidence wizard

In the evidence wizard, when the caseworker clicks **Finish** in the evidence end dating page, the end date evidence transaction is triggered. Note the following points:

1. If the end date evidence transaction fails, the transaction is rolled back, no record is committed to the database and the appropriate validation errors are displayed to the caseworker in the same evidence end dating page. For dynamic evidence records, the end dating validation errors are aggregated so that all validation errors are displayed to the caseworker. To enable aggregated validation error messages for non-dynamic evidence records, in the customized non-dynamic evidence validation classes, replace the `InformationalManager.failOperation()` method call with the `MultiFailOperation.failOperationWithMPO()` method call. If you do not replace the method, when the first validation error occurs, the application might display the validation error message in the user interface instead of in the aggregated validation error messages.

Note: The end date of all selected evidence records is aggregated in one single transaction. Therefore, if the end dating of one evidence record fails, the whole transaction is rolled back and no evidence records are end dated.

2. If the end date evidence transaction is successful, all selected evidence records are end dated and the data is committed to the database.

Customizing the default implementation

You can customize the default implementation in the `curam.core.sl.infrastructure.impl.ListAutoEndDateEvidenceImpl.listEvidenceForAutoEndDating()` method. The method lists the evidences to be end dated that are displayed in the evidence record end dating page of the evidence wizard. To customize the method, create a custom implementation class that extends the `curam.core.sl.infrastructure.impl.ListAutoEndDateEvidenceImpl` default implementation class.

The custom class must never directly implement the interface class because compilation exceptions might occur during an upgrade if you add new methods to the interface. To ensure that the application runs the new custom class rather than the default implementation, you must use the standard Guice dependency injection mechanism to implement a new module class that extends the `com.google.inject.AbstractModule` module. You must insert the fully qualified module class name into the `MODULECLASSNAME` database table.

Enabling hook points

You can enable the hook points through the standard Guice dependency injection mechanism. Hook points are provided to the evidence end dating feature through the following interface methods:

- The `curam.core.sl.infrastructure.impl.AutoEndDateEvidenceHook.preAutoEndDateEvidence(curam.core.facade.infrastructure.struct.AutoEndDateEvidenceDetails)` interface method is started before the end dating of evidence records.
- The `curam.core.sl.infrastructure.impl.AutoEndDateEvidenceHook.postAutoEndDateEvidence(curam.core.facade.infrastructure.struct.AutoEndDateEvidenceDetails)` interface method is started after the end dating of all evidence records.

The hook points are started only through the end dating process that is triggered when a caseworker clicks **Finish** in the evidence wizard evidence end dating page.

Related tasks

[Enabling the end dating of previous evidence when creating evidence](#)

[Applying end dating in the creation of evidence records](#)

Participant evidence integration

Evidence is the term that is used for data in the calculation of eligibility and entitlement. Participant data is also regarded as evidence, for example, a concern's date of birth.

Previously, participant data wasn't always treated as classic evidence. It is correct for a concern's date of birth to be maintained within the Participant Manager rather than being stored on a separate evidence entity, that is, one that is interfaced to the Evidence API. However, it must also be propagated across all cases that belong to the concern and any changes in such evidence must trigger reassessment:

- A modification applied to participant data automatically applies to all cases that use this data.
- Modifying such data trigger reassessment of all cases that use this data.

The following core participant entities are integrated with evidence:

- Address
- AlternateID
- AlternateName
- BankAccount
- Citizenship
- ConcernRole
- ConcernRoleRelationship
- Education
- Employer
- Employment
- EmploymentWorkHour
- Foreign Residency
- Person
- ProspectEmployer
- ProspectPerson

Integration of participant data as evidence

Participant evidence integration is available by default, but, like evidence, it requires configuration. If the configuration is not carried out, then all newly integrated participant evidence does not integrate with the Evidence API. However, it continues to function as previously. When configured, the participant evidence is linked to one or more cases by using an Evidence Descriptor. As with classic evidence, the Evidence Descriptor can be associated with either an Integrated Case or a Product Delivery.

The required configuration links the participant evidence types to the Integrated Case, or Cases, or Product, or Products, that use them. Such data is stored on the AdminICEvidenceLink and ProductEvidenceLink, respectively. Participant data that is stored at the Integrated Case level must be configured on the AdminICEvidenceLink entity. Participant evidence that is used by a Product must be configured on the ProductEvidenceLink entity.

Administration

Administration of participant evidence integration involves two aspects: AdminICEvidenceLink and ProductEvidenceLink.

AdminICEvidenceLink

Every integrated case type that wants to integrate the available 15 entities as evidence must insert an entry into the AdminICEvidenceLink table. This table must link evidenceMetadataID (from EvidenceMetadata table) and adminIntegratedCaseID (from AdminIntegratedCase table) for each participant entity that is required as evidence and for each integrated case type.

ProductEvidenceLink

Every product delivery case type that wants to integrate the available 15 entities as evidence must insert an entry into the ProductEvidenceLink table. This table must link evidenceMetadataID (from EvidenceMetadata table) and productID (from Product table) for each participant entity that is required as evidence and for each product type.

Integrating new participant entities as evidence

To integrating new, or existing, participant entities with evidence, metadata must be configured for Integrated Case types and Product types, while other infrastructural support must be implemented.

Implementing the ParticipantEvidenceInterface

A participant entity that is being integrated into the evidence solution must implement the ParticipantEvidenceInterface. So, the following entities must be implemented:

- calcAttributionDatesForCase
- getDetailsForListDisplay
- getEndDate
- getStartDate
- insertEvidence
- insertEvidenceOnModify
- modifyEvidence
- readAllByParentID
- readEvidence
- selectForValidation
- validate
- checkForReassessment
- createSnapshot
- getChangedAttributeList
- readAllByConcernRoleID
- removeEvidence

Register entity in a Registrar Module

Participant entities that are being integrated to evidence must be registered that uses a Registrar Module. For more information, see the *Evidence Registrar Module* related link. The default participant evidence types are configured in CoreRegistrarModule, which binds the evidence type to its entity. The map bindings are loaded at run time and are used by the Evidence Controller when the Evidence Controller looks up the appropriate evidence entity for a type, that is, the entity that implemented the ParticipantEvidenceInterface.

Applying participant evidence to all cases

A new hook class `ApplyChangesForEvidence` is added.

The new `ApplyChangesForEvidence` class represents a hook that can be overridden by custom code. The `ApplyChangesForEvidence.isApplyChangesAutomatedForEvidence` method is called from Evidence Controller to determine whether reassessment needs to be triggered when evidence is applied.

The default implementation defaults to false and, therefore, the user must manually apply the changes on the associated cases. If the solutions need to customize, the implementers must use `ProductHookRegistrar.registerApplyChangesHooks` method to add details of the hooks to use for applying changes. The static map attribute, `applyChangesHookMap` present in `ProductHookManager` class is used to store pairs of product type and the name of the class that implements the hook for that product type. The method `ProductHookManager.getApplyChangesHook` gets the implementation subclass of the `ApplyChangesForEvidence` class for the specified product type. The method `EvidenceController.applyParticipantEvidence` is updated to obtain product delivery and product details for the case and then call `ProductHookManager.getApplyChangesHook` to obtain correct instance of the `ApplyChangesForEvidence` class for the given product.

Modifications that are required to existing business processes

In all places where there are existing calls to insert, modify, and less frequently, remove methods, the code needs to be updated to start the `EvidenceController` and the insert, modify, and remove methods. The proceeding example demonstrates how an insert works with evidence.

Before

```
// insert new citizenship entry
citizenshipObj.insert(citizenshipDtls);
```

After

```
//
// Call the EvidenceController object and insert evidence
// Evidence descriptor details
EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls =
    new EvidenceDescriptorInsertDtls();
evidenceDescriptorInsertDtls.participantID =
    details.concernRoleID;
evidenceDescriptorInsertDtls.evidenceType =
    CASEEVIDENCE.CITIZENSHIP;
evidenceDescriptorInsertDtls.receivedDate =
    Date.getCurrentDate();

// Evidence Interface details
EIEvidenceInsertDtls eiEvidenceInsertDtls =
    new EIEvidenceInsertDtls();
eiEvidenceInsertDtls.descriptor.assign(
    evidenceDescriptorInsertDtls);
eiEvidenceInsertDtls.descriptor.participantID =
    citizenshipDtls.concernRoleID;
eiEvidenceInsertDtls.evidenceObject =
    citizenshipDtls;

// EvidenceController business object
curam.core.sl.infrastructure.impl.EvidenceControllerInterface
evidenceControllerObj =
    (curam.core.sl.infrastructure.impl.EvidenceControllerInterface)
    curam.core.sl.infrastructure.fact.EvidenceControllerFactory
    .newInstance();

// Insert the evidence
EIEvidenceKey eiEvidenceKey =
    evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);
```

Sequence diagrams for participant evidence

Diagrams illustrate the development, both client and server, of creating and modifying evidence operations.

Create participant evidence sequence diagram

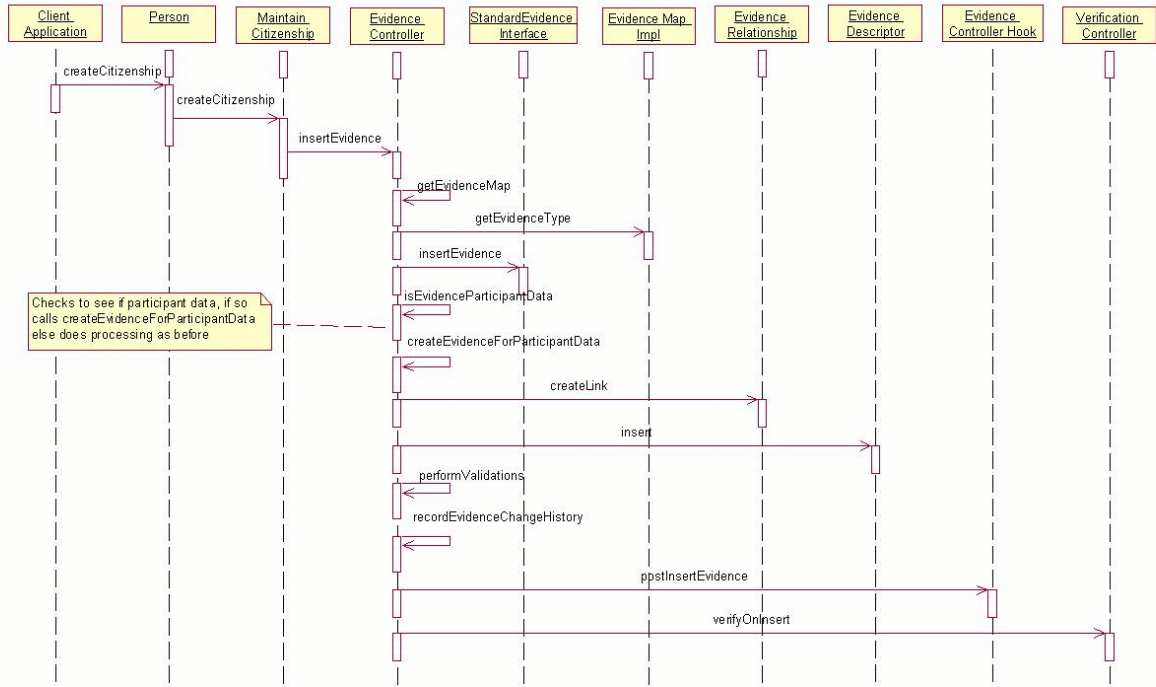


Figure 3: Participant evidence sequence

Specific processing for participant data when you are creating evidence

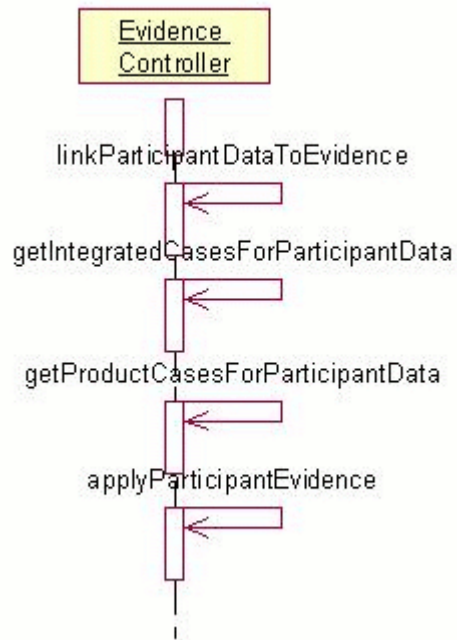


Figure 4: Evidence sequence diagram

Modify participant evidence sequence diagram

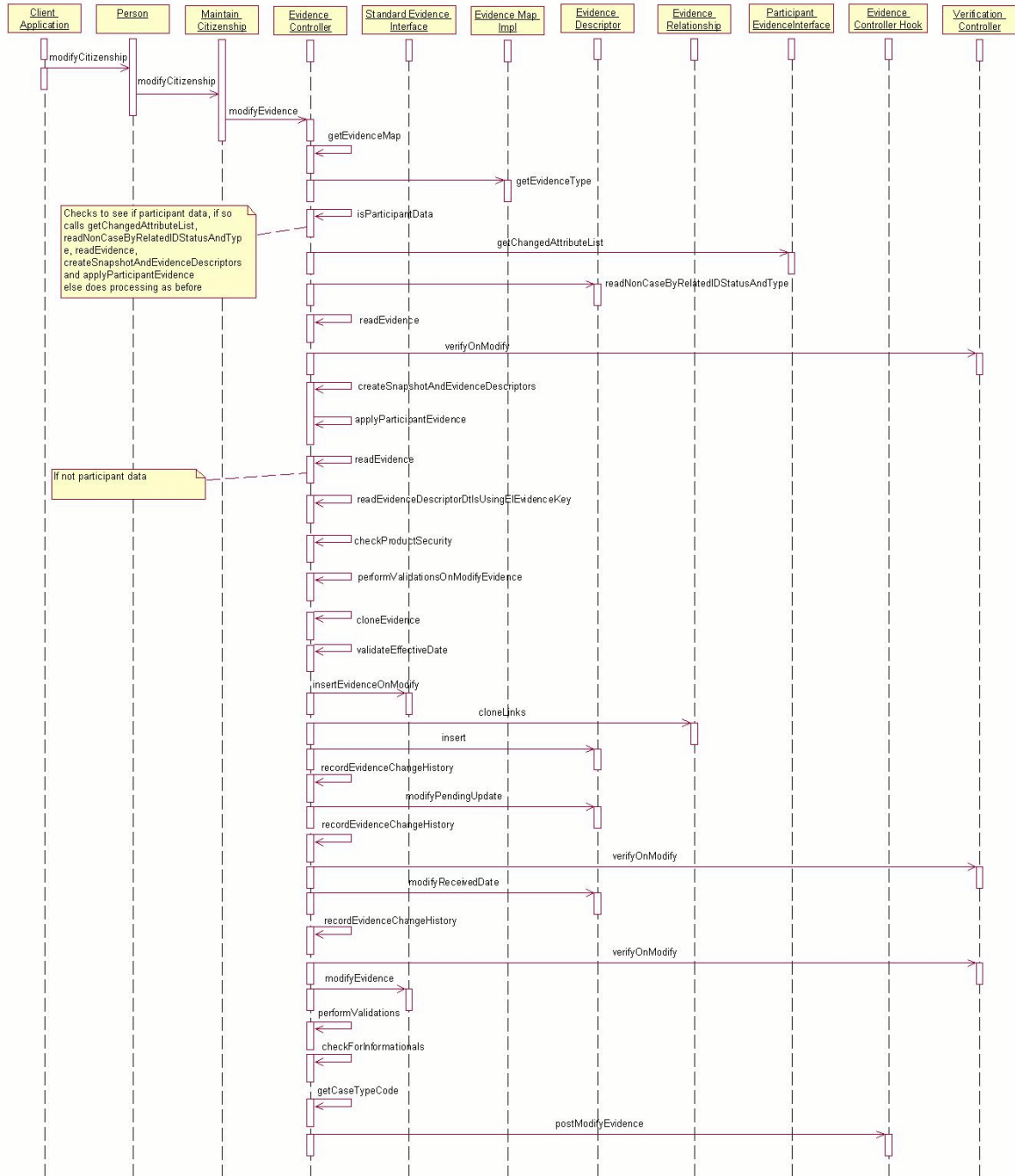


Figure 5: Modify participant

Implementing conditional verifications

Conditional verifications is a feature where flexibility is provided to determine whether verification is applicable for evidence through programmatic support as opposed to manually means.

Conditional verification

The conditional verification programmatic support is encompassed through rule-class implementations and verification for a piece of evidence is determined based on a set of conditions. The Verification Engine checks the conditions that are specified at the time of adding or modifying

evidence. However, the Verification Engine creates an outstanding verification only when a condition that is defined is met rather than every time a verifiable data item is added or modified. The conditions can range from conditions against the value of the verifiable data item to more complex conditions where the values of a set of dependent evidences determine whether verification is required.

Rule artifacts supplied by the verification framework

To facilitate integration between the verification framework and the rule implementations that are supplied by other components, the framework supplies core rule artifacts. The artifacts contain abstract rule classes that other components rule implementations must adhere to.

Rule sets

The rule set `VerificationRuleSet` is available as part of verification framework. The rule set holds all the framework's artifacts, such as the rule classes and the data container classes.

Rule classes

The following three rule classes are available as part of `VerificationRuleSet`.

- `VerificationDeterminator`
- `VerificationDeterminatorResult`
- `VerificationDeterminatorParams`

Verification Determinator

The business logic that determines whether conditional verification is required for particular evidence type goes in the `VerificationDeterminator` rule class. Components creating rule implementations must adhere to the specification by directly or indirectly extending this class. The preceding table lists the attributes that are available in the rule class.

S.No	Rule attribute name	Type	Purpose
1	determine	Verification Determinator Result	The implementation contains the business logic that determines the output of conditional verification. A value of TRUE indicates to the evidence framework that verifications are not applicable for the evidence. A value of FALSE denotes that verifications need to be explicitly added.
2	verificationDeterminatorParams	Verification Determinator Params	The attribute is populated by the conditional verifications framework and contains the values for all the input parameters for an instance.

Verification Determinator Result

The rule class is a data container whose purpose is to store the results of business logic in the Verification Determinator. The class has the following two attributes:

- **Result**: a Boolean that states whether verification is required or not for a given evidence.
- **Reason**: a code table value from `VerificationSkippedReason`, which contains the values of reason for which the conditional verification is not applicable

It is the responsibility of the rule implementations to create, populate, or both, these attributes so that the verification framework, after it examines the state of the attribute, can take appropriate business decisions.

Verification Determinator Params

While it determines whether conditional verification is required or not, the framework will supply various input parameters to the rule implementation classes for various calculation purposes, such as the evidence that is being edited, the associated case identifier for the evidence, and so on. The proceeding table provides complete details of the input parameters.

S.No	Property name	Data type	Description
1	verifiableDataItemName	String	Represents the name of the <i>Verifiable Data Item</i> , such as <i>Person Income</i> , <i>Date Of Birth</i> and so on. The value comes from the code table <i>VerifiableItemName</i> .
2	evidenceDescriptorID	Number	The unique identifier of the applicable evidence record.
3	caseID	Number	The unique identifier of the case with which the evidence is associated.

New propagator

Verifications are applicable to active evidences and to evidences that are in the in-edit state. A new propagator *ActiveInEditEvidenceRowRuleObjectPropagator* is provided for this purpose, which propagates both these evidence types. Use the new propagator to propagate the evidences to the rule data objects that are used in the conditional verification implementation classes.

Customizing dynamic evidence

You can modify the structure or behavior of predefined dynamic evidence types so that customers can customize the types.

Use the proceeding steps to customize default Dynamic Evidence Types. You can also use this information when you upgrade to new versions of Dynamic Evidence Types where the types were customized as part of a project implementation.

The product contains the proceeding components that ship with predefined Dynamic Evidence Types.

- Customization configuration prerequisite
- Customization process
- Evidence gap analysis
- Evidence definition
- Extract evidence
- Source control management
- Pre-production system
- Runtime activation

Typically, customers modify either the structure or behavior of some of the preceding types to fit their specific business requirements before customers deploy the types into a production environment.

Customization configuration prerequisite

To enable Dynamic Evidence customization, the following property must be enabled from the System Administration application:

```
curam.dynamicEvidence.evidencetype.customisation.enabled
```

This property, when set, causes a 'customized Indicator' to be set for Dynamic Evidence Type Versions created or modified by customers. The indicator shows in the **Dynamic Evidence Type Versions list** page in the Administration Suite. In Cúram-shipped Dynamic Evidence Type Versions, the flag is not set. So, customers can, at a glance, see what Dynamic Evidence is and is not customized.

Customization process

Use the following sequence of steps to develop Dynamic Evidence.

1. Evidence Gap Analysis.
2. Evidence Definition
3. Extract Evidence
4. Source Control Management
5. Pre-Production system
6. Runtime Activation

Step 1: Evidence gap analysis

The first step is to identify the evidence requirements for a project, based on program requirements, administrative and legislative policy, and other business needs. Once the evidence requirements are identified, map the evidence requirements onto Cúram-shipped Dynamic Evidence Types.

In some cases, the mapping results in a requirement to create new, project-specific Dynamic Evidence Types to implement the data requirements of their programs. In other cases, customers want to change or augment Cúram-supplied Dynamic Evidence Types.

In either case, Dynamic Evidence Type specifications must be defined that use the Dynamic Evidence Administration pages and the Dynamic Evidence Editor.

For more information, see the *Dynamic Evidence Types* and *Dynamic Evidence Type Versions* related links.

Step 2: Evidence definition

During the second step, the Dynamic Evidence Administration pages are used to either create new Dynamic Evidence Types or to customize existing Cúram-supplied Dynamic Evidence Types.

Create new, project-specific Dynamic Evidence Types and Versions

Create a new Evidence Type and, that uses the Dynamic Evidence Editor, metadata for its Evidence Type Version is created. For more information, see the *Dynamic Evidence Type Versions* related link.

Note: Select the appropriate `curam.dynamicEvidence.type.code.prefix` and `DYNEVDCODE` settings for the new Dynamic Evidence Types. Selecting the appropriate settings ensures that there is no conflict in the Evidence Type Code table entries for the newly defined Dynamic Evidence Types. For more information, see the *Dynamic Evidence Configuration Extractor* related link.

Reuse or modify Cúram-shipped Dynamic Evidence Type Versions

Occasionally, a Cúram-shipped Dynamic Evidence Type Version (for a particular Dynamic Evidence Type) might match customer requirements exactly or it might need a small modification to meet the

requirements. In such cases, you must clone the Cúram-shipped Active Dynamic Evidence Type Version with the **New InEdit Copy** action, even if no change is required. This process might seem counter-intuitive for situations where no changes are needed, but it is important to ensure that your usage of these Dynamic Evidence Type Versions are unaffected by future updates.

The effective date of Cúram-shipped Dynamic Evidence Types is normally set to a date in the distant past, for example, 1 January 1900. You must assign an effective date that is appropriate for the business requirements of the program that is being implemented to the newly cloned Dynamic Evidence Type Version.

If a Cúram-shipped Dynamic Evidence Type is updated by IBM, it is always done with the release of a new Dynamic Evidence Type Version. That is, previously released Dynamic Evidence Type Versions are not changed. The Cúram-shipped Dynamic Evidence Type Version effective date is incremented by one day from the previous version. The new effective date ensures that customers can always analyze the changes and decide whether to apply them to their customized versions.

Where a Cúram-supplied Dynamic Evidence Type is used in a customer program, such as a Product Delivery or Integrated Case, you must cancel all Dynamic Evidence Type Versions that are not required. This cancellation ensures that no Evidence Records are inadvertently recorded against them in a runtime environment. While they exist on a product system, a caseworker can potentially try to enter an Evidence Record for a received date that overlaps with the Cúram-shipped Dynamic Evidence Type Version, resulting in undesired behavior.

Step 3: Extract evidence

When all Dynamic Evidence Types are defined, most customers then want to preserve the state of these Dynamic Evidence Types in their Software Configuration Management system. The Dynamic Evidence Extractor provides the required functions. For more information, see the *Dynamic Evidence Configuration Extractor* related link.

Note: Preserving the state of the defined Dynamic Evidence Types is not mandatory. You can create Dynamic Evidence Types in a testing or staging environment and transport them into a production environment without this step. However, for most of customers this is required.

If a customer extracts their Dynamic Evidence Types, customers must manage the database primary keys and use keys from designated key ranges. If it is not configured, the database just creates arbitrary Primary Keys (PKs) that might result in Primary Key conflicts.

To manage PKs for Dynamic Evidence, use the Range Aware Key Server (RAKS). The mechanism supports Cúram Configuration Transport Manager and is documented in the Cúram Business Object Module Development Guide. The mechanism involves enabling all entities that are being extracted by the extractor to use RAKS and administration configuration of the RAK server.

The benefit of RAKS is that when the system is configured, it is guaranteed to generate the correct PKs for newly created records. Dynamic Evidence Type artifacts can safely be extracted in their current state, without the risk of PK conflicts.

Step 4: Source control management

By now, the customer has a component folder that contains all new and customized Dynamic Evidence Types. When the custom component that contains the extracted artifacts is included in the server component order, rebuilding the Cúram database imports all extracted artifacts in the Cúram system. The contents of the custom component folder can now be placed under source code control.

For more information about component orders, see the *Base directory and directory structure setup* related link.

Step 5: Pre-production system

All Dynamic Evidence Types can now be transported from the Source system to the production system that uses the *Cúram Configuration Transport Manager*.

Transporting modified Dynamic Evidence Type Versions where data exists in respect of them in the target system is not supported. Such modifications would require a new Evidence Type Version where there is no existing live Evidence Record data.

Note: All Dynamic Evidence type versions must be in an **Active** state before transportation.

For more information, see the *Cúram Configuration Transport Manager Guide* relate link.

Step 6: Runtime Activation

The final step in the process is to activate all transported Dynamic Evidence Types in the production system so that they are available for use.

Related concepts

[Dynamic Evidence Types](#)

[Dynamic Evidence Type Versions](#)

[Dynamic Evidence Configuration Extractor](#)

[Base directory and directory structure setup](#)

The base directory for the configuration and metadata must be named `evidence`, and the file `evidence.properties` must exist. The base directory must contain three sub-directories: `properties`, `server metadata`, and `client metadata`.

Related information

[Cúram Configuration Transport Manager Guide](#)

Designing an evidence solution

The evidence solution simplifies the task of designing an evidence solution by providing a balance between reusability and support for customization.

Getting started

The evidence solution reuses components of an evidence solution as much as possible, especially those aspects of evidence maintenance common across all evidence types. Simultaneously, the evidence solution supports customization. It enforces certain design standards that must be addressed at the start of a project.

Modeling the evidence solutions

The most difficult task in designing an evidence solution is to define the required evidence and to model the evidence solution. The task includes the modeling of all the real world events and circumstances that must be captured in order to determine eligibility for benefits and services.

The task of modeling an evidence solution works best as a shared project. Business analysts understand the business requirements, and thus the evidence that must be captured to determine case eligibility. The architect acts as a bridge between the business requirements and the technical components of the evidence solution.

There is no one way to go about modeling evidence solutions. However, certain high-level considerations must be addressed as part of designing an evidence solution. The proceeding are six key high-level considerations.

Gather rules from legislation

The purpose of the task is to gather the rules that are required to determine case eligibility that uses as many primary resources as possible, for example, government forms. Gathering rules from legislation is a time-consuming process that requires business analysts with the knowledge and experience of working with legislation.

Design rule set

The initial design of a rule set tends to be based on legislation research. It is beneficial to involve a rules architect in the designing of rule sets from as early on in the project as possible. The rules architect bridges between the rules legislation and the development of a rule set in the application.

Determine what must be captured to return rule results

During this stage of a project, the focus starts to shift from rules to evidence. Each rule in the rule set requires some piece or even several pieces of evidence to return a rules result. Using a simple example, assume that an income support benefit established an income limit such that any person whose income exceeds this limit is not eligible for income support. Based on this simple example, logically the income amount over time must be captured for all claimants.

Perform gap analysis

The purpose of the gap analysis is to determine what information is already captured in the application and what information still needs to be captured.

Create LDM and page specifications

A logical data model based on the business requirements is constructed. It includes the evidence entities for capturing evidence as well as the categories for these entities and any relationships between them. From this LDM, pages can be designed to capture and maintain the information. The page specifications can also include navigation to related evidence.

Check the soundness of model

Before the evidence requirements are implemented in the application, a technical architect can review the business requirements and check the soundness of the logical data model. This includes reviewing the rule set and checking whether the page specifications and navigation can be used to build a user interface.

In summary, the preceding tasks include three milestones in designing an evidence solution: the completion of a rule set, a logical data model, and a document with page specifications.

Modeling the evidence solution requires two extra tasks: mapping evidence to products and identifying evidence relationships.

Mapping evidence to products

Mapping each evidence type to its products is an important task in designing an evidence solution.

At a high level, mapping each evidence type to its product establishes the different types of evidence that must be maintained for a product in order to determine eligibility. Mapping each evidence type also determines whether an evidence type is shared between multiple products. If shared, then the evidence is best maintained at the integrated case level. If evidence is specific to one product, then that product needs its own site map for maintaining its unique evidence.

The task of mapping evidence to products is important to the evidence framework because of how active evidence is attributed. Evidence is attributed for each evidence type and for each product delivery. So when evidence is activated, the system automatically calculates an attribution period for each of the product delivery cases that share the evidence. If evidence isn't shared, then the evidence is only attributed to the specific product delivery case that the evidence record applies to. Each evidence type has its own algorithm for calculating attribution periods. For more information, see the *Defining algorithms for calculating attribution periods* related link. Mapping evidence types to products determines the products that must be considered in an evidence type's algorithm.

The proceeding table provides a sample to illustrate the mapping of evidence to products. For each evidence type, the table indicates whether the evidence is maintained at the integrated case level. The table also indicates the products that use that evidence to determine eligibility.

Table 18: Sample mapping of evidence to products

Evidence Type	Maintained at the Integrated Case level?	Products
Income	Yes	Foodstamps, Cash Assistance, Medical Assistance
Income usage	Yes	Foodstamps, Cash Assistance, Medical Assistance
Sporting activity	No	Cash Assistance Only
Sporting activity expense	No	Cash Assistance Only

The evidence types income and income usage that are included in the preceding table are maintained at the integrated level and shared between the three products: Foodstamps, Cash Assistance, and Medical Assistance. The two evidence types can be accessed from a site map that is maintained at the integrated case level. However, three separate attribution periods are created for each active evidence record. The algorithm that is used to calculate attribution periods for income and income usage evidence must, therefore, consider any specific attribution requirements for the three products.

The evidence types sporting activity and sporting activity expense that are included in the preceding table are not maintained at the integrated case level. These evidence types must be accessible from an evidence site map specific to Cash Assistance. The algorithm that is used to calculate attribution periods for these two evidence types needs only to consider specific requirements for the Cash Assistance product.

Note: The actual mapping of evidence to products is implemented by using evidence types and the evidence metadata and product evidence link entities. Each evidence entity in the logical data model has its own unique evidence type that is defined in the evidence types code table. The evidence type is associated with one or more evidence metadata records. Evidence metadata records are linked with product evidence link records. When set up as part of evidence administration, all evidence that is associated with a product is maintainable for that product.

Identifying evidence relationships

Evidence relationships can significantly affect the development requirements for an evidence solution. Evidence relationships affect the design of the evidence object tab and custom code that is required to insert an evidence relationship record.

The evidence object tab must contain the subtabs for the related parent or child evidence.

Custom code must be added to an evidence entity's insert method to cater for evidence relationships. This custom code adds a record to the evidence relationship table for each evidence relationship.

To ease the implementation of evidence relationships, identify the relationships between evidence types in the early stages of designing an evidence solution. The proceeding table provides a sample list of evidence types and their relationships.

Table 19: Relationships between evidence types

Evidence Type	Related Evidence Type	Relationship Type
Income evidence	Income usage evidence	Parent-child
Property evidence	Loan evidence	Parent-child
Loan evidence	Loan insurance policy evidence	Parent-child
Property evidence	Loan insurance policy evidence	Grandparent-grandchild
Carer evidence	Benefit evidence	Parent-child (optional)

Each row in the preceding table represents an evidence relationship, with the parent evidence type in the first column and the related child (or grandchild) evidence type in the second column. The second column can be used to determine the evidence types that require custom code to cater for the evidence relationships. For each related evidence type, custom code is added to that evidence type's insert method.

Use the preceding table as a starting point for defining navigation.

Designing the user interface

Designing the user interface involves three main tasks: defining the layout of evidence pages, defining navigation requirements, and determining the details displayed for each evidence type on the screens provided with the evidence framework.

The lengthiest task is to define the layout of the custom evidence pages. The evidence framework makes the task of screen design easier by providing infrastructure support for functionality common across all evidence screens. However, each evidence type still requires custom evidence screens for capturing and maintaining evidence.

Defining the layout of evidence pages

After you develop the logical data model, the next task is to define the layout of evidence pages. Each evidence entity in the logical data model requires its own view, create, and modify screens as well as the evidence type, and evidence object tabs. The screens must include a visualization (both content and layout) of the eventual evidence screens that are to be implemented.

The evidence framework provides evidence views for all view, create, and modify screens, as well as for evidence object, and evidence type tabs. The evidence views include the common evidence information that is maintained for all evidence types. So, each evidence screen must include an evidence view to access the common information. For example, when you are designing a create evidence screen, the create view must be included in the create evidence screen design.

The proceeding table describes the common information that is provided in the view, create, and modify evidence views and the name of the evidence entity from which the common information is retrieved. By including an evidence view in an evidence screen, the common information that is described in this table automatically appears in the evidence screen.

<i>Table 20: Standard information provided in views</i>		
Evidence view	Common information that is provided	Retrieved from the evidence entity
View evidence	Participant ID, Received Date, Effective Date (effective date of change), Status	Evidence Descriptor
	Updated By (username), Updated On	Evidence Change History
	Approval Requested Indicator, Approval Status	Approval Request
Create evidence	Participant ID, Received Date	Evidence Descriptor
Modify evidence	Participant ID, Received Date, Effective Date, Status	Evidence Descriptor
	Updated By (username), Updated On	Evidence Change History

Table 20: Standard information provided in views (continued)		
Evidence view	Common information that is provided	Retrieved from the evidence entity
	Approval Requested Indicator, Approval Status	Approval Request

Determining the details that are displayed on evidence screens

The `get_details_for_list_display` method is required for all evidence types and is used to populate the details field on the evidence list screens, apply evidence changes, and validate evidence changes screens. The details field also appears on a number of evidence approval screens, including both the approve and reject evidence screens.

The purpose of the details field is to provide information significant to an evidence type for standard evidence screens. For example, the details field for income evidence screens might display the income amount and income frequency.

Defining algorithms for calculating attribution periods

A key feature of the evidence framework is how evidence is attributed. The evidence architecture provides infrastructure support for attributing active evidence, thus removing the burden of this complexity from the user.

When you design an evidence solution, an algorithm for calculating attribution periods must be provided for each evidence type. For example, the calculation of attribution periods for the income evidence type might be based on the income frequency that is recorded for an income evidence record.

When evidence is shared across multiple product deliveries within an integrated case, attribution periods are calculated for each product delivery separately. So, the algorithm for an evidence type must consider business requirements that are specific to the evidence type, as well as business requirements specific to each product linked to the evidence type.

Expanding on the previous example, the algorithm for the income evidence type might also consider the delivery patterns for each product that is linked to the income evidence type.

There are extra considerations when you define these algorithms. For example, attribution rules might change over time, for example, rules-based legislation. To support changing attribution rules, Cúram rule sets or rate tables might be used to implement the attribution logic.

Validating evidence

The evidence framework supports adding validations to the insert evidence, modify evidence, and apply evidence changes subpattern for each evidence type. The subpatterns access the informational manager that permits for standard warning and error validations.

Warnings must be used to inform users of important information that must be updated or fixed before the evidence is applied. Errors are used when a process must not be completed until the validations run successfully.

The use of warnings must be considered when you add validations for the insert and modify evidence subpatterns, in particular for evidence relationship validations. This permits the user to focus on the capture of evidence requirements without being forced to follow certain restricted paths in maintaining related evidence. As the activation and removal of evidence can affect case eligibility and determination, all validations for the apply evidence changes subpattern must throw errors rather than warnings.

A benefit to validating evidence is the reduction in errors. Validations can also be used to check for gaps in the evidence and to identify duplications. For example, certain evidence types might require one active evidence record at all times. For example, an active primary address. Validation can be used to ensure that there is no gap, that is, there is always an active address record. To identify duplications, an extra

validation can be implemented that does not permit more than one evidence record to be active at the same time. For example, the primary address evidence record.

As part of designing an evidence solution, it is important to recognize the strong relationship between validations and how the rules loaders must be designed to load the evidence data. For example, consider a validation that states that a person cannot collect income more than one time from the same employer for the same time period. Given this validation, the loader needs only to search for one piece of income data at a point in time. Conversely, if the validation stated that a person might collect multiple salaries from the same employer over the same time period, the rules loader needs to look for a list of items to load.

Additional validation to support apply evidence changes

An extra validation feature is built into the apply evidence changes process. When evidence changes are applied, the system calls a hook that applies validations at the case level rather than at the evidence level. The extra validation ensures that a specified minimum amount of evidence exists for the case. For example, before a loan evidence record is activated, validation might be called as part of the hook that checks for an associated (active) property evidence record.

Validation considerations for calculating attribution periods

There are a number of validation considerations for calculating attribution periods. One consideration is whether evidence can have a null attribution end date. If all records for an evidence type must be attributed to finite periods, then a validation must be added to prevent a null attribution end date.

Another consideration is whether gaps are permitted between attribution periods. If no gaps are permitted, then validation must be added to ensure that these gaps do not occur. Also, validations can be used to ensure that attribution periods do not overlap.

Extending evidence processing

An evidence hook supports the extension of an evidence subpattern with custom functionality. Hooks are provided for inserting, modifying, and removing evidence. A hook is also provided for applying evidence changes.

Evidence hooks can be used to raise workflow events and trigger workflows. For example, the remove evidence process might include an evidence hook that raises a workflow event. The event might trigger a workflow process instance that sends a notification to a caseworker when an active evidence record is flagged for removal by another user. Each time an active evidence record is flagged for removal, the workflow event triggers a new instance of the workflow to notify the relevant caseworker.

External APIs

To create, modify, and read the Dynamic Evidence record, use the external API commands for the `curam.dynamicEvidence.facade.external.impl.DynamicEvidenceMaintenanceExtInf`.

Customer projects can reference external APIs directly. Call the external operations that constitute the official API from your own code.

Create a Dynamic Evidence record

The proceeding sample shows the API code for creating a Dynamic Evidence record. Comments that explain the parameters precede the `createEvidence` method.

```
/**
 * Create Dynamic Evidence record from the contents of
 * * {@link DynamicEvidenceObject} specified. The attributes contained in
 * * {@link DynamicEvidenceObject} should have the
 * * same name and raw data type as those attributes defined in the Evidence Type
 * * Version Definition of the record to be created.
 * *
 * * @param evidenceObject the Dynamic Evidence Object
```

```

* {@link DynamicEvidenceObject}
*
* @return Result containing the new Evidence Id.
* @throws InformationalException generic Informational Exception
* @throws AppException generic Application Exception
*/
public ReturnEvidenceDetails createEvidence(
    final DynamicEvidenceObjectInf evidenceObject) throws AppException,
    InformationalException;

```

Modify a Dynamic Evidence record

The proceeding sample shows the API code for modifying a Dynamic Evidence record.

```

/**
 * Modify the Dynamic Evidence record identified by the evidence case key
 * specified. The contents of the specified DynamicEvidenceObject are used
 * to update the evidence record.
 *
 * @param evidenceObject the evidence object data used to update the evidence
 * record. It is assumed that this parameter is initialized via the read
 * operation and updates to the values are done to that initialized
 * DynamicEvidenceObject.
 * @return details of the modified evidence record
 * {@link ReturnEvidenceDetails}
 *
 * @throws InformationalException a generic Informational Exception.
 * @throws AppException a generic Application Exception.
 */
public ReturnEvidenceDetails modifyEvidence(
    final DynamicEvidenceObjectInf evidenceObject) throws AppException,
    InformationalException;

```

Read a Dynamic Evidence record

The following sample shows the API code for reading a Dynamic Evidence record.

```

/**
 * Read Dynamic Evidence record from the application. The class structure of
 * this object to store the result to be returned should contain each
 * attribute of the same name and raw data type as those attributes defined
 * in the Evidence Type Version Definition.
 *
 * @param evidenceKey the key identifier to the evidence record to be read.
 * @return the Dynamic Evidence Object {@link DynamicEvidenceObject} just
 * read.
 *
 * @throws InformationalException generic Informational Exception
 * @throws AppException generic Application Exception
 */
public DynamicEvidenceObject readEvidence(final EvidenceCaseKey key)
    throws AppException, InformationalException;

```

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 10504-1785 US

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, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

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:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

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.



Part Number:

(1P) P/N: