IBM Cúram Social Program Management
Version 6 Release 0

Health Care Reform Developer Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 49

Revised: March 2014

This edition applies to IBM Cúram Social Program Management v6.0.5 and to all subsequent releases unless otherwise indicated in new editions.

Contents

About this information

Describes how to customize the IBM Cúram Solution for Health Care Reform.

Overview of Health Care Reform support

The Affordable Care Act (ACA) introduced new requirements for states in relation to making affordable healthcare available to state residents. Healthcare is available not just through the existing Medicaid and Children's Health Insurance Programs, but also through the introduction of new programs to provide state residents with help in paying for private health insurance.

In support of the ACA legislation, the IBM Cúram Income Support and IBM Cúram Income Support for Medical Assistance products were extended. These solution modules now support the Health Care Reform provisions of the Affordable Care Act (ACA) with the addition of the Cúram Solution for Health Care Reform.

Intended audience

This publication is intended for developers who are customizing the IBM Cúram Solution for Health Care Reform.

Readers must be familiar with the following topics:

- IBM Cúram Solution for Health Care Reform.
- Cúram Server Developers Guide
- Cúram Server Modeling Guide
- Persistence Cookbook
- Cúram Universal Access Configuration Guide
- Cúram Universal Access Customization Guide
- Working with Cúram Intelligent Evidence Gathering
- Authoring Scripts Using Intelligent Evidence Gathering
- Working with Cúram Express Rules
- Cúram Express Rules Reference Manual
- Inside Cúram Eligibility and Entitlement Using Cúram Rules
- Cúram Dynamic Evidence Configuration Guide
- Cúram Evidence Broker Developers Guide
- Cúram Verification Guide
- Cúram Batch Processing Guide
- Cúram Web Services Guide

Chapter 1. Customizing the Health Care Reform portal

The Health Care Reform portal uses the IBM Cúram Universal Access Motivation infrastructure for the online application processes required by ACA legislation.

Each Health Care Reform motivation is associated with an IEG script, a data store schema, and a display rule set. The following Health Care Reform motivations are available by default:

- Find Assistance
- Browse for plans
- Quick Shopping
- Employer Sponsored Coverage
- Apply for an exemption

For more information about motivations, see the *IBM Cúram Universal Access Customization Guide*.

IEG scripts customization

The default Health Care Reform portal IEG scripts are in the HCROnline component. You can customize the default IEG scripts by creating a custom copy.

For more information about customizing IEG scripts, see the *Authoring Scripts Using Intelligent Evidence Gathering (IEG)* guide.

Eligibility Display Rules customization

When an IEG script completes, the eligibility results page is displayed according to the eligibility results display rules. You can write custom display rules to customize eligibility calculations for the eligibility results page. In addition, Health Care Reform provides several other mechanisms for customizing rule sets.

The default display rules reference the default eligibility rule sets to determine eligibility. The `curam.healthcare.eligibility.ruleset.name` property points to the name of this rule set. You must update this property if custom eligibility rules are to be used.

For information about configuring properties, see "Configuring Application Properties" in the *Cúram System Configuration Guide*.

For information about customizing rule sets in a compliant manner, see the *Curam Express Rules Reference Manual* and the *Cúram Development Compliancy Guide*.

There are several areas in the script rules where you can provide a custom implementation as follows.

Customizing the conditional display of IRS income information

You can customize the eligibility rules that determine the display of retrieved income from the IRS.

About this task

IRS income data that is retrieved for members in a tax household is not displayed if any of the following conditions are true:

- There is more than one financial household within the overall household.
- There are any American Indians or Alaskan Natives in the household.
- The household income is below the Medicaid or CHIP threshold for any of the applicants in the household.

These rules are implemented by the `IRSIncomeDisplayDeterminator` rule class available in the default `HealthCareReformEligibilityRuleset` rule set .

Procedure

1. Create a custom rule class that adheres to the default structure provided in the Abstract Eligibility Rule set, `AbstractEligibilityRuleset.IRSIncomeDisplayDeterminator` This custom rule class must ultimately extend the `AbstractEligibilityRuleset.DefaultIRSIncomeDisplayDeterminator` rule class.
2. Update the `curam.healthcare.displayirsincome.invoking.ruleclass.name` property to point to the fully qualified name of the custom rule class. For example, `MyRuleSet.MyRuleClass`.

Customizing the conditional display of specific questions for Medicaid, CHIP, or IA

You can override the default eligibility rules that determine which specific questions are asked based on eligibility for Medicaid, CHIP, or IA.

About this task

Certain eligibility rules are run as the citizen progresses through the script. These rules control the flow of the script according to the citizen's eligibility for certain programs. When the user enters income information for the household, these rules run. The results of these rules allow the script to ask intelligent questions pertinent to the program for which a household member is considered eligible.

Procedure

1. Create a custom rule class that adheres to the default structure provided in the `AbstractEligibilityRuleset.EligibilityDeterminationCalculator` rule class. This custom rule class must ultimately extend the `AbstractEligibilityRuleset.DefaultEligibilityDetermination` rule class.
2. Update the `curam.healthcare.eligibility.invoking.ruleclass.name` property to point to the fully qualified name of the custom rule class. For example, `MyRuleSet.MyRuleClass`.

Customizing the determination of projected annual income for a citizen

You can override the default eligibility rules that determine the projected annual income for a client.

About this task

Income calculation rules are run after you capture a household member's complete income details, including any deductions or exclusions. The projected annual income is then calculated by rules that are based on these details. The citizen can choose to attest to the determined projected annual income or chose to enter a different value. If the customer enters a different value, the rules take this value into consideration for calculating final eligibility.

Projected annual income is determined by invoking MemberIncomeCalculator available in the default HealthCareReformEligibilityRuleset. The property is "curam.healthcare.memberincome.invoking.ruleclass.name" and is set to a default implementation of the `HealthCareReformEligibilityRuleset.MemberIncomeCalculator` rule class.

Procedure

1. Create a custom rule class that adheres to the default structure provided in the Abstract Eligibility rule set `AbstractEligibilityRuleset.MemberIncomeCalculator`. This custom rule class must ultimately extend the `AbstractEligibilityRuleset.DefaultMemberIncomeCalculator` rule class.
2. Update the `curam.healthcare.memberincome.invoking.ruleclass.name` property to point to the fully qualified name of the custom rule class. For example, `MyRuleSet.MyRuleClass`.

Chapter 2. Integration with external systems

The Health Care Reform solution can call external systems at certain points to gather information necessary for application processing. For example, a call can be made to the Federal Hub to verify SSN and citizenship status for a citizen.

The customization and configuration options for these integration points are as follows:

Customizing the external system implementations

By default, Health Care Reform provides several interfaces and corresponding implementations for integrating with external systems. Customers are free to provide their own implementations for these integration points.

About this task

Note: You might want to customize the default Federal Hub implementation by using the provided customization points.

The following table lists the default external system interfaces, default implementations, and Federal Hub implementations.

Interface	Default Implementation	Federal Hub Implementation
curam.hcr.verification.service.impl. SSACompositeBusinessService	curam.hcr.verification.service.impl. SSAVerificationServiceImpl	curam.hcr.verification.service.impl. FederalSSACompositeServiceImpl
curam.hcr.verification.service.impl. AnnualIncomeDataService	curam.hcr.verification.service.impl. AnnualIncomeDataServiceImpl	curam.hcr.verification.service.impl. FederalAnnualIncomeVerificationServiceImpl
curam.hcr.verification.service.impl. IRSHouseholdDataService	curam.hcr.verification.service.impl. IRSHouseholdDataServiceImpl	No service available
curam.hcr.verification.service.impl. LawfulPresenceVerificationService	curam.hcr.verification.service.impl. LawfulPresenceVerificationServImpl	curam.hcr.verification.service.impl. FederalLawfulPresenceVerificationServiceImpl
curam.hcr.verification.service.impl. MECVerificationService	curam.hcr.verification.service.impl. MECVerificationServiceImpl	curam.hcr.verification.service.impl. FederalMECVerificationServiceImpl
curam.hcr.verification.service.impl. ResidencyVerificationService	curam.hcr.verification.service.impl. ResidencyVerificationServiceImpl	No service available
curam.hcr.verification.service.impl. IncomeDataService	curam.hcr.verification.service.impl. IncomeDataServiceImpl	curam.hcr.verification.service.impl. FederalCurrentIncomeVerificationServiceImpl
curam.hcr.verification.service.impl. CloseDHSCaseService	curam.hcr.verification.service.impl. CloseDHSCaseServiceImpl	curam.hcr.verification.service.impl. FederalCloseDHSCaseService
curam.hcr.verification.service.impl. ESIVerificationService	curam.hcr.verification.service.impl. ESIVerificationServiceImpl	curam.hcr.verification.service.impl. FederalESIVerificationServiceImpl
curam.hcr.verification.service.ridp. fars.impl.FARSVerificationService	curam.hcr.verification.service.ridp. fars.impl.FARSVerificationServiceImpl	curam.hcr.verification.service.ridp. fars.impl.FederalFARSServiceImpl
curam.hcr.verification.service.ridp. primary.impl. RIDPPrimaryRequestVerificationService	curam.hcr.verification.service.ridp. primary.impl. RIDPPrimaryRequest VerificationServiceImpl	curam.hcr.verification.service.ridp. primary.impl. FederalRIDPPrimaryRequestServiceImpl
curam.hcr.verification.service.ridp. secondary.impl. RIDPSecondaryRequest VerificationService	curam.hcr.verification.service.ridp. secondary.impl. RIDPSecondaryRequest VerificationServiceImpl	curam.hcr.verification.service.ridp. secondary.impl. FederalRIDPSecondary RequestServiceImpl

Procedure

1. To create a custom implementation, write a new class that extends one of the external system default implementations.
2. Bind the custom implementation to the corresponding interface by using a Guice module. For example:

```
public class CustomModule extends AbstractModule {
@Override
protected void configure() {
    binder().bind(IncomeDataService.class).to(CustomIncomeDataService.class);
  }

}
```

3. Ensure that the module is added to a custom Module Class Name `.DMX` file. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
  <table name="MODULECLASSNAME">
    <column name="moduleClassName" type="text" />
  <row>
    <attribute name="moduleClassName">
      <value>gov.myorg.CustomModule</value>
    </attribute>
  </row>
</table>
```

Customizing request or response fields for external system calls

You can customize the request and response fields that are used by the external system interfaces by extending the respective request or response classes. You can then use the updated request or response classes in the custom implementation of the external system interface.

Procedure

1. Extend the request or response classes. For example:

```
CustomCitizenshipVerificationRequestDetails
extends CitizenshipVerificationRequestDetails {
//Define custom attributes
//Define getter and Setter methods
}
CustomCitizenshipVerificationResponseDetails
extends CitizenshipVerificationResponseDetails {
      //Define custom attributes
      //Define custom getter and setter methods
}
```

2. Use the updated request or response classes in the custom implementation of the external system interface. For example:

```
CustomCitizenshipVerificationServiceImpl
implements CitizenshipVerificationService {

CustomCitizenshipVerificationResponseDetails
verify(CustomCitizenshipVerificationRequestDetails requestDetails){
}
}
```

External system processors

During the application process, external system Java classes called processors call out to external systems and store the information received in the data store. For example, a call is made to the Federal Hub to verify SSN and citizenship by using the CombinedSSAServiceViewProcessor processor. The response is stored in the data store by the processor and can be used later to facilitate the electronic verification process. By default, the following processors are available:

- `curam.hcr.verification.datastore.impl.CombinedSSAServiceViewProcessor`
- `curam.hcr.verification.datastore.impl.AnnualIncomeViewProcessor`
- `curam.hcr.verification.datastore.impl.CurrentIncomeViewProcessor`
- `curam.hcr.verification.datastore.impl.LawfulPresenceViewProcessor`
- `curam.hcr.verification.datastore.impl.MECViewProcessor`
- `curam.hcr.verification.datastore.impl.RIDPFARSViewProcessor`
- `curam.hcr.verification.datastore.impl.RIDPPrimaryViewProcessor`
- `curam.hcr.verification.datastore.impl.RIDPSecondaryViewProcessor`

Configuring the Federal Hub implementation

By default, all external system calls are routed to the default (empty) implementations for the external system interfaces. Complete the following steps to route the external system calls to the Federal Hub implementations. You must restart the server after updating these values.

About this task

For information about configuring properties, see Configuring Application Properties in the *Cúram System Configuration Guide*.

Procedure

1. Set the `curam.healthcare.test.registerMockExternalSystems` property to `false`.
2. Set the `curam.fed.hub.verification.system.name` property to the Federal Hub system name.
3. Set the `curam.fed.hub.verification.system.registered` property to `true`.
4. Restart the server.

Configuring a State systems implementation

You might want to implement a custom implementation to call State systems as well as calling the Federal Hub.

About this task

For example, you might want to retrieve Current Income from the State Quarterly Wages system, and to fall back on the corresponding Federal Hub service only if the information is not available.

Procedure

Create a custom implementation for the service that first calls the State system, and then calls the Federal Hub implementation.

Customizing electronic verifications

External systems are also used for electronic verification of information that is provided in the application. Health Care Reform provides support for integrating with external systems such as state systems or third-party commercial applications that are identified by states as data sources. You can also customize electronic verification.

By default, Health Care Reform provides processing for Electronic Verification of data such as Citizenship, Residency, or SSN. The framework for Electronic Verification supports adding implementations for custom verification processing for data elements that are either not covered by default processing or those data elements that are added as part of the custom implementation. Also, it is possible to override the default Verification Processing, if needed.

Default verification processors

By default, the following verification processors are available.

- curam.hcr.verification.online.impl.ResidencyVerificationProcessor - Considers the Residency to be verified if it was indicated (isStateResident attribute of the Person data store entity or has address with the state to be configured state) that a Person was a state resident (or) If the Person was indicated to be a state resident and the information that is retrieved about the person from external system (stored in the ExternalSystemResidencyInformation data store entity) also indicates that the person is a state resident. This processing is completed for all the persons who are marked as applicant (isApplicant attribute of the Person data store entity) on the case.
- curam.hcr.verification.online.impl.CitizenshipVerificationProcessor - Considers the Citizenship to be verified if it was indicated (isUSCitizen or isUSNational or lawfullyPresent attribute of the Person data store entity) that a Person was a US citizen or US Nation or Lawfully Present alien and the information that is retrieved about the person from external system (stored in the ExternalSystemCitizenshipInformation data store entity) also indicates that the person citizenship verified. This processing is completed for all the persons who are marked as applicant (isApplicant attribute of the Person data store entity) on the case.
- curam.hcr.verification.online.impl. IncarcerationVerificationProcessor - Considers the Incarceration status to be verified if it was indicated (isIncarcerated attribute of the Person data store entity) that a Person is incarcerated (or) If the Person was indicated to be not incarcerated or incarcerated pending disposition and the information that is retrieved about the person from external system (stored in the RetrievedPersonInformation data store entity) also indicates the same. This processing is completed for all the persons who are marked as applicant (isApplicant attribute of the Person data store entity) on the case.
- curam.hcr.verification.online.impl.HouseholdSSNVerificationProcessor - Considers the SSN to be verified if the SSN was provided (`ssn` attribute of the Person data store entity) and the information that is retrieved about the person from the external system (stored in the ExternalSystemSSNInformation data store entity) also indicates that the given SSN was verified. This processing is completed for all the persons who are marked as applicant (isApplicant attribute of the Person data store entity) on the case.
- curam.hcr.verification.online.impl.IncomeVerificationProcessor - Considers the Income data to be verified if the Income was provided(IncomeItem data store entity has records) and the information that is retrieved about the person from

the external system (store in the IRSAnnualTaxReturn or ExternalSystemIncome data store entity) are reasonably compatible/E verified. This processing is completed for all the persons.

- curam.hcr.verification.online.impl. MECVerificationProcessor - Considers the MEC to be verified if the person indicated to not receiving benefits (isReceivingBenefits attribute of the Person data store entity) and the information that is retrieved about the person from the external system (stored in the ExternalSystemMECDetails data store entity) also indicates the same. This process is completed for all the persons.

Adding custom verification processing

Complete the following steps to add custom verification processing.

Procedure

1. Edit `CT_VerificationItemType.ctx` to add an entry to the VerificationItemType code table.
2. Create an implementation of the curam.hcr.verification.online.impl.VerificationProcessorinterface. Ensure that the getVerificationType() API returns the code table code you have added.
3. Install the custom implementation by using a custom Guice module. The custom Verification Processing implementation can be bound by using a Guice Set MultiBinder. For example:

```
public class CustomModule extends AbstractModule {
  @Override
  protected void configure() {
    Multibinder<VerificationProcessor> binder = Multibinder.newSetBinder(
       binder(), VerificationProcessor.class);
      binder.addBinding().to(CustomVerificationProcessor.class);
  }
}
```

4. Add an entry that contains the custom Guice module name to a `.DMX` file for ModuleClassName entity.

Overriding the default verification processing

Complete the following steps to override the default verification processing.

About this task

Each entry in the VerificationItemType represents a kind of data item, such as Citizenship.

Procedure

1. Review CT_VerificationItemType.ctx to identify the code for the data item type for which the default processing must be overridden.
2. Create an implementation of the curam.hcr.verification.online.impl.VerificationProcessorinterface. Ensure that the getVerificationType() API returns the code table code you have identified.
3. Install the custom implementation by using a custom Guice module. The custom Verification Processing implementation can be bound by using a Guice Set MultiBinder. For example:

```
public class CustomModule extends AbstractModule {
  @Override
  protected void configure() {
    Multibinder<VerificationProcessor> binder = Multibinder.newSetBinder(
        binder(), VerificationProcessor.class);
      binder.addBinding().to(CustomVerificationProcessor.class);
  }
}
```

4. Add an entry that contains the custom Guice module name to a `.DMX` file for ModuleClassName entity.

Chapter 3. Customizing case management

You can customize Health Care Reform case management artifacts such as dynamic evidence, eligibility rule sets, and conditional verifications.

Dynamic evidence customization

Health Care Reform ships with a number of dynamic evidence configurations in the HCR component. The Health Care Reform dynamic evidence configurations model information that is captured and maintained for the various ACA programs.

For information about customizing dynamic evidence, see the *Cúram Dynamic Evidence Configuration Guide*.

Eligibility Rules customization

Health Care Reform ships with a default set of eligibility rule sets in the HCR component. You can customize these eligibility rules for your custom requirements.

For information about customizing eligibility rules, see the *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules guide*.

For information about compliantly customizing the default rule sets, see the *Cúram Development Compliancy Guide*.

Conditional verifications customization

Health Care Reform application cases and integrated cases are configured to use the verification framework. You can customize conditional verification rule sets in the same way as other rule sets.

For more information about configuration of verifications and conditional verifications, see the *Cúram Verification Guide*.

For information about compliantly customizing the default rule sets, see the *Cúram Development Compliancy Guide*.

Chapter 4. Customizing plan management

Complete the following tasks to customize the default plan management implementation.

Integration with Plan Management

When a citizen applies for insurance affordability assistance through Cúram, they must go to a plan management vendor's website to view and purchase plans. To facilitate this access, you must integrate a plan management vendor with the Cúram application. You can integrate with the plan management vendor of your choice.

Important: IBM Cúram implements a vendor-agnostic approach to plan management integration and does not include an implementation of the plan management adaptor in the product. Each project is responsible for implementing their own integration between the Cúram system and the plan management system of choice.

Plan management integration is accomplished with a combination of both user interface and web services integration.

A plan management vendor's user interface is shown in an inline frame on a Cúram page.

Information is exchanged between Cúram and the plan management vendor through two categories of web services:
- Web services that are owned by Cúram (inbound)
- Web services that are owned by the plan management vendor (outbound)

This approach allows the citizen to enroll on a plan on the plan management vendor's system with the eligibility information that is determined on the Cúram side. In addition, Cúram can query the plan management vendor's web services to read and store any plans in which a citizen enrolls.

The plan management adapter interface

A plan management interface is provided which customers must implement. The custom implementation allows customers to communicate with their chosen plan management vendor through web services.

The methods in the interface are called at different points during processing. For example, the getEnrollmentDetails() method is called to determine the plan details after a citizen successfully enrolls on a plan in the plan management system.

A default `curam.planmanagement.adapter.impl.PlanManagementAdapterDefault` implementation of the plan management adapter interface is provided. To provide some insulation from future changes, extend this class instead of directly implementing the interface.

`curam.planmanagement.adapter.impl.PlanManagementAdapter`
- `getBenchmarkPlanDetails()`

Retrieves the benchmark plan amount and essential health benefit premium amount from a plan management vendor.

- `getEnrollmentDetails()`

  Retrieves the enrollment details for a completed enrollment. For example, the enrolled plan details.
- `getAvailableEmployerPlanDetails()`

  Retrieves the available employer insurance plans for an employee.
- `getBenchmarkPlanDetailsForBenefitMembers()`

  Retrieves the benchmark plan amount and essential health benefit premium amount from a plan management vendor.
- `updateEntitlementDetails()`

  Informs the plan management vendor of a change in entitlement for a specific enrollment.
- `getPlanUpdates()`

  Retrieves any updates to plans for an enrollment, typically called during re-enrollment.
- `continueEnrollment()`

  Informs the plan management vendor that an existing enrollment on a plan is to be continued, typically called during the re-enrollment period.
- `getPolicyID()`

  Retrieves the policy identifier for a specific enrollment.
- `getEmployerOpenEnrollmentDetails()`

  Retrieves the open enrollment details for an employer.

Note: For more information about the plan management adapter interface, see the Javadoc in the `HCR` component.

Configuring the plan management adapter

The custom plan management adapter typically communicates with a plan management vendor over a web service with stubs generated from the plan management vendor's WSDL file.

Procedure

1. Create a directory named axis in a custom component.
2. Add a `ws_outbound.xml` file to this directory. This file must reference the WSDL file that is provided by a plan management vendor. , For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
<service
location="components/CustomComponent/axis/PlanMgmtWebService/PlanManagementVendor.wsdl"
    name="PlanManagementVendor"
  />
</services>
```

3. From a command prompt under the `EJBServer` directory, run `build wsconnector2` to generate the stubs to the `build` directory. These stubs are now available to call in the custom PlanManagementAdapter implementation.
4. Create an implementation of the plan management adapter interface and bind it using a Guice module. For example:

```
@Override
protected void configure() {
  bind(PlanManagementAdapter.class).to(CustomPlanManagementAdapter.class);
}
```

For more information about bindings in Guice, see the *Persistence Cookbook*.

5. Code the custom implementation of the plan management adapter by using the generated stubs.

   For more information about web services in Cúram, see the *Cúram Web Services Guide*.

Plan management web services provided by Cúram

A plan management vendor must call Cúram web services to be able to populate their screens and to carry out plan management processing.

For example, when a household is enrolling on a plan in the plan management vendor's system, the vendor requires details about the household such as names, date of births, address, and eligibility information. Cúram provides the retrieveDemographicsAndEligibilityDetails() web service for this purpose.

The following web services are provided:

`curam.planmanagement.adapter.intf.HealthCareWebService`

- `retrieveDemographicsAndEligibilityDetails()`
- `getHouseholdSummaryDetails()`
- `getEntitlementDetails()`
- `policyIDAvailable()`
- `updateEmployerEnrollment()`

For more information about these web services, see the Javadoc in the HCR component.

Related concepts:

"Health Care Reform web services" on page 16
The web services that are available for Health Care Reform.

Configuration parameters for plan management

The following configuration properties exist for plan management integration.

Property	Description
curam.healthcare.planManagementVendorUrl	The plan management vendor URL for the main find assistance flow. A unique enrollment identifier is appended to this URL.
curam.healthcare.planManagementVendorBrowseForPlansUrl	The plan management URL used to allow a citizen to browse for (but not purchase) insurance plans. A unique enrollment identifier is appended to this URL.
curam.healthcare.planManagementVendor EmployerCoverageUrl	The plan management URL used to allow employees to shop for insurance plans provided by their employer. A unique enrollment identifier is appended to this URL.

Property	Description
curam.healthcare.planManagementVendorAvailable	This property indicates whether a plan management vendor is available. By default, it is set to `false` to enable testing but must be set to `true` when integrated with a plan management vendor.

Callback URLs for plan management

Callback URLs are the URLs that a plan management vendor uses to return control to the Cúram user interface. For example, after an enrollment completes, a callback URL is used to redirect back to the Cúram results page.

The default callback URLs are listed in this table.

Callback URL	Description
https://<host>:<port>/CitizenPortal/en_US/HealthCare_finishEnrollmentPage.do?o3ctx=4096	A plan management vendor redirects to this URL upon successful completion of an enrollment.
https://<host>:<port>/CitizenPortal/en_US/HealthCare_saveAndExitEnrollmentPage.do?o3ctx=4096	A plan management vendor redirects to this URL if a user chooses to save and exit from the plan management vendor's screens. This option would enable a user to resume the enrollment later.
https://<host>:<port>/CitizenPortal/en_US/HealthCare_cancelEnrollmentPage.do?o3ctx=4096	A plan management vendor redirects to this URL if a user chooses to cancel/quit from the plan management vendor's screens.

Batch processing for plan management

The following plan management batch processes are available.

For more information about batch processes, see the *Cúram Batch Processing Guide*.

Employer enrollment notification batch process

The purpose of this batch process is to generate notifications for employees to indicate that the open enrollment period for their employer is about to begin.

This batch process looks at active EmployerEnrollment records on the database. For each one, it calls out to the plan management vendor by using the curam.planmanagement.adapter.impl.PlanManagementAdapter. getEmployerOpenEnrollmentDetails() API. Using the response from the plan management vendor, a pro-forma communication is generated and stored against each employee returned.

Plan management web service API reference

The plan management web services that are available for the IBM Cúram Solution for Health Care Reform and the schema that is used for the data.

Health Care Reform web services

The web services that are available for Health Care Reform.

Related concepts:

"Plan management web services provided by Cúram" on page 15
A plan management vendor must call Cúram web services to be able to populate their screens and to carry out plan management processing.

retrieveDemographicsAndEligibilityDetails

A plan management vendor requests eligibility details for an enrollment. The eligibility details and details for each person in the enrollment are returned from IBM Cúram Health Care Reform.

Table 1. Request

Data Member	Type	Description
EnrollmentDetails	EnrollmentDetails	The health care retrieve eligibility request that contains the enrollment ID.

Table 2. Response

Data Member	Type	Description
EligibilityAnd DemographicDetails	EligibilityAnd DemographicDetails	Response containing eligibility details, details about each person in the enrollment group, previous enrollments for each person that is being enrolled and details about assistors.

getEntitlementDetails

A plan management vendor calls the IBM Cúram Health Care Reform solution to get updated entitlement details for an existing enrollment.

Table 3. Request

Data Member	Type	Description
EnrollmentDetails	EnrollmentDetails	The health care retrieve eligibility request that contains the enrollment ID.

Table 4. Response

Data Member	Type	Description
EntitlementUpdateDetails	EntitlementUpdateDetails	Response containing the updated tax credit amount

getHouseholdSummaryDetails

A plan management vendor calls the IBM Cúram Health Care Reform solution to notify any change in the status of an existing enrollment. For example, when a carrier finishes processing the enrollment and made a policy ID available.

Table 5. Request

Data Member	Type	Description
EnrollmentDetails	EnrollmentDetails	Contains the enrollment ID for which the request is being made

Table 6. Response

Data Member	Type	Description
HouseholdSummaryDetails	HouseholdSummaryDetails	Response containing eligibility details, details about each person in the enrollment group, previous enrollments for each person that is being enrolled and details about assistors.

policyIDAvailable

A plan management vendor calls the IBM Cúram Health Care Reform solution to notify that a carrier has finished processing the enrollment and made a policy ID available.

Table 7. Request

Data Member	Type	Description
EnrollmentDetails	EnrollmentDetails	Contains the ID of the enrollment for which a policy ID is available.

updateEmployerEnrollment

A plan management vendor calls this API to notify the agency that the open enrollment period has begun for a specific employer.

Table 8. Request

Data Member	Type	Description
EmployerEnrollment	EmployerEnrollment	Contains the employerEnrollmentID for the employer with an open enrollment period.

Table 9. Response

Data Member	Type	Description
EmployerEnrollmentReceived	EmployerEnrollmentReceived	An indicator that represents successful receipt and storage of the employer identifier.

Health Care Reform schema elements

The schema that is used for Health Care Reform data.

Table 10. EnrollmentDetails

Data Member	Type	Description
enrollmentID	Long	The enrollment key.

Table 11. EligibilityAndDemographicDetails

Data Member	Type	Description
eligibilityDetails	eligibilityDetails	
persons	persons	

Table 11. EligibilityAndDemographicDetails (continued)

Data Member	Type	Description
previousEnrollments	previousEnrollments	
assistors	assistors	
employerDetails	employerDetails	

Table 12. eligibilityDetails

Data Member	Type	Description
program	String	Values are as follows: EP1 Insurance Assistance EP2 CHIP EP3 Medicaid EP4 State Basic Plan EP5 None (for when the household is just shopping for plans)
maxPremiumTaxCredit	Double	
maxPremiumTaxCreditAnnual	Double	The amount of premium tax credit that remains for the year.
monthsRemaining	Int	The number of months that remains in the plan year.
costSharingSubsidy	Double	
premiumPayment	Double	
maximumCoPay	Double	
stateSubsidy	Double	
enrollmentPeriod	String	Values are as follows: EPD1 Open EPD2 Special
coverageStartDate	Date	
coverageEndDate	Date	

Table 13. persons

Data Member	Type	Description
person	List of person	

Table 14. person

Data Member	Type	Description
personID	Long	Unique identifier for a person within the exchange
ssn	String	
firstName	String	

Table 14. person (continued)

Data Member	Type	Description
middleName	String	
lastName	String	
dateOfBirth	Date	
gender	String	Values are as follows: SX1 Male SX2 Female
tobaccoUser	Boolean	
coverageCategory	String	Values are as follows: CC1 Parent/Caretaker CC2 Pregnant Woman CC3 Adult CC4 Child
address	Address	
phoneNumber	PhoneNumber	
emailAddress	String	
nativeAmerican	Boolean	Indicates whether the person is an American Indian or Alaskan Native.
isPrimaryContact	Boolean	Indicates whether the person is the primary contact for the group that is being enrolled
costSharingEliminated	Boolean	True for AI/NA individual with household income less than or equal to 300% of FPL
subscriberID	Long	Unique identifier of the primary client that is assigned to each member.
taxFilerRelationshipList	TaxFilerRelationshipList	

Table 15. Address

Data Member	Type	Description
addressLine1	String	
addressLine2	String	
city	String	
county	String	
state	String	
zip	String	

Table 16. TaxFilerRelationshipList

Data Member	Type	Description
taxFilerRelationships	List of TaxFilerRelationship	

Table 17. TaxFilerRelationship

Data Member	Type	Description
relatedPersonID	Long	
taxFilerRelationshipType	String	Values are as follows: TFRT26001 Dependent TFRT26002 Spouse TFRT26003 Tax Filer

Table 18. previousEnrollments

Data Member	Type	Description
enrollment	List of enrollment objects	

Table 19. enrollment

Data Member	Type	Description
enrollmentID	Long	
planID	String	
policyID	String	
coverageEndDate	Date	
previousPremium	Double	
previousTaxCredit	Double	
previousEnrollees	previousEnrollees	

Table 20. previousEnrollees

Data Member	Type	Description
enrollee	List of enrollee objects	

Table 21. enrollee

Data Member	Type	Description
personID	Long	

Table 22. assistors

Data Member	Type	Description
assistor	List of assistor objects	

Table 23. assistor

Data Member	Type	Description
firstName	String	
lastName	String	
address	Address	
phoneNumber	PhoneNumber	
certificationNumber	String	
assistorType	String	

Table 23. assistor (continued)

Data Member	Type	Description
assistorID	Long	
agencyOrganisationID	Long	

Table 24. PhoneNumber

Data Member	Type	Description
countryCode	String	
areaCode	String	
phoneNumber	String	
Extension	String	

Table 25. EmployerDetails

Data Member	Type	Description
employerID	Long	
coverageStartDate	Date	

Table 26. EntitlementUpdateDetails

Data Member	Type	Description
enrollmentID	Long	
updatedPremiumTaxCredit	Double	

Table 27. HouseholdSummaryDetails

Data Member	Type	Description
effectiveDate	String	
zipCode	String	
personList	PersonList	

Table 28. PersonList

Data Member	Type	Description
persons	List of Person	

Table 29. person

Data Member	Type	Description
dateOfBirth	Date	
tobaccoUser	Boolean	
isPrimaryContact	Boolean	Indicates whether the person is the primary contact for the group being enrolled

Table 30. EmployerEnrollment

Data Member	Type	Description
employerEnrollmentID	String	The employer enrollment identifier.

Table 31. EmployerEnrollmentReceived

Data Member	**Type**	**Description**
employerEnrollmentReceived	Boolean	Indicates the employer enrollment identifier has been successfully received and stored.

Chapter 5. Customizing change of circumstances

To customize change of circumstances for your environment, you must be familiar with the default implementation. Use this information to understand the process flow, and to identify the steps that you must complete to customize your system.

Change of circumstances process flow

Use this information to understand how the components work together to handle changes in client circumstances.

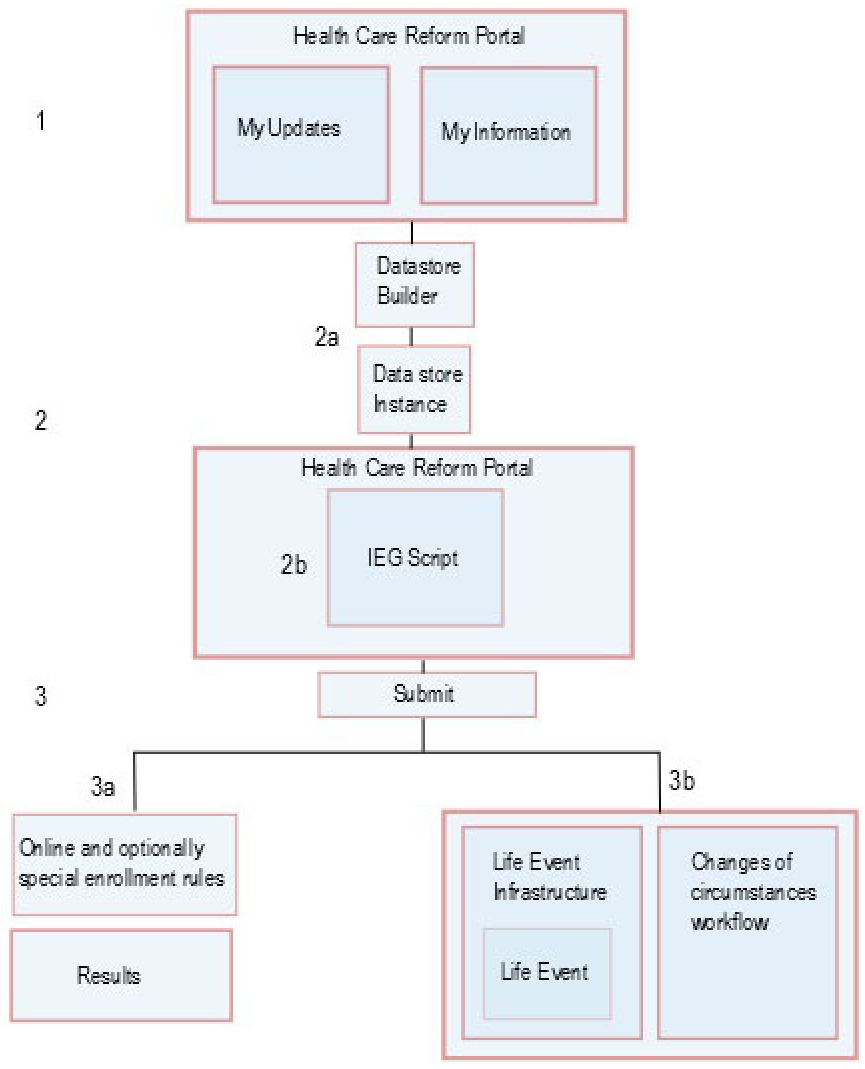
1
Health Care Reform Portal
My Updates
My Information
Datastore Builder
2a
Data store Instance
2
Health Care Reform Portal
2b
IEG Script
3
Submit
3a
3b
Online and optionally special enrollment rules
Results
Life Event Infrastructure
Life Event
Changes of circumstances workflow

Figure 1. Change of circumstances process flow

1 A citizen with an existing application logs in to the Health Care Reform portal

They can see a read-only summary of some of their evidence, such as SSN, Address, Household Members, and Income by clicking the **View your information to provide updates** link on the landing page, or the **My Information** menu option. They can also see the history of their submitted life events in **My Updates**.

The read-only data that is shown is the most current information for each evidence type, specifically the most recent active evidence. If the citizen has recently created an in-edit version of evidence by a previous change of circumstances, that in-edit version is displayed instead.

2 The citizen decides to update their data

After they review the information, the citizen can click the **Update My Information** link to update their information. This link is only available if there is no outstanding change of circumstances for the citizen. Clicking the link starts the following processing:

2a The change of circumstances Datastore Builder retrieves the evidence from the ongoing Insurance Affordability integrated case and creates a data store instance for the data retrieved. This data store instance becomes the data store used for the change of circumstances IEG script.

2b The change of circumstances IEG script opens with the data pre-populated for the citizen to make the required changes. The citizen can add, update, or remove data. Remove refers to end-dating particular evidence types. The citizen continues through the script and completes their updates.

3 The citizen submits their change of circumstances updates

When the citizen clicks submit, the following processing starts:

3a Online and special enrollment rules

HCR Online rules, and optionally special enrollment rules, are run to generate a results page for the citizen. Citizens can only enroll on Advanced Premium Tax Credit (APTC) plans outside the configured open enrollment period if they meet the special enrollment criteria. A set of special enrollment rules are run to determine whether the reported change qualifies an individual for special enrollment.

A results page with the outcome of those rules is displayed to the citizen. Depending on the results, the citizen can proceed to enrollment.

3b Life Event Infrastructure

The change of circumstances process uses the Life Event infrastructure as the mechanism for updating the ongoing Insurance Affordability case with the new or modified data that is supplied by the citizen. When the life event associated with the change of circumstances is submitted, the following processing is triggered:

1. The state of the life event is updated to pending. The **Update My Information** link is disabled to prevent the citizen from triggering a second change of circumstances while there is still one in progress.
2. The change of circumstances workflow is started.

Change of circumstances workflow

The default change of circumstances workflow contains the following process steps.

Case and Participant processing

This step creates new household members as participants on the Participant Data Case.

Evidence Updater

This step adds or updates evidence directly on the ongoing case as in-edit evidence. An attempt is made to add or update the evidence with validations turned on. If this attempt fails, an attempt is made to add this evidence with validations turned off. The following scenarios are catered for:

1. New evidence is added. In this case, the new evidence is written as in-edit evidence.
2. A citizen updates evidence, such as income, and specifies when the update became effective. In this case, a new record is created in the succession on the integrated case.
3. A citizen updates an in-edit record that was previously added by the citizen In this case, the evidence is updated in place.
4. A citizen updates an existing active record that has an in-edit version that was not entered by the citizen. In this case, the data is not written but is captured in a case note for resolution by the case worker.

Evidence Corrections

If validation errors occur when evidence is added, a manual task is created and assigned to the ongoing case owner. This task contains two links:

- Primary Link 1 goes to the evidence workspace to facilitate the case worker in resolving the issues reported.
- Primary Link 2 (Case Note) allows the case worker to resolve any issues that were captured in a case note.

Post Evidence Mapping

By default, this mapping sets the life event state to `Completed`.

Customizing the default change of circumstances implementation

Complete the following steps to customize the default change of circumstances implementation to suit your custom environment.

Before you begin

You must complete a full analysis of your requirements, and identify the information that you want citizens to be able to modify.

Procedure

1. Customize the default change of circumstances IEG script and data store schema.

```
/components/HCROnline/data/initial/clob/ChangeOfCircumstance.xml
/components/HCROnline/data/initial/clob/ChangeOfCircumstance.xsd
```

   a. In most cases, you are updating the default change of circumstances IEG script to align it with the existing enrollment and internal case worker scripts. For example, you might want to make one of the following changes:
      - Create a custom evidence entity for which you want to capture data.
      - Customize a default evidence entity, typically by adding one or more attributes.
      - Customize the flow of the script. For example, by modifying control questions.

   - Customize the script to facilitate adding, updating, or removing evidence for a newly added evidence type.

   b. Depending on the changes to the script, you might need to make parallel changes to the schema that is associated with the script.
2. Configure the change of circumstances life event to call your custom script by overriding the change of circumstances entry in the following files:
   ```
   /components/HCROnline/data/initial/LifeEventContext.dmx
   /components/HCROnline/data/initial/LifeEventType.dmx
   ```
3. To add a custom evidence entity, you must write new prepopulator and updater implementations.
   a. A prepopulator takes the data from a dynamic evidence instance and puts that data into data store format so that it can be read by the script. For information about configuring a new prepopulator, see the Javadoc of the following class:
   ```
   curam.healthcare.lifeevents.coc.prepopulators.impl.Recertification
   ```
   b. An updater, or mapper, takes the data store information after a change of circumstance and identifies which evidence on the ongoing case needs to be added, modified or removed. For information about configuring a new updater, see the Javadoc of the following class:
   ```
   curam.healthcare.lifeevents.coc.mappers.impl.LifeEventDefaultEvidenceMapper
   ```
4. If you are extending a default entity, you must extend the provided prepopulator and mapper classes that are associated with this type and add the custom code. For a list of all of the prepopulator and mapper classes that can be extended, see the following class:
   ```
   curam.healthcare.lifeevents.impl.Module
   ```
5. Configure the online and program group logic rules to reflect your custom changes. The existing portal and case management rule-sets were updated to cater to change of circumstances.
   You can find the online rules in the following location:
   ```
   ./EJBServer/components/HCROnline/CREOLE_Rule_Sets/HealthCareReformEligibilityRuleset.xml
   ```
   You can find the main program group logic rules in the following location:
   ```
   ./EJBServer/HCR/CREOLE_Rule_Sets/HCRProgramGroupRuleSet
   ```
   You can find a rule set per program in the following location:
   ```
   ./EJBServer/HCR/CREOLE_Rule_Sets
   ```
6. Thoroughly test the custom changes made to the change of circumstances process. You must ensure the following results:
   - The correct online results are been achieved.
   - The correct information is being written to the ongoing case.
   - The correct program group logic results are being achieved.
7. Customize the change of circumstances workflow.

Customizing the change of circumstances IEG script

You can complete one or more of the following tasks to customize the change of circumstances script for your custom environment.

About this task

The change of circumstances script starts with a summary page. From this summary page, all of the necessary change of circumstance actions can be done.

For example, add, modify and remove, where remove refers to the end-dating of evidence.

Adding custom entities through the change of circumstances script

To add custom entities through the change of circumstance script, you must make the following changes to the script.

Procedure

1. Provide an **Add** link on the summary page. This link must point at an existing page.
2. Add a new data store entity to reflect the new custom evidence entity.
3. Add an attribute called evidenceCoCStatus to this data store entity. This attribute is based on the code-table EVIDENCECOCSTATUS, which contains the following values:
   - ADDED
   - MODIFIED
   - REMOVED

   The default for this data store schema attribute is a blank value.
4. Set the newly added attribute to ADDED after the IEG page that gathers the data for this new entity has been submitted. This is achieved through invoking the UpdateEvidenceCoCStatus custom function which takes the name of the entity as a parameter.
5. This attribute can be used, as follows, in conditions to display or hide data:
   ```
   "IsRecordAdded() or <MyEntity>.evidenceCoCStatus=="ADDED""
   ```
6. The evidence updater can use the value of this attribute to determine any data store entity that needs to be added as evidence. The ADDED status can also be deduced using the localID for the entity in question as this will not have been set for newly added entities. The localID attribute is used to hold the unique identifier of evidence on the database.

Modifying entities through the change of circumstances script

To modify entities through the change of circumstance script, you must make the following changes to the script.

Procedure

1. Provide a **Change** link on the summary page. This link must point at an existing page.
2. Set the evidenceCoCStatus attribute to MODIFIED. This is achieved by comparing the attributes. New validations are added that call a custom function (HasChanged or HasAttrValueChanged, which always return true). The parameters are the new value and the fully qualified attribute name. This function can deduce if the attribute has changed by looking up its original value.
3. An additional boolean attribute dataSubmitted, which defaults to false, will be added to the schema on the given data store entity. It is needed in case other page validations fail. The custom function SetDataSubmitted will be called in the last validation, setting the flag to true. This has the effect of resetting evidenceCoCStatus to a blank value if this flag is set. The flag will be reset to false in the custom function following the page, UpdateEvidenceCoCStatus.

Removing entities through the change of circumstances script

To remove entities through the change of circumstance script, you must make the following changes to the script.

Procedure

1. Provide a **Change** link on the summary page. This link must point at an existing page.
2. Set the evidenceCoCStatus attribute to REMOVED. This is achieved by comparing the attributes. New validations are added that call a custom function (HasChanged or HasAttrValueChanged, which always return true). The parameters are the new value and the fully qualified attribute name. This function can deduce if the attribute has changed by looking up its original value.
3. An additional boolean attribute dataSubmitted, which defaults to false, will be added to the schema on the given data store entity. It is needed in case other page validations fail. The custom function SetDataSubmitted will be called in the last validation, setting the flag to true. This has the effect of resetting evidenceCoCStatus to a blank value if this flag is set. The flag will be reset to false in the custom function following the page, UpdateEvidenceCoCStatus.

Customizing the change of circumstances workflow

You can use the following steps to customize the default change of circumstances workflow.

Before you begin

You can find the change of circumstances workflow in the following location:

`./EJBServer/components/HCROnline/workflow/ChangeOfCircumstances_v1.xml`

Procedure

1. If you want to or add or remove steps, or to change the flow structure of the existing workflow, create a version of the workflow to make your custom changes. Customize the change of circumstances workflow in the standard supported fashion of customizing workflows as follows:
   a. Using the PDT, view the latest version of the process definition that requires modification. Create a new version of that process definition using the tool.
   b. Make the changes, validate it and release the workflow.
   c. Export the newly released workflow process definition using the PDT and place it into the workflow subdirectory of the `...\EJBServer\components\custom` directory.
2. If you are happy with the structure and the steps in the default workflow, you can implement your own version of each step. Use the provided hook points for each step in the workflow to implement your own version of a step. Add your own implementations for each of the steps. Then, bind those new implementations by using Guice to ensure that they are called as each step in the workflow is called. The following abstract classes are available:

   Case and Participant Processing step
   curam.healthcare.lifeevents.coc.sl.impl.CaseAndParticipantProcessing

   Evidence Updater step
   curam.healthcare.lifeevents.coc.sl.impl.EvidenceUpdater

Post Evidence Updater step
: curam.healthcare.lifeevents.coc.sl.impl.PostEvidenceUpdater

3. The Evidence Corrections step is a manual activity in the change of circumstances workflow. If you want to change this step, update the workflow process definition.

Chapter 6. Monitoring Cúram processes

Use the following Cúram views to monitor and troubleshoot problems with process instances and to see process instance errors.

About this task

Use these views to see workflow processes and see specific errors in workflow and deferred processes. Plan to monitor the information in the following locations regularly for potential errors or exceptions. You can troubleshoot problems by steps such as suspending process instances or overriding event waits, or by retrying or aborting failed workflow process instances.

Application intake process overview

Use this information to understand how the components work together to intake and process citizen information and deliver the appropriate insurance assistance.

For documentation purposes, the process diagram is split into two parts.

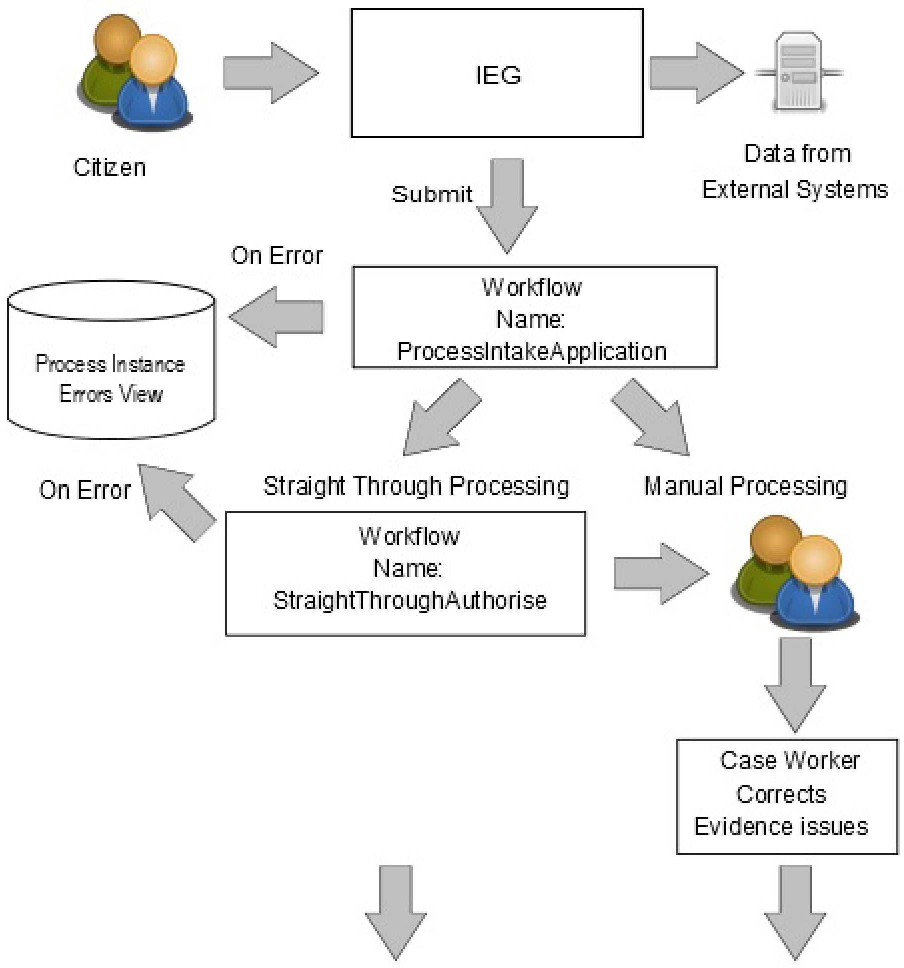
Citizen
IEG
Data from External Systems
Submit
On Error
Workflow Name: ProcessIntakeApplication
Process Instance Errors View
On Error
Straight Through Processing
Manual Processing
Workflow Name: StraightThroughAuthorise
Case Worker Corrects Evidence issues

Figure 2. Application intake process diagram: Part 1

A citizen applies for assistance
As the citizen completes the dynamic application questionnaire, the data is stored in the data store.

Qualified citizen data is verified with external systems
Data that is configured to be verified with external systems is compared with the specified external data sources and the data store updated where required.

The citizen submits the application and the intake process starts
When the user clicks submit, the ProcessIntakeApplication workflow starts. An application case is created and data from the data store is applied to the application case. If there are workflow process errors, you can see them in the Process Instance Errors view.

The straight-through processing workflow
If none of the data on the case requires manual processing, the case is routed to the StraightThroughtProcessing workflow. Straight-through applications are authorized automatically. If there are workflow process errors, you can see them in the Process Instance Errors view. On authorization, the deferred transaction EVIDENCE_SHARE_BULK is started and evidence is shared to the integrated case.

Manual processing

If any of the data on the case requires manual processing, the case is routed to the deferred transaction process APPLICATIONAUTHORIZATION. If there are deferred transaction process errors, you can see them in the Process Instance Errors view. Applications in the manual processing workflow are authorized by the case worker. On authorization, the deferred transaction EVIDENCE_SHARE_BULK is started and evidence is shared to the integrated case.

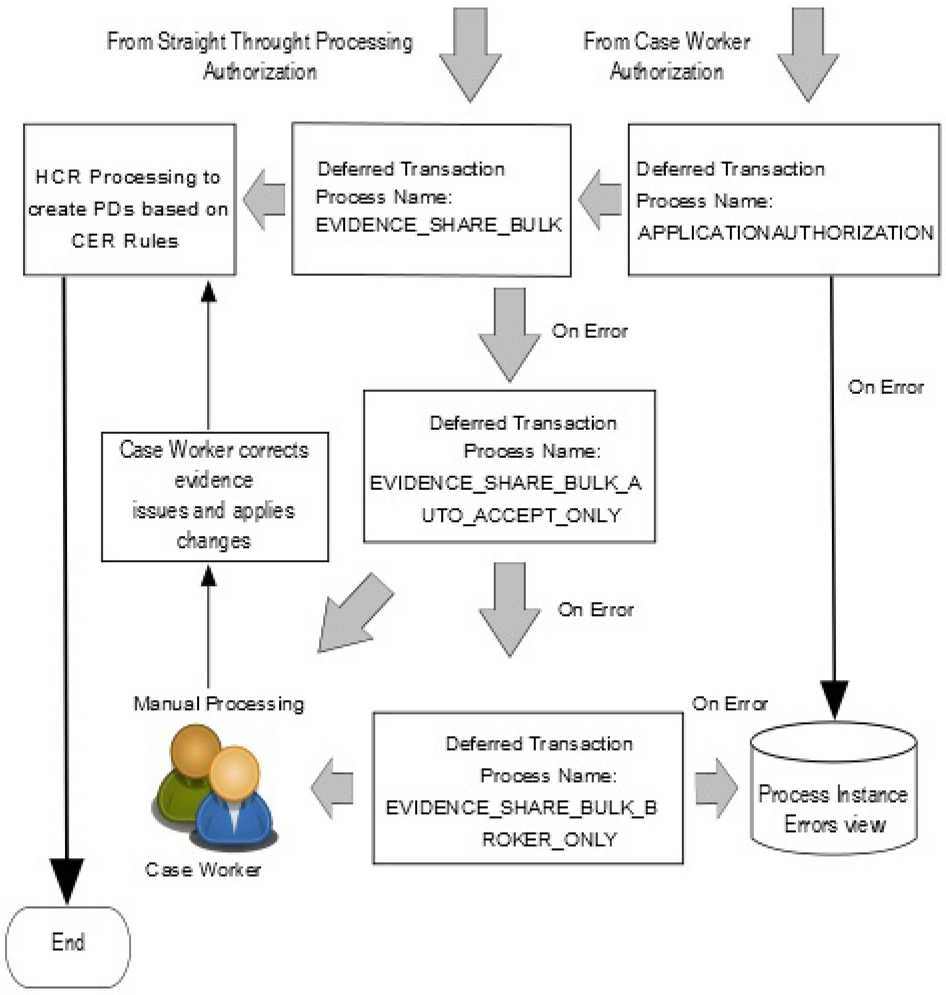
From Straight Throught Processing Authorization
From Case Worker Authorization
HCR Processing to create PDs based on CER Rules
Deferred Transaction Process Name: EVIDENCE_SHARE_BULK
Deferred Transaction Process Name: APPLICATIONAUTHORIZATION
On Error
On Error
Case Worker corrects evidence issues and applies changes
Deferred Transaction Process Name: EVIDENCE_SHARE_BULK_AUTO_ACCEPT_ONLY
On Error
Manual Processing
On Error
Deferred Transaction Process Name: EVIDENCE_SHARE_BULK_BROKER_ONLY
Process Instance Errors view
Case Worker
End

Figure 3. Application intake process diagram: Part 2

Where possible, evidence from the application case is brokered directly to the integrated case

On authorization, the EVIDENCE_SHARE_BULK deferred transaction process is started and evidence is shared to the integrated case. Program group logic processing is triggered automatically.

Where possible, after direct brokering has failed, evidence from the application case is added as 'In Edit' to the integrated case, manual intervention is then needed

If the evidence cannot be brokered directly, it is directed to the EVIDENCE_SHARE_BULK_AUTO_ACCEPT_ONLY deferred transaction process. A case worker accepts the in-edit evidence, corrects any issues, and applies the changes. Program group logic processing is triggered when the changes are applied.

Where possible, after brokering as 'In Edit' has failed, evidence from the application case is added as 'Incoming' to the integrated case, manual intervention is then needed

If the evidence cannot be brokered as 'In Edit' evidence, it is directed to the EVIDENCE_SHARE_BULK_BROKER_ONLY deferred transaction process. A case worker accepts the incoming evidence, corrects any issues, and applies the changes. Program group logic processing is triggered when the changes are applied. If there are deferred transaction process errors, you can see them in the Process Instance Errors view.

The program group logic runs on the integrated case and creates the required product delivery cases

Program group logic is triggered when evidence changes are applied to the integrated case and creates the required product delivery cases.

Program group logic is triggered automatically by the EVIDENCE_SHARE_BULK deferred transaction process, and each time a case worker applies evidence changes on the integrated case.

Note: The Health Care Reform program group logic to determine which potential multiple product delivery cases are to be created depends on predefined rule sets and therefore bypasses the Common Intake product delivery creation process. It does not configure the product delivery type for the program and therefore does not use the productDeliveryCaseID field on the programauthorisationdata table. For the Common Intake process, if a product delivery type is configured against a program, this product delivery type is created as part of a successful program authorization and recorded in the ProgramAuthorisationData entity.

Monitoring workflow process instances

Use the Process Instances view to see the status of each workflow process instance. By searching and filtering, you can see the current process instances and their status. Generally, the complete or in-progress processes are of most interest.

About this task

For troubleshooting, you have the following options:

- You can suspend a process instance that is in progress. You must resume the process instance before any further activities can run.
- You can stop a process instance that is in progress. Once aborted, a process instance cannot be resumed.
- All activities that wait for events to be raised have a failure mode where the event they are waiting on is raised before the activity runs. To progress such process instances, you can override the event wait.

Procedure

1. Log in as the admin user.
2. Select **Administration Workspace** > **Process Monitoring** > **Process Instances**
3. Use the search and filtering options to see the current workflow processes on the system.

Process Instance Errors

The Workflow Engine records information about errors that occur during the lifetime of a workflow process instance. You can use this information for troubleshooting problems with the process instance.

This troubleshooting includes retrying or aborting failed workflow process instances.

Retrying a failed process instance instructs the Workflow Engine to re-enact the workflow process instance from where it failed.

Aborting stops the process instance and its activities and closes any tasks that are associated with manual activities in the process instance. Depending on where the process was aborted, some manual steps might be required before the process is fully stopped.

Monitoring Process Instance Errors

Use the Process Instance Errors view to find workflow process or deferred process errors.

About this task

Plan to monitor the Process Instance Errors view regularly for potential operational errors or exceptions. You can abort or retry failed workflow process instances.

Procedure

1. Log in as the admin user.
2. Select **Administration Workspace** > **Process Monitoring** > **Process Instance Errors**
3. Use the search and filtering options to find process instance errors.
4. Click the error details for more information.

Chapter 7. Configuring Account Transfer to the Federally Facilitated Exchange

You can configure how Account Transfer applications are processed and sent to the Federally Facilitated Exchange.

This implementation uses the Cúram data store and the Cúram Persistence Infrastructure.

The FederalExchange component

The FederalExchange component helps state agencies to process Account Transfer applications that originate from the Federally Facilitated Exchange (FFE). In addition, the FederalExchange component sends applications that originate from the state agency to the FFE for applicants that are not eligible for Medicaid or CHIP.

Configuring Federal Exchange

Configure Account Transfer to the Federal Exchange to your requirements by modifying the appropriate properties.

About this task

For information about configuring properties, see "Configuring Application Properties" in the *Cúram System Configuration Guide*.

Activating Account Transfer

Account Transfer is switched off by default. When you activate Account Transfer, eligibility determinations are processed and prepared to be sent to the FFE by the FederalExchange component.

Procedure

To activate Account Transfer, set the `curam.healthcare.account.transfer.activate.outbound.mapping` property to `true`.

Enabling batch processing of account transfer applications

If you want to process Account Transfer applications by batch processing, you can modify a property to stop the account transfer application data from being sent to Cúram for inbound applications and to the FFE for outbound applications.

About this task

Enabling batch processing prevents the mapping of the data and the processing of that mapped data by Cúram or the FFE. The `FederalExchangeApplication` entity stores the jobs that are pending for each batch process.

Procedure

To process Account Transfer applications by batch processing, change the value of the `curam.healthcare.account.transfer.processing.mode` property from the default value of `online` to `batch`.

Configuring the sending of Account Transfers to Cúram

You might want to complete the mappings in real time and to stop the processing just before the Account Transfer is sent to Cúram for case processing.

About this task

If you stop processing applications, the applications remain in a PENDING state on the `FederalExchangeApplication` entity.

For other possible states for entries on this entity, see the HCRFedExchangeAppStatus code table.

Procedure

To ensure that no Account Transfer applications are sent to Cúram, update the `curam.healthcare.account.transfer.auto.submit` property from `true` to `false`.

Selecting the source data set for outbound mapping

Outbound mapping can be completed from two different sets of source data, intake and case processing. Different data store schemas are used to store the data in each case.

About this task

The following sets of source data are available:

Intake The data that is stored in the data store as a result of a case worker completing the internal case worker intake application for Health Care Reform.

Case processing
The data that is stored in the data store as a result of running an implementation of HCRDatastoreBuilder to convert case and person evidence for a Health Care Reform application to data store data.

Procedure

1. Set the `curam.healthcare.account.transfer.outbound.mapping.source` property to `intake` for the internal case worker intake application for Health Care Reform, or `caseprocessing` for the data store data that is derived from case and person evidence.
2. Set the `curam.healthcare.account.transfer.internal.datastore.schema` property to the correct schema name depending on the source of the data.

Setting the identity of the sender US state

Ensure that the sender is identified correctly by setting the correct US state. The codes that are used to denote the sender state are stored as properties.

About this task

Procedure

Update the following properties for the US state for which HCR is implemented:

```
curam.healthcare.account.transfer.sender.state.code
curam.healthcare.account.transfer.sender.county
curam.healthcare.account.transfer.sender.category.code
```

Setting the data store schema name for the FFE schema

You can set the schema name for the data store schema representation of the FFE schema.

Procedure

Set the schema name for the data store schema representation of the FFE schema in the `curam.healthcare.fedexchange.version.schema` property.

Extending Federally Facilitated Exchange data mappings

You can modify the default Federally Facilitated Exchange data mappings to add attributes or entities to the data that is sent or received.

Federally Facilitated Exchange (FFE) mappings are called when data is received from the FFE (inbound) or sent to the FFE (outbound).

When you receive data from the FFE, you must map data from the FFE data schema to the Cúram data store schema so that Cúram can process that data.

When you send data to the FFE, you must map data from the Cúram data store schema to the FFE data schema so that the FFE can process that data.

Adding or updating the attributes for a data store entity

Modify the properties of the appropriate Persistence Infrastructure event to add or update an attribute.

About this task

After an entity is mapped by the FederalExchange component, a Persistence Infrastructure event is sent to allow custom listeners of the event to add or update the attributes on the entity.

As with all data store processing, the attributes that are added to an entity must conform to the data store schema that is configured for that instance of data store data. For information about the event signature and more information on usage, see the Javadoc.

Procedure

To add or update an attribute, configure the appropriate event for the custom listeners:

- Inbound
  `curam.hcr.fedexchange.mapper.impl.EntityMapper.MapEvent.customInboundMap`
- Outbound

```
curam.hcr.fedexchange.mapper.impl.EntityMapper.MapEvent.customOutboundMap
```

Example listener:

```
/**
  * Raised when in-bound mapping (from FFM to Curam) has been completed
  * on a given
  * data store entity.
  *
  * @param mappedEntity
  *         The data store entity that contains mapped data.
  *         Note that if the entity is a child it will not have
  *         been added to its parent at this point and will therefore
  *         not have a unique identifier.  The result of this is that
  *         no children can be added to this entity during the
  *           processing
  *         of this method.
  *  @param originalElement
  *         The original data that can act as the source for the
  *         mapped data.
  *
  * @throws AppException
  *          Generic Exception Signature.
  * @throws InformationalException
  *          Generic Exception Signature.
  */
public void customInboundMap(Entity mappedEntity,
     Element originalElement)
                          throws AppException, InformationalException{}
```

Adding an entity as a child of a mapped data store entity

When an entity is mapped and you add child entities to the entity, an event is sent that allows custom listeners to add extra child entities to the entity.

About this task

Any child entity types that are added must exist in the data store schema for the data store data that is being processed.

Procedure

To add a child entity, configure the appropriate event for the custom listeners:

- Inbound

  ```
  curam.hcr.fedexchange.mapper.impl.EntityMapper.MapEvent.
  ```
- Outbound

  ```
  curam.hcr.fedexchange.mapper.impl.EntityMapper.MapEvent.
  customOutboundMapChildren
  ```

Example listener:

```
/**
* Raised when in-bound mapping (from FFM to Curam) has been completed * on a given
  * data store entity and when children can be added to that entity.
  * Custom processing should check for an existing child before
  * creating one.
  *
  * @param mappedEntity
  *         The data store parent entity that contains mapped data.
  *         This entity can be used to create valid child entities
  *         underneath.
  * @param originalElement
  *         The original data that can act as the source for the
  *         mapped child data.  It is possible to traverse up or down
  *         the DOM tree using the originalElement as the starting
```

```
 *         point
 *
 * @throws AppException
 *          Generic Exception Signature.
 * @throws InformationalException
 *          Generic Exception Signature.
 */
 public void customInboundMapChildren(Entity mappedEntity,
  Element originalElement)
                        throws AppException, InformationalException{}
```

Adding or replacing a top-level data store entity

An element mapping provider can map implementations for a target entity type. You can use this element mapping provider to add custom processing to create an entity that is at a top level.

About this task

Typically, a child of the Application root entity or another entity type that can have only a single instance.

Procedure

Use the appropriate events to add a custom mapping implementation for the element mapping provider:

- Inbound

  ```
  curam.hcr.fedexchange.mapper.impl.ElementMapperEvent.
  addElementMapperEvent(Map elementMappers)
  ```

  The elementMappers contains a map of entity types and mapping implementations to which you can append extra entity types and custom mapping implementations that are called as part of the element mapping provider processing.
- Outbound

  ```
  curam.hcr.fedexchange.mapper.ffe.impl.FFEElementMapperEvent.
  addFFEElementMapperEvent(Map elementMappers)
  ```

  A custom listener to this event is implemented in much the same way as for the listener for the inbound provider event. The target entity type that is being added to the map is an entity type in the Federal Exchange data store schema. The mapping implementation maps data from the Cúram data store schema to the Federal Exchange data store schema.

Example listener:

```
/**
* Raised when {@linkplain ElementMapperProvider} is initialized to
* allow
 * additional EntityMapper to be included.
 *
 * @param elementMappers
 *          The map containing a string that represents the
 *          element being mapped from and the mapping implementation
 *          that creates and maps to the corresponding element on the
 *          target schema.
 *
 * @throws AppException
 *          Generic Exception Signature.
 * @throws InformationalException
 *          Generic Exception Signature.
```

```
*/
public void addElementMapperEvent(Map<String,
                     Provider<? extends EntityMapper>> elementMappers)
     throws AppException, InformationalException;
```

Adding or updating entities for an outbound response to the FFE

When the response to be sent to the FFE is built by the FederalExchange component, an event is sent with all of the response data. Listeners to this event can then update the response data before they send it to the FFE.

Procedure

To add or update outbound entities, configure the following event:

```
curam.hcr.fedexchange.mapper.impl.EntityMapper.
MapEvent.customOutboundMapResponse
```

Example listener:

```
/**
  * Raised when out-bound mapping (from Curam to FFM) has been completed on the
    * response being
  * sent to the FFM.
  * Custom processing should check for an existing response entity before
  * creating one.
  *
  * @param mappedEntity
  *          The data store parent entity that contains mapped data.
  *          This entity for an applicant can be used to create valid response
  *           entities.
  * @param originalElement
  *          The original data that can act as the source for the
  *          mapped response data.
  *
  * @throws AppException
  *           Generic Exception Signature.
  * @throws InformationalException
  *           Generic Exception Signature.
  */
  public void customOutboundMapResponse(Entity mappedEntity,
                                      Element originalElement)
                       throws AppException, InformationalException{}
```

The Web Service Java API

You can use the Federal Exchange component Java API for data that is received from or sent to the Account Transfer web service, or to send data to the FFE.

Inbound processing

You can use the Java API as entry and exit points for data that is received from or sent to the Account Transfer web service.

You use the `curam.hcr.fedexchange.ws.impl.AccountTransferWS.initiateAccountTransfer` method to send an account transfer to Cúram from the FFE or to send an account transfer response to Cúram from the FFE.

Outbound processing

To send data to the FFE, the API includes events that provide the data to be sent.

```
curam.hcr.fedexchange.ws.impl.AccountTransferWS.
OutBoundDataEvent.sendOutBoundTransferDataEvent
```

This property sends an account transfer application from Cúram. The listener that receives this event can alter the data to meet specific custom needs (if not already catered for by the custom mapping processing) and then send that data to the FFE by a web service. Any mapping updates that must be made by custom processing must use the events that are sent during data mapping.

```
curam.hcr.fedexchange.ws.impl.AccountTransferResponseWS.
OutBoundResponseEvent.sendOutBoundResponseEvent
```

This property sends a response of an account transfer application from Cúram. The listener that receives this event can alter the data to meet their specific needs and then send the data to the FFE through a web service. Any mapping updates that must be made by custom processing must use the events that are sent during outbound response processing.

HCRFedExchangeAppStatus code table descriptions

A list of the possible status states of the FederalExchangeApplication that uses the HCRFedExchangeAppStatus code table. For clarity, the status states are divided by the direction of the request. The status states are listed in the same sequence in which the transitions happen.

Table 32. Account Transfer from FFM to State Medicaid Agency

Code	Java Identifier	Full Description
HCRIFEIP	IBD_IN_PROGRESS	The initial status on creation of a Federal Exchange Application. On creation, the record contains the root data store entity ID of the external data store that is used to store the Account Transfer payload from the Federally-Facilitated Marketplace (FFM).
HCRIFEUD	IBD_UPDATE_PENDING	This state is the other initial state that is possible for an Account Transfer received by the State. The Federal Exchange Application is created in this state if there is an existing Account Transfer with the same Global Application ID. In other words, this transfer is considered as a change of circumstances. Currently, the default Account Transfer feature does not address change of circumstances payloads. Instead, the FederalExchangeApplication is saved and an event is raised that can be handled by custom listeners to provide the required processing.
HCRIRSAK	IBD_ACKNOWLEDGED	Set after an inbound request is stored and successfully acknowledged.
HCRIFEER	IBD_ERROR	Set when any issues are encountered during the mapping of the FFM payload to the internal data store, or when the FFM payload is stored in the external data store.
HCRORSIP	OBD_RESPONSE_IN_PROGRESS	Set when the processing for sending a response to the FFM is initiated.
HCRORSAK	OBD_RESPONSE_ACKNOWLEDGED	Set after the response sent by the State Medicaid Agency is acknowledged successfully by the Federally-Facilitated Exchange (FFE).
HCRIFEPD	IBD_PENDING	If transfers are configured to happen in batch mode. The Account Transfer payload is stored in the external data store but no further processing happens as part of the online processing.

Table 33. Account Transfer from State Medicaid agency to FFM

Code	Java Identifier	Full Description
HCROFEIP	OBD_IN_PROGRESS	Set on a new instance of FederalExchangeApplication that is created for a transfer from the State to the FFM.
HCROFEPD	OBD_PENDING	Set if transfers are configured to happen in batch mode. No further processing is done for this transfer as part of online processing.
HCROFEAK	OBD_ACKNOWLEDGED	Set after a transfer from the State is acknowledged successfully by the FFE.

Table 33. Account Transfer from State Medicaid agency to FFM (continued)

Code	Java Identifier	Full Description
HCROFEER	OBD_ERROR	Set if an acknowledgement to an outbound transfer was not received or is not successful. Also set if there are any issues during mapping from the HCR data store to the FFM data store. If there were errors during mapping, Federal Exchange Applications are not transferred.
HCRIRSIP	IBD_RESPONSE_IN_PROGRESS	Set on receiving the response from the FFM for an Account Transfer from the State.
HCRIFEAK	IBD_RESPONSE_ACKNOWLEDGED	Set when the response from the FFM for an Account Transfer from the State is successfully acknowledged
HCRIRAER	IBD_RESPONSE_ACKNOWLEDGE _ERROR	Set if any issues were encountered when the response to an outbound account transfer is stored, or if there were issues with the generation of an acknowledgement.

Adding a new entity

You can add a new entity to replace an existing entity or to create an entity that is not mapped and created by default. For each new entity, write an entity mapper and add the new entity to the Federal Exchange data store schema.

Writing an EntityMapper

You must write an EntityMapper for each new entity. An EntityMapper must implement the `curam.hcr.fedexchange.mapper.impl.EntityMapper` interface.

About this task

An implementation of `curam.hcr.fedexchange.mapper.impl.ElementMapperUtil` is provided in the map method to facilitate searching for required elements and attributes in the source XML to be used to populate the entity or entities that are being created.

Procedure

1. Using the provided example, implement an EntityMapper.
2. After you implement the EntityMapper, register it by using the ElementMapperEvent inbound or outbound event as appropriate. This depends whether the Mapper implementation is being called for inbound or outbound processing.

Example

This example outlines how IncomeItem entities might be mapped from the FederalExchange external system into Cúram and added to the data store.

```
/**
 * Sample entity mapping implementation that creates a new
 * data store entity and appends it to a parent entity.
 */
public class SampleEntityMapperImpl implements EntityMapper {

  /** The source element to map from, the source elements of interest can be
   * searched for by using this element **/
  private Element source;
  /** The FederalExchangeApplication persistence infrastructure implementation
   *  for the FederalExchangeApplication entity **/
  private FederalExchangeApplication federalExchangeApplication;

  @Override
  public void setSource(Element source) {
    this.source = source;
  }

  @Override
  public void map(Entity parent, ElementMapperUtil elementMapperUtil) {
    Datastore ds = parent.getDatastore();
    //get the element from the source i.e. the element from the FederalExchange
    //XML
    List<Element> incomeItems =
      elementMapperUtil.getElements(FFEEntityType.PERSONINCOME.entityType(),
        source);

    for(Element incomeItemSource : incomeItems){
      //create the new entity in the target data store
      Entity incomeItem = ds.newEntity(EntityType.INCOMEITEM.entityType());
      //set the attributes on the new target entity
      incomeItem.setTypedAttribute(IncomeItemFieldMap.STARTDATE.hcrField(),
        FieldMapperUtil.formatDate(
          elementMapperUtil.getAttribute(incomeItemSource,
           elementMapperUtil.createFindAttributeQuery(
             IncomeItemFieldMap.STARTDATE.ffeField()))));
      incomeItem.setTypedAttribute(IncomeItemFieldMap.ENDDATE.hcrField(),
        FieldMapperUtil.formatDate(
          elementMapperUtil.getAttribute(incomeItemSource,
              elementMapperUtil.createFindAttributeQuery(
                IncomeItemFieldMap.ENDDATE.ffeField()))));

      incomeItem.setTypedAttribute(IncomeItemFieldMap.INCOMEAMOUNT.hcrField(),
        elementMapperUtil.getAttribute(incomeItemSource,
          elementMapperUtil.createFindAttributeQuery(
            IncomeItemFieldMap.INCOMEAMOUNT.ffeField())));
      //add the new entity as a child of the parent entity
      parent.addChildEntity(incomeItem);
    }
  }

  @Override
  public void postMap(Entity rootEntity, Entity personEntity) {
    //no post map processing required
  }

  @Override
  public void setFederalExchangeApplication(
      FederalExchangeApplication federalExchangeApplication) {
    this.federalExchangeApplication = federalExchangeApplication;
  }
}
```

Updating the Federal Exchange data store schema

If a new entity is being added by custom processing, then you must update the data store schema that is used to store the entity for inbound and outbound mapping.

Before you begin

It is important to note the following when you update the Federal Exchange data store schema for Account Transfer.

- If element text exists in the payload from the Federal Exchange, then this text is converted into an attribute. This allows the text to be stored in the data store. For example:

  ```
  <IncomeAmount>1200</IncomeAmount>
  ```

You define that Federal Exchange payload XML in the data store schema as follows:

```
<xsd:element name="IncomeAmount">
       <xsd:complexType>
            <xsd:attribute name="value" type="d:SVR_STRING"/>
        </xsd:complexType>
</xsd:element>
```

Note the use of the attribute `value` to store the element text.

- If the element in the Federal Exchange payload contains a name space prefix, then the data store schema must contain an attribute that defines the name space prefix value as the default value. For example:

```
<hix-core:IncomeAmount>1200</hix-core:IncomeAmount>
```

You define that Federal Exchange payload XML in the data store schema as follows:

```
<xsd:element name="IncomeAmount">
       <xsd:complexType>
            <xsd:attribute name="value" type="d:SVR_STRING"/>
            <xsd:attribute name="nameSpacePrefix" type="d:SVR_STRING"
   default="hix-core:"/>
       </xsd:complexType>
</xsd:element>
```

Procedure

1. Identify the relevant data store schema. The name of the data store schema name that stores the Federal Exchange data for Account Transfer is denoted by the `curam.healthcare.fedexchange.version.schema` property.
2. Update the data store schema with the new entity.

Notices

This information was developed for products and services offered in the U.S.A. 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

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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.

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

Printed in the Republic of Ireland