IBM Cúram Social Program Management

Health Care Reform Developer Guide

Version 6 Release 0

IBM Cúram Social Program Management

Health Care Reform Developer Guide

Version 6 Release 0

Note

Before using this information and the product it supports, read the information in “Notices” on page 27

Revised: May 2013

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

Contents

Chapter 1. About this information

The Health Care Reform Developer Guide describes how to develop applications for the IBM Cúram Solution for Health Care Reform.

1.1 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.

1.2 Intended audience

This publication is intended for developers who are developing applications on 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 2. Customizing the Health Care Reform Portal

The Health Care Reform Portal utilizes the Motivation framework provided by Universal Access in order to configure the various online application processes required by ACA legislation.

Each Health Care Reform motivation, such as 'Enroll in a QHP with Assistance' or 'Apply for Exemption' is associated with an IEG script, a datastore schema and a display rule set. Please see the Curam Universal Access Customization Guide for more in-depth details on motivations.

2.1 IEG scripts customization

The Health Care Reform portal scripts can be found in the HCROnline component. The OOTB IEG Scripts can be customized by creating a custom copy. For more information, see the guidelines that are provided in the *Authoring Scripts Using Intelligent Evidence Gathering* guide.

2.2 Rules customization

When the IEG script has completed, the eligibility results page is displayed. This page results in the execution of display rules in order to display eligibility result information. Custom display rules can be written in order to customize eligibility calculations for this page. Please refer to the Cúram Development Compliancy Guide for details on how to compliantly customize rule sets. In addition, Health Care Reform provide several other mechanisms for customizing rule sets. These are listed below.

The OOTB display rules call out to the OOTB eligibility rule sets in order to determine eligibility. The environment property named curam.healthcare.eligibility.ruleset.name points to the name of this rule set. This property must be updated if custom eligibility rules are to be used.

There are several areas in the script rules where it is possible to provide a custom implementation. These areas are listed below.

Conditional display of IRS Income information

IRS income data that is retrieved for members in a tax household will not be 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/CHIP threshold for any of the applicants in the household

These rules are implemented by the IRSIncomeDisplayDeterminator rule class available in the OOTB rule set named HealthCareReformEligibilityRuleset.

This class can be overridden with a custom rule class by setting the curam.healthcare.displayirsincome.invoking.ruleclass.name environment property to the name of the custom rule class. The custom rule class must ultimately extend the AbstractEligibilityRuleset.DefaultIRSIncomeDisplayDeterminator rule class.

Conditional display of Medicaid/CHIP/IA specific questions

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

OOTB, the HealthCareReformEligibilityRuleset.EligibilityDeterminationCalculator rule class is used to carry out this processing. This class can be overridden with a custom rule class by setting the curam.healthcare.eligibility.invoking.ruleclass.name environment property to the name of the custom rule class. The custom rule class must ultimately extend the AbstractEligibilityRuleset.DefaultEligibilityDetermination rule class.

Determine projected annual income for a citizen

Income calculation rules are run after capturing the a household member's complete income details (including any deductions or exclusions). The projected annual income is then calculated by rules based on these details. The citizen can choose to attest to the determined projected annual income or chose to enter a different value. If a different value is entered than what was determined by rules, rules would take this value into consideration when calculating final eligibility.

OOTB, the HealthCareReformEligibilityRuleset.MemberIncomeCalculator rule class is used to carry out this processing. This class can be overridden with a custom rule class by setting the curam.healthcare.memberincome.invoking.ruleclass.name environment property to the name of the custom rule class. The custom rule class must ultimately extend the AbstractEligibilityRuleset.DefaultMemberIncomeCalculator rule class.

2.3 Life events customization

Health Care Reform life events are modeled as simple life events and Universal Access provides hook points for the configuration of these life events as normal.

This configuration is achieved by configuring LifeEventType, LifeEventContext, LifeEventCategory, and LifeEventCategoryLink entities or by using the Administrator's User Interface.

For more information about configuring Simple Life Events using the Administrator's UI, see "Configuring Life Events" in the *Cúram Universal Access Configuration Guide*.

Java code is used instead of mappings to implement the complicated logic of taking changes that are reported in the life event script and turning into evidence changes that can be reflected on the Insurance Affordability integrated case.

2.3.1 OOTB life events processing

An overview of the OOTB life events processing

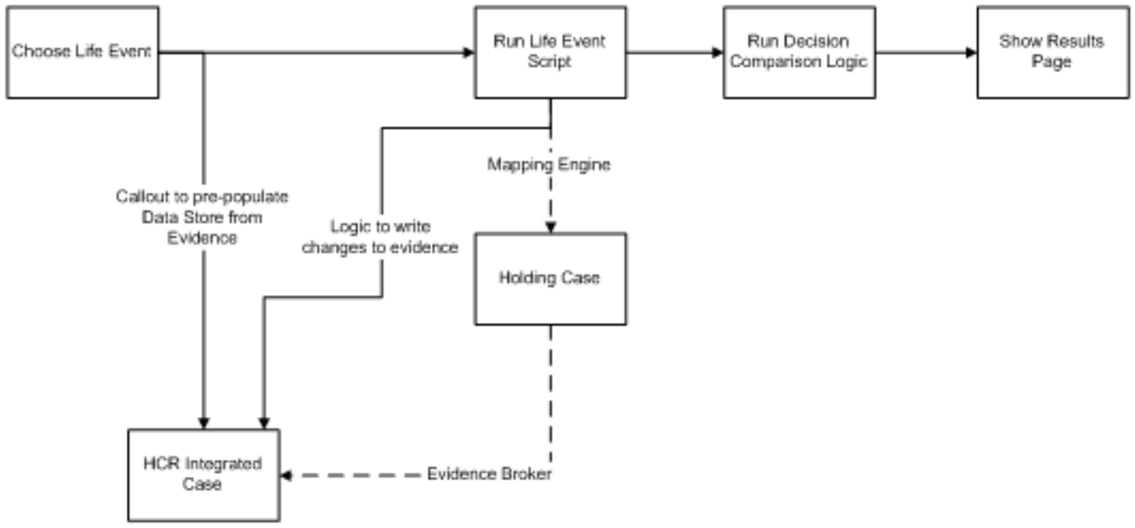

Figure 1. OOTB life events processing

- On launch of the IEG Script that is configured for the life event, the pre-population hook that is configured for this life event is called to populate the data store. This implementation should adhere to “curam.citizen.lifeeventbroker.impl.DatastoreBuilder”. The implementation in Health Care Reform pre-populates information from Active Evidence records on the Health Care Reform Integrated Case, about citizen's household, existing jobs, existing ESC, and other information relevant to the life event. Few more information about the citizen's household are also populated in order to determine citizen's new Eligibility and entitlement determination as a result of the changes reported. This implementation is common for all the OOTB Health Care Reform life events. Default Health Care Reform Implementation is ‘curam.healthcare.lifeevents.impl.HealthCareDatastoreBuilderImpl’).
- In order to take the changes that are made to the Data Store and write them back to the IC as In-edit evidence changes, Health Care Reform has implemented a life event submission hook point that is given by Universal Access (curam.citizen.lifeeventbroker.impl.LifeEventUpdater). This implementation takes the changes reported in the life event script and turns them into evidence changes that can be reflected on the Insurance Affordability integrated case. (Default Health Care Reform Implementation name is ‘curam.healthcare.lifeevents.impl.HealthCareLifeEventUpdaterImpl’). The evidence changes here can be categorized into the following,
- Creation of new Evidence – This is done when new jobs, new employer sponsored coverage, or new additional incomes are reported through the life events.
- Correction of existing Evidence – This is done when existing jobs, employer sponsored coverage, or additional incomes are end dated and thus are no longer relevant.
- Change of circumstances – This is done when there is a change in circumstance such as increase/decrease in income.

Pre population of data store and submission hook points are bound by a Life Event Type Name to these default Health Care Reform implementations. These implementations can be extended by custom implementations, if needed.

2.3.2 Overriding OOTB life events processing

Callout to pre-populate the data store from Evidence or the logic to write changes back to IC can be overridden as follows.

2.3.3 Adding custom life event data store pre-population processing

Complete theses steps to add custom life event data store pre-population

Procedure

1. Create an implementation of the curam.citizen.lifeeventbroker.impl.DatastoreBuilder for custom life event data store pre population processing.
2. Install the custom implementation using a custom Guice module. An entry containing the custom Guice module name should be added to a dmx file for ModuleClassName entity. The custom life event data store pre population processing implementation should be bound using a Guice MapBinder. For example:

```
public class CustomModule extends AbstractModule {
  @Override
  protected void configure() {
    Mapbinder<String, DatastoreBuilder > binder = Mapbinder.newMapBinder(
       binder(), String.class, DatastoreBuilder.class );
      binder.addBinding().to(CustomDataStoreBuilder.class);
  }
}
```

2.3.4 Adding custom life event evidence updater processing

Complete theses steps to add custom life event evidence updater processing.

Procedure

1. Create an implementation of the curam.citizen.lifeeventbroker.impl. LifeEventUpdater for custom life event evidence updater processing.
2. Install the custom implementation using a custom Guice module. An entry containing the custom Guice module name should be added to a dmx file for ModuleClassName entity. The custom life event data store pre population processing implementation should be bound using a Guice MapBinder. For example:

```
public class CustomModule extends AbstractModule {
  @Override
  protected void configure() {
    Mapbinder<String, LifeEventUpdater > binder = Mapbinder.newMapBinder(
       binder(), String.class, LifeEventUpdater.class );
      binder.addBinding().to(CustomLifeEventUpdater.class);
  }
}
```

Chapter 3. Integration with external systems

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

The customization and configuration options for these integration points are listed in the following sections.

3.1 Customizing the external system implementations

The external system interfaces, default implementations and federal hub implementations are listed below.

About this task

Note: The OOTB Federal Hub implementation may need to be customized. Customization points have been added to facilitate this.

About this task

OOTB, 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. The table below lists the OOTB interfaces and their 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. RIDPPrimaryRequestVerificationServiceImpl	curam.hcr.verification.service.ridp. primary.impl. FederalRIDPPrimaryRequestServiceImpl
curam.hcr.verification.service.ridp. secondary.impl. RIDPSecondaryRequestVerificationService	curam.hcr.verification.service.ridp. secondary.impl. RIDPSecondaryRequestVerificationServiceImpl	curam.hcr.verification.service.ridp. secondary.impl. FederalRIDPSecondaryRequestServiceImpl

Procedure

1. To create a custom implementation, write a new class which extends one of the external system default implementations listed above.
2. The custom implementation must be bound to the corresponding interface, using a Guice module.
   ```
   public class CustomModule extends AbstractModule {
   @Override
   protected void configure() {
       binder().bind(IncomeDataService.class).to(CustomIncomeDataService.class);
     }

   }
   ```
3. Ensure the module has been 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>
   ```

3.2 Customizing request/response fields for external system calls

It is possible to customize the request and response fields that are used by the external system interfaces.

About this task

This can be achieved by extending the respective request or response classes. The updated request/response classes can then be used in the custom implementation of the external system interface.

Procedure

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
}
CustomCitizenshipVerificationServiceImpl implements CitizenshipVerificationService {
      CustomCitizenshipVerificationResponseDetails
verify(CutomCitizenshipVerificationRequestDetails requestDetails);
}
```

3.3 External System Processors

During the application process, external system processors call out to external systems and store the information received in the datastore. For example, a call is made to the federal hub to verify SSN and citizenship using the CombinedSSAServiceViewProcessor processor. The response is stored in the datastore by the processor and can be used later to facilitate the electronic verification process. The following processors exist:

- `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`

3.4 Federal Hub Configuration

OOTB, all external system calls will be 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. A server restart will be required when these values are updated.

Procedure

1. The property, “curam.healthcare.test.registerMockExternalSystems”, should be set to "false".
2. The property, “curam.fed.hub.verification.system.name” should be specified with the federal hub system name.
3. The property, “curam.fed.hub.verification.system.registered” should be set to true.

3.5 State systems implementation

A custom implementation might need to call State systems as well as calling the Federal Hub.

For example, current Income might need to be retrieved from the State Quarterly Wages system and only if the information is not available fallback on the corresponding Federal Hub Service. This can be achieved by providing a custom implementation for the Current Income service. This custom implementation can call the State system followed by a call to the OOTB Federal Hub implementation.

3.6 Customizing electronic verifications

External systems are also used for electronic verification of information provided in the application. The intention is for Health Care Reform to provide support for integrating with external systems (state systems or third party commercial applications identified by states as data sources) and also to enable customization of electronic verification.

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

3.6.1 Adding custom verification processing

Complete the following steps to add custom verification processing.

Procedure

1. Add an entry in to the "VerificationItemType" codetable (CT_VerificationItemType.ctx)
2. Create an implementation of the curam.hcr.verification.online.impl.VerificationProcessorinterface. The "getVerificationType()" API should return the codetable code added in the first step
3. Install the custom implementation using a custom Guice module. An entry containing the custom Guice module name should be added to a dmx file for ModuleClassName entity. The custom Verification Processing implementation should be bound using a Guice MultiBinder (Set). See below, for an example.

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

3.6.2 OOTB verification processing overrides

To override an OOTB Verification processing, the steps for adding a custom processor should be followed except for a couple of differences.

Each entry in the "VerificationItemType" represents a kind of data item (example, Citizenship). Instead of creating a new code, the code for the data item type for which the OOTB processing should be overridden should be identified. The custom verification implementation should return this code in its implementation of the "getVerificationType()" API.

3.6.3 OOTB verification processors

The following Verification Processors are shipped OOTB.

- curam.hcr.verification.online.impl.ResidencyVerificationProcessor – Considers the Residency to be verified if it was indicated (isStateResident attribute of the Person datastore 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 retrieved about the person from external system (stored in the ExternalSystemResidencyInformation datastore entity) also indicates that the person is a state resident. This processing is performed for all the persons who are marked as applicant (isApplicant attribute of the Person datastore 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 datastore entity) that a Person was a US citizen or US Nation or Lawfully Present alien and the information retrieved about the person from external system (stored in the ExternalSystemCitizenshipInformation datastore entity) also indicates that the person citizenship verified. This processing is performed for all the persons who are marked as applicant (isApplicant attribute of the Person datastore 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 datastore entity) that a Person is incarcerated (or) If the Person was indicated to be not incarcerated or incarcerated pending disposition and the information retrieved about the person from external system (stored in the RetrievedPersonInformation datastore entity) also indicates the same. This processing is performed for all the persons who are marked as applicant (isApplicant attribute of the Person datastore 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 datastore entity) and the information

retrieved about the person from the external system (stored in the ExternalSystemSSNInformation datastore entity) also indicates the given SSN was verified. This processing is performed for all the persons who are marked as applicant (isApplicant attribute of the Person datastore entity) on the case.

- curam.hcr.verification.online.impl.IncomeVerificationProcessor – Considers the Income data to be verified if the Income was provided(IncomeItem datastore entity has records) and the information retrieved about the person from the external system (store in the IRSAnnualTaxReturn or ExternalSystemIncome datastore entity) are reasonably compatible/E verified. This processing is performed 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 datastore entity) and the information retrieved about the person from the external system (stored in the ExternalSystemMECDetails datastore entity) also indicates the same. This process is performed for all the persons.

Chapter 4. Customizing case management

The sections below deal with customization of Health Care Reform case management artifacts. For example, eligibility rule sets, verifications and dynamic evidence.

4.1 Dynamic evidence customization

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

For information about customizing dynamic evidence, please refer to the *Cúram Dynamic Evidence Configuration Guide*.

4.2 Rules customization

HCR ship with a set of eligibility rule sets located in the HCR component. Please refer to the *Cúram Development Compliancy Guide* for details on how to compliantly customize these rule sets.

See also the *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules guide*.

4.3 Conditional verifications customization

Health Care Reform application cases and integrated cases are configured to use the verification framework.

For more information about configuration of verifications and conditional verifications, see the *Cúram Verification Guide*. Conditional verification rule sets can be customized in the same manner as other rule sets. Please refer to the *Cúram Development Compliancy Guide* for details on how to compliantly customize these rule sets.

Chapter 5. Customizing plan management

5.1 Overview of the plan management implementation

When a citizen applies for insurance affordability through Cúram, they are required to visit a plan management vendor's website in order to view and purchase plans. In order to facilitate this, a plan management vendor must be integrated with the Cúram application.

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

A plan management vendor's user interface is displayed within a HTML iFrame on a Cúram page.

Information is exchanged between Cúram and the plan management vendor using web services. The web services used fall into the following categories:-
- Cúram owned web services (inbound)
- Web services owned by the plan management vendor (outbound)

This approach allows the citizen to enroll on a plan on the plan management vendor's screens using the eligibility information 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 has enrolled.

5.2 The plan management adapter interface

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

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

`curam.planmanagement.adapter.impl.PlanManagementAdapter`
- `getBenchmarkPlanAmount()`
- `getEnrollmentDetails()`
- `getAvailableEmployerPlanDetails()`
- `getBenchmarkPlanAmountForBenefitMembers()`
- `updateEntitlementDetails()`
- `getPlanUpdates()`
- `continueEnrollment()`
- `getPolicyID()`
- `getEmployerOpenEnrollmentDetails()`

Note: A default implementation of the plan management adapter interface is provided OOTB. Customers should extend this class instead of directly implementing the interface. This will provide some insulation from future changes. The default implementation is named curam.planmanagement.adapter.impl.PlanManagementAdapterDefault

Note: For more details, please review the JavaDoc for the plan management adapter interface.

`getBenchmarkPlanAmount()`

Retrieve the benchmark plan amount from a plan management vendor.

`getEnrollmentDetails()`

Retrieve the enrollment details for a completed enrollment e.g. the enrolled plan details.

`getAvailableEmployerPlanDetails()`

Retrieve the available employer insurance plans for an employee.

`getBenchmarkPlanAmountForBenefitMembers()`

Retrieve the benchmark plan amount from a plan management vendor.

`updateEntitlementDetails()`

This is called to inform the plan management vendor that a change in entitlement has occurred for a specific enrollment.

`getPlanUpdates()`

Retrieves any updates to plans for an enrollment. This is typically called during re-enrollment.

`continueEnrollment()`

Informs the plan management vendor that an existing enrollment on a plan is to be continued. This is 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.

5.2.1 Configuring the plan management adapter

The custom plan management adapter typically communicates with a plan management vendor over a web service using stubs that have been generated from the plan management vendor's wsdl file. To get to that point, the following steps are required.

Procedure

1. Create a directory named axis in a custom component
2. Add a ws_outbound.xml file to this directory. This file should reference the wsdl file 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 EJBServer, run ‘build wsconnector2’ – this will generate the stubs to the build directory. These stubs will then be 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 using the stubs that are were previously generated.

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

5.3 Plan management web services provided by Cúram

A plan management vendor will need to call Cúram web services in order 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 will require 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 details on these web services, please review the associated JavaDoc. Also, see the Health Care Reform plan management web service API reference within this document.

5.4 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 will be 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 will be 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 will be appended to this URL.
curam.healthcare.planManagementVendorAvailable	This property indicates whether or not a plan management vendor is available. By default, it is set to false to enable testing but must be set to true once integrated with a plan management vendor.

5.5 Call back URLs for plan management

Call back URLs are the URLs that a plan management vendor will use to return control back to the Cúram user interface. For example, after an enrollment has been completed, a call back URL is used to redirect back to the Cúram results page.

The following call back URLs exist OOTB:

Call back 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 at a later date.
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.

5.6 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*.

5.6.1 Employer enrollment notification batch process

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

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

Appendix. Health Care Reform 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.

A.1 Health Care Reform web services

The web services that are available for Health Care Reform.

A.1.1 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.

Request

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

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 being enrolled and details about assistors.

A.1.2 getEntitlementDetails

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

Request

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

Response

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

A.1.3 getHouseholdSummaryDetails

A plan management vendor calls the IBM Cúram Health Care Reform solution to notify us of a change in the status of an existing enrollment (e.g. when a carrier has finished processing the enrollment and made a policy ID available).

Request

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

Response

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

A.1.4 policyIDAvailable

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

Request

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

A.1.5 updateEmployerEnrollment

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

Request

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

Response

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

A.2 Health Care Reform schema elements

The schema that is used for Health Care Reform data.

EnrollmentDetails

Data Member	Type	Description
enrollmentID	Long	The enrollment key.

EligibilityAndDemographicDetails

Data Member	Type	Description
eligibilityDetails	eligibilityDetails	
persons	persons	
previousEnrollments	previousEnrollments	
assistors	assistors	
employerDetails	employerDetails	

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 remaining for the year.
monthsRemaining	Int	The number of months remaining 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	

persons

Data Member	Type	Description
person	List of person	

person

Data Member	Type	Description
personID	Long	Unique identifier for a person within the exchange
ssn	String	
firstName	String	
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 / Alaskan Native
isPrimaryContact	Boolean	Indicates whether the person is the primary contact for the group 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 assigned to each member.
taxFilerRelationshipList	TaxFilerRelationshipList	

Address

Data Member	Type	Description
addressLine1	String	
addressLine2	String	
city	String	

county	String	
state	String	
zip	String	

TaxFilerRelationshipList

Data Member	Type	Description
taxFilerRelationships	List of TaxFilerRelationship	

TaxFilerRelationship

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

previousEnrollments

Data Member	Type	Description
enrollment	List of enrollment objects	

enrollment

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

previousEnrollees

Data Member	Type	Description
enrollee	List of enrollee objects	

enrollee

Data Member	Type	Description
personID	Long	

assistors

Data Member	Type	Description
assistor	List of assistor objects	

assistor

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

PhoneNumber

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

EmployerDetails

Data Member	Type	Description
employerID	Long	
coverageStartDate	Date	

EntitlementUpdateDetails

Data Member	Type	Description
enrollmentID	Long	
updatedPremiumTaxCredit	Double	

HouseholdSummaryDetails

Data Member	Type	Description
effectiveDate	String	
zipCode	String	
personList	PersonList	

PersonList

Data Member	Type	Description
persons	List of Person	

person

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

EmployerEnrollment

Data Member	Type	Description
employerEnrollmentID	String	The employer enrollment identifier.

EmployerEnrollmentReceived

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

Notices

Trademarks

Printed in the Republic of Ireland