Cúram 8.2

REST API Accelerator Guide

Note

Before using this information and the product it supports, read the information in Notices on page 77

Edition

This edition applies to Cúram 8.2.

Contents

Chapter 1 Cúram REST API accelerator

The REST API accelerator is built on the Cúram Platform Version 6.1, Cúram Outcome Management, and the Adult Social Care accelerator. The REST API accelerator demonstrates how to integrate an external mobile application with Cúram by using the REST API infrastructure.

1.1 Overview of the REST API and mobile accelerators

The Cúram REST API accelerator provides a worked example of creating REST APIs, which are based on requirements that are specified for the Adult Social Care mobile application. The example is designed to help you with creating your own REST APIs.

REST API accelerator personas

For the REST API accelerator, three personas were defined; the business analyst, the Cúram developer, and the mobile application developer. The accelerator documentation provides guidance on how each persona can complete their tasks.

Table 1: REST API accelerator personas

Persona	Description	Tasks	Relevant Persona Documentation
Business Analyst	The business analyst performs a fit gap analysis and defines the mobile application functional requirements. The analyst might be solely responsible for the functional requirements or might work together with a UI designer to develop the wire frames for the mobile application.	• Performs a fit gap analysis of the project requirements against the Cúram application. • Defines the mobile application functional requirements. • Documents the mobile application wire frames.	For more information, see the following sections: • Overview of the Adult Social Care mobile application on page 19 • Performing the Adult Social Care mobile application caseworker scenario on page 53 • 1.3 Analyzing the requirements on page 24

Persona	Description	Tasks	Relevant Persona Documentation
Cúram developer	A Cúram developer designs and implements the custom REST APIs.	• Designs and performs a fit gap analysis to understand which Cúram APIs can be reused. • Models the REST APIs by using IBM® Rational® Software Architect Designer. • Implements and tests the REST APIs. • Implements the configuration-based REST API definitions that use Cúram facades.	For more information, see the following sections: • Overview of the REST API accelerator on page 12 • 1.2 Installing the REST API accelerator on page 21 • Overview of the Adult Social Care mobile application on page 19 • Performing the Adult Social Care mobile application caseworker scenario on page 53 • 1.3 Analyzing the requirements on page 24 • Creating a custom REST API
Mobile application developer	The mobile application developer builds the mobile application in their preferred application development environment. They can integrate with the Cúram REST APIs to use the business functionality that is required for the mobile application.	• Designs the user interface. • Implements the application logic for the user interface. • Integrates with the REST services to consume data.	For more information, see the following sections: • Overview of the Adult Social Care mobile application on page 19 • Installing the Adult Social Care mobile accelerator on page 51 • Performing the Adult Social Care mobile application caseworker scenario on page 53 • 1.3 Analyzing the requirements on page 24 • Integrating with a mobile application on page 50

Architecture overview

The following architecture diagram displays the connection between the mobile application and the Cúram application server.

Mobile application accelerator

The mobile application is built by using IBM MobileFirst and is simulated through the IBM MobileFirst development environment. The IBM MobileFirst platform was used to build the mobile application. It is a suite of products that enables partners and customers to efficiently build and deliver mobile applications for their enterprise.

From using the IBM MobileFirst simulator in your development environment, the mobile application is rendered with HTML and CSS. When the user selects an action in the mobile application, it triggers business logic that is written as a JavaScript application.

The JavaScript application sends a REST API request by using an IBM MobileFirst HTTP adapter with a data payload written in JSON or HTTP.

Cúram REST API accelerator

On the Cúram server side, a REST API intercepts the call. The Cúram application server processes the REST API request with a server-side application, such as Enterprise Java components.

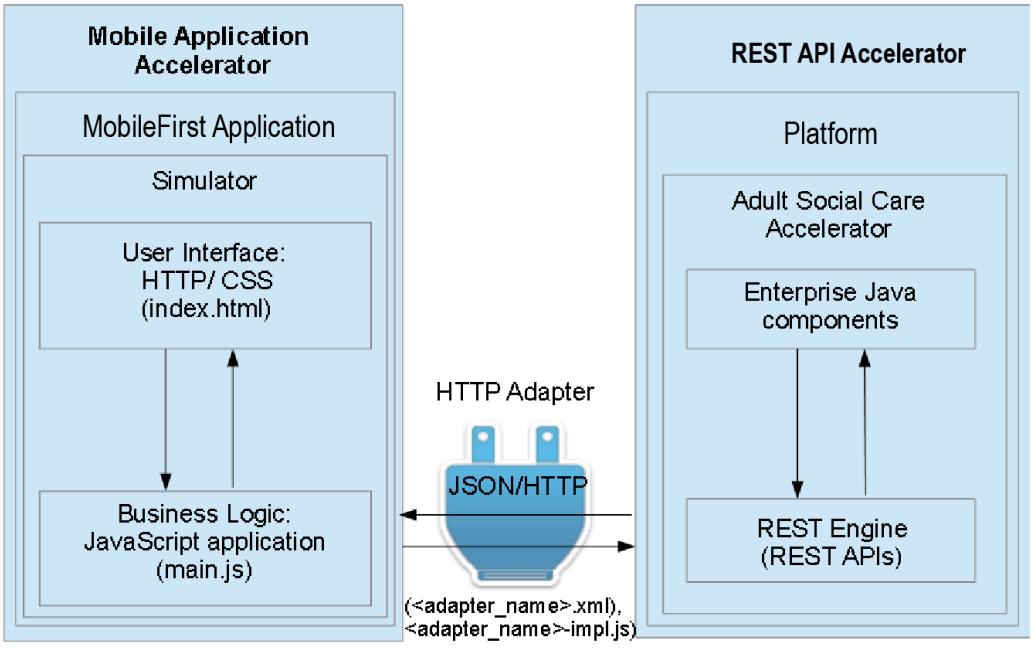


Figure 1: An architecture diagram

Overview of the REST API accelerator

The main objective of the REST API accelerator is to familiarize you with how to create your own custom REST APIs.

The REST API accelerator is based on the existing Adult Social Care accelerator. It would be beneficial for you to go through the overview and performing sections for the Adult Social Care accelerator first to gain a fuller understanding of the Cúram components involved.

The REST APIs that are built as part of this accelerator can be integrated with an external system. For this accelerator, the decision was made to integrate the REST APIs with a mobile application, which was built by using IBM MobileFirst.

The mobile application is not the main focus of this accelerator. The main objective is to provide you with guidance on how to create your own custom REST APIs. For this reason, less emphasis was placed on the front-end mobile application, in particular the look and feel. Documentation for how the mobile application was built is out of scope for this accelerator. Steps are included on how to install the mobile application, where you can see yourself how the application was implemented.

For more information about the Adult Social Care Accelerator, see the *Adult Social Care Accelerator Guide* guide.

REST API accelerator business process

A caseworker uses the Cúram Adult Social Care mobile application on their tablet to search for a client and view their detailed client profile. From the **Client Profile** page, the caseworker can view client and case information. The caseworker can also create and manage practitioner certificate evidence and diagnosis evidence that is associated with the client's Adult Social Care integrated case.

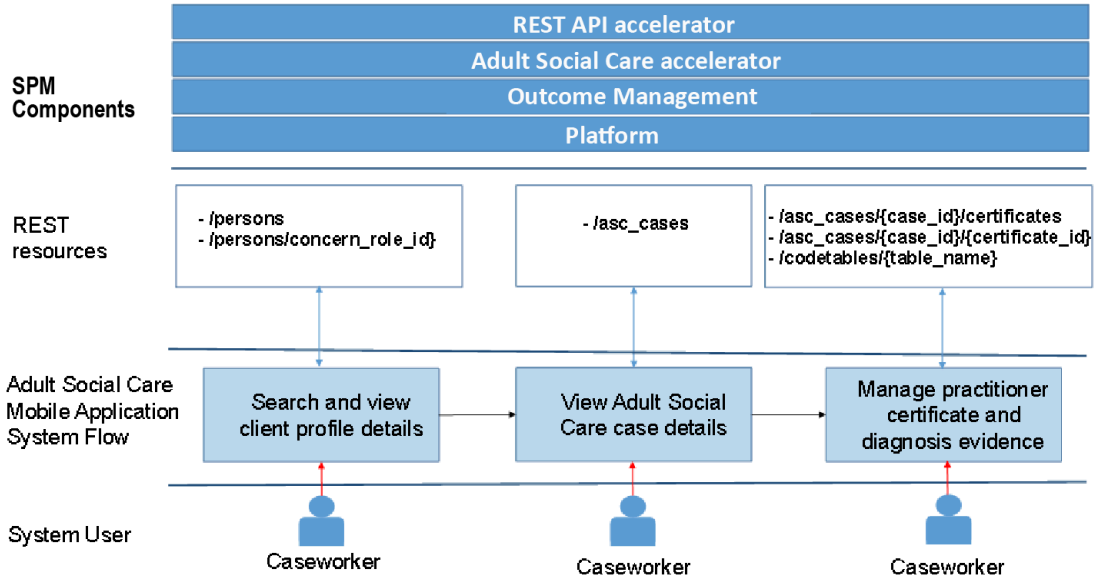


Figure 2: The Cúram REST API overview diagram

The business process uses the following components to satisfy the requirements:

- **Cúram Platform**
  This platform provides the processing and components necessary to enable the business process. The Cúram Case Management component helps to search for a client and to view their client details and case information.
- **Cúram Outcome Management**
  Outcome Management is a separately purchased application module. It is designed to help organizations assess needs, establish goals, plan for goal attainment, and track progress for citizens and their families by creating and managing outcome plans. The Adult Social Care mobile application uses REST APIs included in this module.
- **Adult Social Care accelerator**

  The Adult Social Care accelerator provides a straightforward, worked example of a basic application work flow. The example is designed to familiarize you with the key features

and underlying infrastructure of a basic application. The Adult Social Care accelerator is available for download only to members of Merative such as Lab Services developers who are assigned to work on a Cúram project on a customer site. However, you can reproduce the sample application on the default product by following the detailed step-by-step instructions in Building the Adult Social Care application.

- **Cúram REST API accelerator**
  The REST API accelerator is based on the Adult Social Care accelerator. It demonstrates how to integrate an external mobile application with Cúram by using the Cúram REST API infrastructure. The accelerator is available for download only to members of Merative such as Lab Services developers who are assigned to work on a Cúram project on a customer site..

REST API design decisions

The main decisions that were made during the design of the REST APIs for Adult Social Care are examined here.

Initially, three REST APIs were designed to retrieve the Adult Social Care integrated case details, payment details, and eligibility and entitlement details. The payment details, and eligibility and entitlement details are retrieved from the product delivery case. After an architecture review within Cúram, the decision was made to combine these three REST APIs into one.

1. Case details, payment details, and eligibility and entitlement details are separate concepts in Cúram. However, for the purposes of this API they represent the data that is associated with an Adult Social Care case. Granularity is an important consideration when you design a REST API, and a balance must be struck between the amount of data that is returned in a resource and the number of resources required. The types of operations that are performed on the data should also be considered when defining the granularity. In this instance, no additional operations were required on the case details, payment details, and eligibility and entitlement details, resulting in a single resource.

   The resource `/asc_cases` is specifying the acronym for Adult Social Care for this accelerator. When you design your resource names, you should try to stay away from the use of acronyms. It is worth noting that there are restrictions on the length of the resource name and it might not always be possible to avoid using them.

2. The second decision was made during the design of the resource (GET, PUT, POST, and Delete) methods for both the practitioner certificate and the diagnosis dynamic evidence types. The practitioner certificate and diagnosis details are stored as dynamic evidence, and both are configured for the Adult Social Care accelerator. The entities are configured in a parent-child relationship, with the practitioner certificate as the parent and diagnosis as the child.

   Due to the parent-child nature of the evidence, it was decided to structure the diagnosis as part of the practitioner certificate resource representation. The diagnosis evidence is always referenced within the context of the parent resource. That is, a practitioner certificate record must exist before a diagnosis record can be created for it.

   As diagnosis is nested within the practitioner certificate, the diagnosis resource (GET, PUT, POST, and Delete) methods are performed as part of the `modifyCertificate` REST

API. That is, diagnosis records can be created, updated and deleted as part of modifying a practitioner certificate.

3. Lastly, the decision was made to reuse three existing Cúram REST APIs that fitted the business requirements for the Adult Social Care mobile application. To keep this accelerator as simple as possible, the assumption is that these APIs fit.
   - The GET `/persons` REST API. This REST API takes the full name, which is the combination of the first and surname that is separated by a space. It is not case-sensitive and if you pass in one name, searches across both first and surnames.
   - The GET `/persons/{concern_role_id}` REST API to retrieve a person details.
   - The GET `/codetables/{table_name}` REST API to retrieve a code table details.

Related concepts

Nested resources on page 31
Nesting resources provide REST API consumers an easy and efficient way to manage data by allowing the consumer to send and receive only the required object. The nested resource must be a business object, that is, it must still represent a complete business object.

REST API development process

The following overview includes information for the development process, which was done as part of the creation of the REST API accelerator. It is not intended to go into development methodology, but for you to use as a guidance for the creation of your own custom REST APIs.

Process diagram

The following diagram illustrates the different processes that are required for creating your own REST APIs, from design right through to integration.

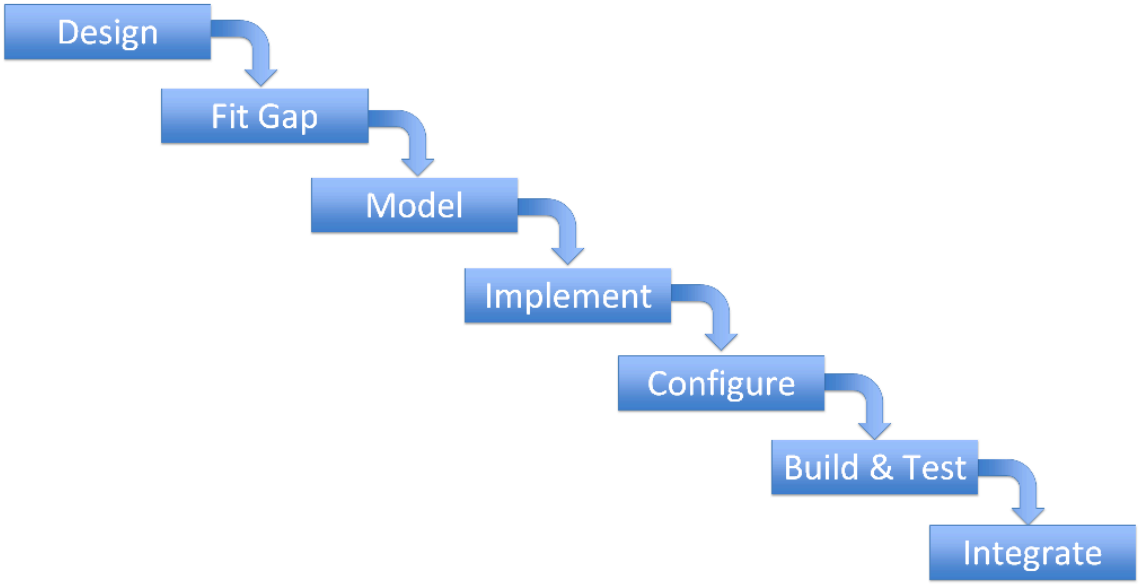


Figure 3: The development process diagram.

REST API implementation overview

The implementation overview provides a short description of all the different processes that are required for creating a custom REST API, from design right through to integration.

Before you begin

Before you start designing your own REST APIs, it is beneficial for you to read the 'Developing Cúram REST APIs' , and the 'Connecting to a Cúram REST API' documentation. Some of the content, which exists in this documentation, is also referenced where applicable as part of this accelerator.

It is also recommended that you review the existing REST APIs supported by Cúram to determine whether they meet the needs of your mobile application. Follow the steps that are defined in the 'Using existing Cúram REST APIs' documentation to perform your review of the existing REST APIs supported by Cúram.

Alternatively, you can create your own custom REST APIs. The following overview describes the development process, which was used for creating the REST APIs for the Adult Social Care mobile application.

About this task

Design

As part of your design you need to define the resources, the methods and the JSON representation, which are the major elements of the API.

- Resources: are identified by using URI path components.
- Methods: define the available operations on a REST resource.
- Representations: are the definition of the structure of the business object that is being represented.

Fit gap

It is recommended that you review the existing REST APIs supported by Cúram to determine whether they meet the needs of your mobile application. You can perform this by using Swagger-enabled tools.

When you know which REST APIs need to be created, you need to perform a fit gap to understand which Cúram APIs can be used to retrieve the information that is required for each one.

Test Driven Development (TDD) was used, where unit tests were created before the REST APIs were modeled. The benefit of creating unit tests up front proves that the Cúram APIs defined as part of the fit gap meet the needs of each of the REST APIs. It also helps in understanding exactly what needs to be modeled, which eliminates the need for remodeling. Handcrafted structs were created to assist with the unit tests and were removed when the structs were modeled by using Rational Software Architect.

Model

For modeling the REST APIs, the following are the key differences that were found from the typical web services modeling.

- The stereotype is of the type *rest*.
- The aggregation names are different at the top level; you must specify *data* if you want to follow the REST standards.
- There are new conventions to set mandatory fields on input keys for the GETS, as well as for the creates and modifies that you are familiar with.

Implement

After the modeling was complete, the implementation was straightforward due to the Test Driven Development approach that was taken. The entire design and workings of the APIs were driven out by JUnit testing. The JUnit tests tried out the APIs that were going to be used by the modeled code. The implementation consisted of moving the code from the JUnit tests into the APIs and then updating the tests to call those APIs.

The `modifyCertificate` method, that provides the implementation for the certificates resource PUT method, was the most complex REST API to implement. The reason for this is that the resource (GET, PUT, POST and DELETE) methods for the diagnosis evidence type need to be implemented as part of the modify certificate operation. The reason is due to the parent-child nature of the evidence, with diagnosis structured as a nested resource of the practitioner certificate.

Resource configuration

You need to create the *`%CURAM_DIR%/EJBServer/%COMPONENT_NAME%/rest/config/ResourcesConfig.xml`* file. The REST configuration file is an XML file that defines the REST resources, including the versions and supported methods, in a REST API. *`%CURAM_DIR%`* is the Cúram installation directory, which by default is *`C:\Merative\Curam\Development`*.

All *`ResourcesConfig.xml`* files are combined with the facade and struct information from the Cúram model to generate a Swagger document. The Swagger document defines all the REST resources and methods that are supported for the API, and a Swagger document is generated for each version of the API. You can define the following resource configurations for each REST API:

- The resource path.
- The type of method, whether a GET, PUT, POST, or DELETE.
- The façade class name, for example, `AdultSocialCareAPI`.
- The method name, for example, `readASCCases`.
- A tag name, which allows the resources to be displayed together when you view a Swagger document, by using the Swagger-enabled tools.

You can also create the *`RestConfig.properties`* file to update the title to be displayed within the Swagger document.

Build & Test

After you build Cúram, a number of different mechanisms can be used to test the REST APIs:

- Unit testing.
- Swagger-enabled tools with the Cúram application.
- Mobile application with the Cúram application.

Integrate

The Adult Social Care mobile application, which is built by using IBM MobileFirst, was used to integrate with the REST APIs. Adapters were created within IBM MobileFirst, which connects the mobile application with the Cúram application server. An adapter is used for integrating between the client application, through the IBM MobileFirst Platform, with enterprise back-end applications and cloud services.

Overview of the Adult Social Care mobile application

The Adult Social Care mobile application is a basic hybrid mobile application. The application is built on top of Adult Social Care and is designed to familiarize you with how to integrate the REST APIs with the mobile application. The mobile application represents a basic path through a Cúram social program. The mobile application was designed using a cascading style sheet that is optimal on a tablet device.

The mobile application is simulated through the IBM MobileFirst development environment, and the deployment to and testing on an android or iOS device is out of the scope for this accelerator. Also, the mobile application is not productized and does not perform any security integration. The main focus of the mobile accelerator is to integrate with the Cúram REST APIs.

Using the mobile application, the caseworker can search for a client and view client contact information, payment information, and eligibility and entitlement information. The caseworker can also create and manage practitioner certificate and diagnosis evidence that is associated with the client's Adult Social Care integrated case.

The Adult Social Care mobile application business scenario

The goal of the Adult Social Care mobile application scenario is to allow a caseworker to access limited client and case information from an off-site location.

Jane is a Community Welfare Officer and deals primarily with people who are incapacitated. Jane spends her time with clients and managing their cases.

Jane needs to finish some work at home tonight that she didn't have a chance to finish at the office today. Jane has a few updates to make to her client, James Smith's, Adult Social Care case.

Jane navigates to the Cúram mobile application on her tablet. She searches for James Smith and accesses his detailed client profile. She can see his full address, and contact details. From the client profile she can also see that he has an on-going Adult Social Care case, which was created a month ago, and he’s receiving Sickness Benefit.

She scans through the case information and looks at the evidence. She can see he has a practitioner certificate that began a month ago and is about to expire. He is diagnosed with lower back pain and neck pain.

Jane can also see the current determination and knows that he is receiving Sickness Benefit while he is out of work. The mobile application allows her to learn about James and his benefits at a glance. She can see lots of important summary information without having to access different screens, and print information.

James was recently hospitalized for cardiovascular disease. He spent the last two weeks in the hospital. The doctor told him that he cannot return to work for 8 more weeks. Jane has a copy of the practitioner certificate document. She extends the existing practitioner certificate evidence end date for another 8 weeks. She also creates the new diagnosis evidence. When she is complete, all the case evidence is up to date.

Adult Social Care mobile application example requirements

A targeted set of business requirements were defined for the mobile application accelerator.

The business requirements are as follows:

- Search for an existing client in the system.
- Manage practitioner certificate and diagnosis case evidence. The user can create, retrieve, update, and delete the evidence from the **Client Profile**page.
- View the eligibility and entitlement information for a client. The user can view the current determination results for the active benefit case.
- View benefit payment information. The **Client Profile** page must display the last payment that was made to the client and the next payment due.

Adult Social Care mobile application design decisions

Analysis of the mobile application requirements resulted in four key design decisions.

1. The first design decision was to use a hybrid application for the Adult Social Care mobile application. The hybrid application was selected because it provided a way to quickly build a mobile application to test with the custom REST APIs.
2. The second design decision was to display both client and case information on the mobile application. The **Client Profile** page displays client and case information that is pulled from the registered person, Adult Social Care integrated case, and associated product delivery case. It allows the user to view the information from one place without having to navigate to separate pages, as is required in the existing Adult Social Care application.
3. The third design decision was to display key features and content that are core to the mobile application use case. The information on the mobile application is a subset of what you find on the Adult Social Care application. The mobile application displays only the data that is required to perform the use case.
4. The fourth design decision was to display only the most recent information on the mobile application. Only the most recent, active practitioner certificate evidence record, and its associated active diagnosis evidence records, are displayed. Also, the most recent determination result from the product delivery case is displayed. The mobile application supplements the existing Adult Social Care application. If needed, the user can access older data from the Adult Social Care application.

1.2 Installing the REST API accelerator

Download and install the REST API accelerator. The REST API accelerator is available for download only to members of Merative, such as Lab Services developers who are assigned to work on a Cúram project on a customer site.

Installing prerequisites

You must follow the prerequisites before you install the Cúram REST API accelerator. Make sure to follow the prerequisites and the installation steps that are specified in the release notes for each of the installers.

Procedure

1. Install a Cúram Development Environment for application development. The basic Cúram Development Environment consists of the Cúram Application Development Environment (ADE), a Java IDE, and IBM® Rational® Software Architect Designer.
2. Install Cúram Platform.
3. Install the Cúram Outcome Management application module.
4. Install the Adult Social Care accelerator.
5. Install the Cúram REST API accelerator.

   For more information about how to install a development environment, see the *Development Environment Installation Guide*.

Generating the Cúram REST APIs

Complete the following steps for building the Cúram REST APIs and run the API resources in the development environment by using Tomcat to view the Swagger document.

Procedure

1. From the *`%CURAM_DIR%`* directory, update the *`SetEnvironment.bat`* and *`SetEnvironment.sh`* files.

   `%CURAM_DIR%` is the Cúram installation directory, which by default is *`C:\Merative\Curam\Development`*.
2. Add the *CATALINA_HOME* environment variable.
   This environment variable defines the home directory of the Tomcat installation, and is used to automatically deploy the REST API resources into Tomcat.
3. From the *`%CURAM_DIR%/EJBServer`* directory, run the following build target:

   ```
   build rest
   ```

   This build target combines the defined REST resources from each component and deploys the REST API to the Tomcat web server in the development environment.

4. From Eclipse, start your application server and Tomcat as normal.
5. Using a web browser, open the following URL to confirm successful deployment of the REST API:

   ```
   http://localhost:9080/Rest/api/definitions
   ```

   where 9080 is the default Tomcat port.

Results

A list of the Swagger documents for each version of the API defined is displayed. Select a version to open the Swagger document for that version.

Testing a REST API for Adult Social Care

To test that the Cúram REST API accelerator is built successfully, complete the following steps to view the JSON representation for one of the REST APIs.

Procedure

1. From the Adult Social Care accelerator, run through the performing the Adult Social Care caseworker scenario to create an Adult Social Care case for James Smith.
2. From a web browser, enter the following URL to confirm whether the *asc_cases* GET resource method is working. A JSON object with an array of case details is displayed.

   ```
   http://localhost:9080/Rest/v1/asc_cases?concern_role_id=101&_limit=1
   ```

Results

The following example JSON representation is displayed successfully in the web browser.


```
{
  "data" : [
    {
      "concern_role_id" : "101",
      "case_id" : "8644096534784245760",
      "startDate" : "2015-07-26T23:00:00.000+0000",
      "expectedEndDate" : null,
      "receivedDate" : "2015-07-26T23:00:00.000+0000",
      "registrationDate" : "2015-07-26T23:00:00.000+0000",
      "priorityCode" : {
        "tableName" : "CasePriority",
        "value" : "CP1",
        "description" : "High"
      },
      "integratedCaseType" : {
        "tableName" : "ProductCategory",
        "value" : "PC24000",
        "description" : "Adult Social Care"
      },
      "decisions" : [
        {
          "startDate" : "2015-08-30T23:00:00.000+0000",
          "endDate" : null,
          "amount" : 0.00,
          "frequency" : "",
          "decision" : {
            "tableName" : "CaseDetIntervalResult",
            "value" : "CDIR2",
            "description" : "Not Eligible"
          }
        },
        {
          "startDate" : "2015-07-26T23:00:00.000+0000",
          "endDate" : "2015-08-29T23:00:00.000+0000",
          "amount" : 132.23,
          "frequency" : "Weekly",
          "decision" : {
            "tableName" : "CaseDetIntervalResult",
            "value" : "CDIR1",
            "description" : "Eligible"
          }
        }
      ],
      "payment" : {
        "lastPayment" : 0.00,
        "nextPayment" : 132.23,
        "lastPayDate" : null,
        "nextPayDate" : "2015-07-26T23:00:00.000+0000"
      }
    }
  ]
}
```


Related tasks

1.3 Analyzing the requirements

Use the following practices for analyzing the default system functions against the customer's mobile application business processes and to document project requirements.

Completing a fit gap analysis scenario

You can use the following guidance to complete a fit gap analysis between the project requirements and the default application.

Fit gap analysis scenario

Fiona, a business analyst, is asked to do a fit gap analysis of the differences between the Adult Social Care application and the customer’s mobile application project requirements. Fiona is focusing on the requirements for the practitioner certificate and person phone details.

These are the project requirements:

- The mobile application must allow the user to create and manage practitioner certificate evidence. The certificate details include the received date, medical practitioner name, medical certificate type, examination date, certification from date, certification to date, date certificate was signed, claim form signed indicator, inpatient indicator, and facility name.
- The mobile application must display the client's home telephone number.

At the start of the fit gap analysis, Fiona must have a solid understanding of the Cúram default features. Fiona read the overview and performing sections for the Adult Social Care accelerator first to gain a fuller understanding of the Cúram components involved.

Analyzing the default functionality

You can use the following resources to help you with your analysis.

Reading product documentation

You can read the product documentation to learn about Cúram product features.

About this task

The product documentation includes business and technical documentation for specific versions and editions of the Cúram product family. Links to Redbooks, and white papers are also provided when articles about the product are available. Business analysis documentation is written to provide an overview of the business processes supported by Cúram applications. You can use this information to understand the default features, and how they can be applied to your business processes.

For the business analyst scenario, Fiona finds documentation on how phone numbers are managed in Cúram. The documentation provides a high-level overview of how phone numbers are maintained as person evidence.

Evaluating the default application scenario

Fiona logs in to the Cúram application to explore the default application features to see whether they meet the project requirements, or if there are gaps. Fiona is analyzing the practitioner certificate and person phone project requirements.

Complete the Adult Social Care application intake script

Fiona completes the Adult Social Care application intake script to see if the script captures the required data.

Fiona creates an application case by entering the client's details, a practitioner certificate, and diagnosis information. After examining the default Adult Social Care application, Fiona notes gaps in the script. The practitioner certificate questions do not capture the inpatient indicator and facility name.

Examining the practitioner certificate evidence

Fiona examines the default practitioner certificate evidence to see whether there are any gaps.

Procedure

1. Select the **Care and Protection** > **Applications** > **Cases** tab to view the application case.
2. Select the reference number to open the application case.
3. Select the **Evidence** tab to view the default evidence types that are associated with the application case. Fiona can see that the inpatient indicator and facility name are not captured on the practitioner certificate evidence. This is a gap.

Examining person evidence

Fiona is analyzing the person phone requirements. After reading participant documentation, she knows there is a fit with the Cúram product, but needs to do further analysis to determine the alignment. She decides the next thing she should do is explore the person phone evidence on her local installation of the Adult Social Care application.

Before you begin

The person must exist on the Cúram system.

Procedure

1. From the **Person** tab, select the **Evidence** tab and select **New Evidence** from the page level menu.
2. The system displays the **New Phone Number** window. Select the **Phone Type** drop-down field.

What to do next

Fiona can see the phone number details on the New Phone Number modal page. She verifies it is the same as the layout required for her project. She sees that the label 'Phone Number' is used instead of 'Telephone', which is inconsistent with other systems in her project. Fiona notes this as a gap. She also notes that ‘Home’ is not displayed in the Phone Type code table.

Now that Fiona has tested the application's person phone functionality, she knows that there is a medium-level solution fit since configuration is required.

Cúram Analysis Documentation Tooling (CADT)

The Cúram Analysis Documentation Tooling (CADT) is a tool that technical users can download and run on a Cúram application to generate analysis documentation that is specific to that application. Technical users can share the analysis documentation with non-technical users to facilitate technical and business analysis of a Cúram application. The analysis documentation primarily supports fit gap analysis and customization impact analysis that relates to Cúram pages and database tables.

Use the analysis documentation to help you to explore the metadata and business functions of an Cúram application and the interactions between them. It supports you in exploring the metadata that is associated with those pages and database tables. In addition, you can see information about related artifact types like Tabs, Domain Definitions, Code Tables, Message Files, Application Properties, and Workflows.

The Cúram Analysis Documentation Tooling (CADT) is available for download from Cúram Support. You must request access to download software.

Analyzing requirements using the Cúram Analysis Documentation Tooling (CADT)

You can use the Cúram Analysis Documentation to determine which application artifacts are used by a particular feature, in this case the person phone feature. Fiona can further explore application artifacts by using the Cúram Analysis Documentation, which is generated by using the Cúram Analysis Documentation Tooling (CADT).

Before you begin

The Cúram Analysis Documentation Tooling (CADT) is run on the Adult Social Care application that is built on top of the Cúram Platform and Fiona is provided with the Cúram Analysis Documentation.

About this task

You can complete these steps of the design process, but information is also provided on how Fiona completes them as part of the Fit Gap Analysis scenario.

Procedure

1. From the Cúram Analysis Documentation search text box, enter **Phone**, select **Pages** from the drop-down list, and click **Search**.

   Fiona can see from the search results several pages that contain the string 'phone'. Fiona wants to understand the context in which phone numbers are used in Cúram. By glancing at the pages that contain the string 'phone', she sees that phone number details are used in the Cúram application pages.
2. From the search results, select **New Phone Number (Participant_createPhoneNumber)** from the Name column. You see the **New Phone Number** page details.

   Fiona verifies that it is the same as the layout used for her project. She sees that the label 'Phone' is used instead of 'Telephone', which is inconsistent with other systems in her project. Fiona notes this different label as a gap.

3. Scroll to the Fields section. Select the **Phone_Type_Code** from the Fields section. You see the Domain Definition details for **Phone_Type_Code**.
4. From the **Options** domain definition details, select the **PhoneType** link to see the code table details for the phone type code table.

   Fiona observes that there is no 'home' telephone number type in the PhoneType code table. This is a requirement for her project, so she notes this as a gap; however, she needs to determine if this is a requirement across the organization.
5. From the **Related Pages** section, it contains a list of pages that use the PhoneType code table. Any modification made to the PhoneType code table impacts all of the pages in this list.
6. Scroll to the Phone section. You see the phone number attributes.

   Fiona verifies that the phone number consists of the same three elements as her project's phone number (country code, area code, and phone number). She confirms that the number can be stored in the format that is required by the project.
7. Scroll to the Messages section. You see the validations for telephone numbers and an indicator to show if it is configurable. You see the fields that are mandatory and the ones that are conditionally displayed.

   Fiona verifies that the messages are consistent with the project's requirements.
8. From the Cúram Analysis Documentation search text box, enter **Phone** and select **Code Tables** from the drop-down list and click **Search**.

   Fiona can see from the search results a list of code tables that contain the string 'phone'. One of these code tables is the BusinessPhoneType code table. Fiona remembers when she was assessing the impact of changing the word 'phone number' to 'telephone', there were other screens in the application that are related to phone numbers that the project plans to use, but that don't seem to use the PhoneType code table. She wants to make sure that there are no changes that are required to any other code tables. Based on the information that is shown to her for the BusinessPhoneType code table, she realizes that this code table contains phone number types that businesses would use and should therefore not contain the value 'home'. This is not a gap.

Defining the mobile application requirements

The mobile application is a software application that is developed specifically for use on small, wireless computing devices, such as smart phones or tablets. The Cúram mobile application provides the caseworker with quick and easy access to Cúramdata. You must define the business requirements for your Cúrammobile application.

Before you begin defining the mobile application requirements, you must decide what format is best suited for your project. Mobile applications are categorized according to whether they are web-based or native applications, which are created specifically for a given platform. A third category, hybrid applications, combine elements of both native and web-based applications.

Native applications, web-based applications, and hybrid applications are all ways to meet the needs of the mobile user. There is no unique best solution: each type has strengths and weaknesses. The choice of one versus the other depends on the organization's project

requirements. Table 1 provides a list of some of the strengths and weaknesses for each mobile application type.

Table 2: Mobile Application Types

Mobile Application Type	Strengths	Weaknesses
Native application	• Access to all the device-specific features, including GPS, camera, gestures, and notifications. • Can be used without an internet connection. • Provides a full experience to the user on their iOS or Android smartphone. • Most responsive option that is key to usability. • As native applications are platform-specific and written in the platform's native language, they can be fully optimized for the platform making them more efficient.	• Higher development costs. Building a native application requires more specialized talent. Developers not familiar with the development process for their target platform need to upskill in order to develop the application. • Maintenance can be complicated for users and developers. Changes must be packaged in a new version and placed in the app store. • Less discoverable. The user must go to the app store, search for an application, and download it.
Web-based application	• Most discoverable. A user can easily search for a web application by using a search engine. • The web-based application can typically be developed much faster than a native application. • Maintenance is simple. It can be done as often as needed. • Lower development costs. It’s cheaper to develop hybrid and web applications, as these require skills that build upon previous experience with the web. • No need to distribute the software to machines that use the application. • Any application updates are made to the application alone and are immediately available to the user. • Inherent cross platform support as they run solely on the browser and are platform agnostic.	• The web-based application system performance can be much slower than the native application. • Cannot do work offline. • Web applications are not optimized for the platform they are running on as they are not written in the platform's native programming language. • Cannot use any of the platform's features such as the camera, contacts list, accelerometer. • Web applications cannot access platform-specific UI features so they cannot avail of the same style that native and hybrid applications can avail of.

Mobile Application Type	Strengths	Weaknesses
Hybrid application	• Less expensive option for organizations that want to sell an application in an app store without spending significant development effort on a native application. • Maintenance is simple. It can be done as often as needed. • Lower development costs. It's cheaper to develop hybrid and web applications, as these require skills that build up on previous experience with the web. • The application can target multiple platforms reducing cost of development and negating the need to hire experts to develop for each platform. • Hybrid applications can avail of the platforms UI style features allowing the application to be developed to look as close to a native application as possible.	• The hybrid application system performance can be much slower than the native application. • Hybrid applications cannot avail of a device's full range of features as they are often restricted while running through the WebView. • Hybrid applications will not be fully optimized for the device they are running on as the bulk of their code is written in platform independent web development languages.

The mobile application is meant to provide quick access to client and case information. When you define the requirements, keep in mind that you do not need to recreate the existing Cúram functions in the mobile application. At the same time, do not create unnecessary new functions. The basic idea is to cut features and content to eliminate things that are not core to the mobile application use case.

You must decide what client and case information to display on the mobile application pages. Try to use only the most recent information unless there is a valid business reason to display older data. For example, if the user needs to create and maintain case evidence, then consider displaying only the most recent, active, evidence records. It might not be necessary to display older data as the caseworker can view that information from the Cúram application. You must also consider the impact that the older data has on system performance. It is important to make sure your mobile site loads quickly. Users can become frustrated if they must wait a long time to see your content. Any data that is not necessary can slow down the system performance.

As part of the requirements definition process, try to identify customer pain points that can be addressed by the mobile application. If possible, meet with real system users to validate assumptions about customer pain points.

When you design the user interface, follow the relevant UI design guidelines for your mobile application. Apple and Android each have their own guidelines. Following a method for displaying data and interacting with content makes your app easier to learn by end users. Following these conventions also ensures that your application is consistent with the other Cúram applications already on a mobile device.

Analysis results for the mobile application

Analysis of the mobile application requirements resulted in four key design decisions.

The first design decision was to use a hybrid application for the Adult Social Care mobile application. The hybrid application was selected because it provided a way to quickly build a mobile application to test with the custom REST APIs. Also, there were no iOS or Android development dependencies. For these reasons, the hybrid application was selected as the best solution.

The second design decision was to display both client and case information on the mobile application. The **Client Profile** page displays client and case information that is pulled from the registered person, Adult Social Care integrated case, and associated product delivery cases. It allows the user to view the information from one place without having to navigate to separate pages, as is required in the existing Adult Social Care application.

The third design decision was to display key features and content that are core to the mobile application use case. The information on the mobile application is a subset of what you find on the Adult Social Care application. The mobile application displays only the data that is required to perform the use case.

The fourth design decision was to display only the most recent information on the mobile application. Only the most recent, active practitioner certificate evidence record, and its associated active diagnosis evidence records, are displayed. Also, the most recent determination result from the product delivery case is displayed. The mobile application supplements the existing Adult Social Care application. If needed, the user can access older data from the Adult Social Care application.

1.4 Creating a custom REST API

You can use the following documentation as a basis for creating your own custom REST APIs.

REST API designs

Before you start developing REST APIs, ensure that you read through the design principles and design basics to ensure good REST API design.

Cúram REST API design principles

When you design custom REST API resources, follow these conventions to ensure good REST API design.

General design conventions

- Use plural nouns and lowercase letters for resource path names, for example, `/notes` for the note resource or `/persons` for the person resource.

- Use snake case for URL query parameters and resource names, for example, *concern_role_id* and *full_name*.
- Representations must remain consistent across operations. The same representations need to be used for GET, PUT, and POST.
- Identifiers must not be in URI form. Do not use HATEOAS style.

Conventions for collection resources and member resources

- A collection resource returns all objects in the collection, for example, `/persons` returns all persons in the system. Typically, collection resources have a required query parameter to reduce the results set, for example, `/persons/full_name=James Smith`.
- The response to a collection request is an object with a property named `data`. This property is an array that holds all of the matching members of the collection.
- A request for a collection where the query parameter does not match any resources returns a successful 200 status code and an empty data array.
- A member resource is an individual item in a collection, for example, `/persons/106`.
- A request for a member resource that does not exist returns a 4xx status code

Nested resources

Nesting resources provide REST API consumers an easy and efficient way to manage data by allowing the consumer to send and receive only the required object. The nested resource must be a business object, that is, it must still represent a complete business object.

To decide whether a resource should be nested, you must consider the following:

- If the nested array is a stand-alone entity that is referred to outside of the context of the parent resource.

  If the nested array is referenced outside the context of the parent resource, it may be defined as a root resource.
- If the resource is modified frequently.

  If a small part of a resource is frequently modified, each modification invalidates the cache state of the whole resource. In this case, it is good design practice to separate the content into a nested resource. For example, PUT requests on an entire resource that only adds a single item in an array, in cases where the array changes frequently, are wasteful and not a good design.
- If the size of the representation is large or small.

  If it is small, perhaps it can be included in the main representation. If it is large, it can dominate the main representation and might be more ideal as a nested resource.

Cúram REST API design basics

A Cúram REST API is a collection of URL resources that can be used to create, read, and modify data in a Cúram system.

Resources are complete business objects and each resource represents an object in Cúram. Each resource, which is identified by a path, has a set of methods that can act on the resource to create, read, modify, or delete the resource. The resource is represented by a JSON object.

REST is different from the RPC (Remote Procedure Call) style APIs developed within Cúram. RPC APIs usually target specific information or data to be displayed on a particular user screen, and for performing modifications to that information. REST APIs transfer and act on representations of complete business objects.

The design and granularity of REST APIs is important and you must carefully design the REST API before you begin development.

To design a REST API you must identify the major elements of the API, specifically:

- Resources: are identified by using URI path components.
- Methods: define the available operations on a REST resource.
- Representations: are the definition of the structure of the business object that is being represented.

Resource values for Adult Social Care

The following include the URL path resource values that are specified for the Adult Social Care mobile application. The sample resource URL paths can help you design your own resources for your custom REST APIs.

Table 3: Adult Social Care REST APIs

Resource Path	Description

`/asc_cases`	The collection resource is used to retrieve the list of Adult Social Care case details and its associated payment and decision details.
	
	The resource `/asc_cases` is specifying the acronym for Adult Social Care for this accelerator. When you design your resource names, you should try to stay away from the use of acronyms. It is worth noting that there are restrictions on the length of the resource name and it might not always be possible to avoid using them.
	
	The default sort order of this list is by the registration date. This resource supports 1 optional query parameter (_limit) to limit the list of items returned.
	
	For example, with the following URL, you can retrieve all the Adult Social Care cases:
	
	`http://<host>:<port>/Rest/v1/asc_cases/`
	
	To retrieve the latest Adult Social Care case details, the input struct supports 2 query parameters:
	
	• _limit
	• case_id (mandatory)
	
	Example URL for this REST resource:
	
	`http://<host>:<port>/Rest/v1/asc_cases?concern_role_id=101&_limit=1`

Resource Path	Description

/asc_cases/{case_id}/certificates	The collection resource to retrieve the list of practitioner certificates and its associated diagnoses details. The default sort order of this list is by the registration date. This resource supports 1 optional query parameter (_limit) to limit the list of items returned.
	For example, with the following URL, you can retrieve all the practitioner certificates:
	`http://<host>:<port>/Rest/v1/asc_cases/<case_id>/certificates/`
	You can also retrieve the latest certificate details with the following URL example:
	`http://<host>:<port>/Rest/v1/asc_cases/case_id=1253456789/certificates?_limit=1`

/asc_cases/{case_id}/certificates/{certificate_id}	The member resource is used to retrieve, update, and delete a certificate record. For the REST API PUT method, this method is used to perform the create, retrieve, update, and delete operations for the certificate associated diagnosis records.

Table 4: Cúram REST API resources

The mobile application is using the following 3 existingCúram REST APIs.

Resource Path	Description
/persons/{concern_role_id}	Returns a list of persons that match the search filters specified. The returned list is sorted by name, in alphabetical order.
/persons?full_name=<users first and last name>	Returns the details of a specific person.
/codetables/{table_name}	Return the details for each codetable. For example, `http://localhost:9080/Rest/v1/codetables/IncapacityDiagnosis`

Methods defined for Adult Social Care

To help you design the methods for your custom REST APIs, you can view the following sample methods, which are defined for the Adult Social Care mobile application.

Table 5: Methods

Method	Resource	Query Parameter	Description
GET	*/asc_cases*	• _limit • concern_role_id	A collection resource to return specific Adult Social Care case details for a person.

Method	Resource	Query Parameter	Description
GET	*/asc_cases/{case_id}/certificates*	• _limit • case_id	A collection resource that returns a list of practitioner certificate records for a person. To adheres to the REST API standard, this api is provided, but not used in the Adult Social Care mobile application. When the _limit query parameter is specified, it limits the number of certificate records to be returned and retrieves the latest Adult Social Care case based on the registration date. This resource is being used by the mobile application.
POST	*/asc_cases/{case_id}/certificates*		A member resource to create a practitioner certificate that is associated with a person's Adult Social Care case.
GET	*/asc_cases/{case_id}/certificates/{certificate_id}*		A member resource to retrieve a practitioner certificate and its associated list of diagnoses details.
PUT	*/asc_cases/{case_id}/certificates/{certificate_id}*		A member resource to modify a specific certificate, as well as create, update, and delete a diagnosis record. The full object is required to be passed in.
DELETE	*/asc_cases/{case_id}/certificates/{certificate_id}*		A member resource to delete a specific certificate or diagnosis. A delete only requires the path parameter (certificate_id).

JSON representation for Adult Social Care

To help you design the JSON representation for your custom REST APIs, you can view the sample JSON representation, which is defined for a REST resource. The following JSON representation is defined for the Adult Social Care mobile application.

The following JSON representation is the full representation for retrieving the Adult Social Care case details.

Resource Path	JSON

/asc_cases	{
	"concern_role_id": "24011",
	" case_id": "24012",
	" startDate": "2015-07-27",
	" expectedEndDate": null,
	" receivedDate": "2015-07-27",
	" registrationDate": "2015-07-27",
	" priorityCode": {
	"tableName ": "CasePriority" ,
	" value ": "CP1" ,
	" description ": "High" ,
	},
	" integratedCaseType": {
	"tableName ": "ProductCategory" ,
	" value ": "PC24000" ,
	" description ": "Adult Social Care" ,
	},
	" decisions": [
	{
	"startDate ": "2015-08-31" ,
	" endDate ": null ,
	" amount ": 0 ,
	" frequency ": "" ,
	" decision ": {
	"tableName ": "CaseDetIntervalResult" ,
	" value ": "CDIR2" ,
	" description ": "Not Eligible" ,
	}
	} ,
	{
	" startDate ": "2015-07-27" ,
	" endDate ": "2015-08-30" ,
	" amount ": 132.23 ,
	" frequency ": "Weekly" ,
	" decision ": {
	"tableName ": "CaseDetIntervalResult" ,
	" value ": "CDIR1" ,
	" description ": "Eligible" ,
	}
	}
	],
	" payment": {
	"lastPayment ": 132.23 ,
	" nextPayment ": 132.23 ,
	" lastPayDate ": "2015-07-27" ,
	" nextPayDate ": "2015-08-03"
	}
	}
	]
	}

The following JSON representation is the full representation for the practitioner certificate and its associated diagnosis details.

The practitioner certificate and diagnosis details are both stored in dynamic evidence entities that are configured for the Adult Social Care accelerator. The evidence entities are configured where the practitioner certificate is stored as parent evidence and the diagnosis details are configured as a child entity of the practitioner certificate.

As a result of the parent-child evidence structure, it was decided to structure the JSON representation where the diagnosis is defined as part of the certificate JSON object.

The resource (GET, PUT, POST and DELETE) methods for a diagnosis are performed as part of the same JSON representation as part of the *asc_cases/{case_id}/certificates/{certificate_id}* resource REST API.

Table 6: JSON representations

Resource Path	JSON

/asc_cases/{case_id}/certificates	{
	"certificate_id": "24027",
	" case_id": "24012",
	" fromDate": "2015-07-25",
	" toDate": "2015-08-30",
	" examinationDate": "2015-07-25",
	" signatureDate": "2015-07-25",
	" claimFormSignedInd": true,
	" receivedDate": "2015-07-26",
	" comments": "",
	" changeReason": null,
	" practitionerName": "John Lynch",
	" versionNo": 0,
	" updatedBy": "SYSTEM",
	" updatedOn": "2015-07-26T23:00:00.000+0000",
	" certificateType": null,
	" diagnoses": [
	{
	"diagnosis_id ": "24029" ,
	" examinationDate ": "2015-07-25" ,
	" receivedDate ": "2015-07-26" ,
	" changeReason ": null ,
	" name ": {
	"tableName ": "IncapacityDiagnosis" ,
	" value ": "ICD24000" ,
	" description ": "Gastroenteritis" ,
	} ,
	" versionNo ": 2 ,
	" updatedBy ": "SYSTEM" ,
	" updatedOn ": "2015-07-26T23:00:00.000+0000"
	}
	]
	}
	]
	}

Analyzing the REST APIs

Before you start developing REST APIs, ensure that you do a gap analysis on the existing REST APIs. You should also do a gap analysis of the existing Cúram functionality to determine the correct domain definitions to use for your resource representation. In addition, this gap analysis might identify any existing APIs within Cúram that can be used to implement the functionality of the new REST APIs.

Analyzing existing REST APIs

It is recommended that you review the existing REST APIs supported by Cúram to determine whether they meet the needs of your mobile application.

About this task

Follow the steps that are defined in the 'Using existing Cúram REST APIs' documentation to perform your review of the existing REST APIs supported by Cúram.

Related tasks

Analyzing the new REST APIs

You need to identify any existing APIs within Cúram that can be used to implement the functionality for the new REST APIs.

About this task

For the fit gap analysis of the dynamic evidence entities, it was a little difficult to identify the Cúram APIs that were required. As a result, the Cúram APIs that are used for the Adult Social Care mobile application are defined in the appendix section. You can use the same Cúram APIs for the creation of your own custom dynamic evidence configurations.

The fit gap analysis for Adult Social Care case details, the payment, and eligibility and entitlement details, was easier. You can use the Cúram Analysis Documentation, which is generated by using the Cúram Analysis Documentation Tooling (CADT), to search for the page which displays the details you need to retrieve. Once you have found your page, the **Technical information** section displays the **Page Load Interface**. For more information, see the appendix for the Cúram APIs.

Test Driven Development (TDD) was used, where unit tests were created for each REST API before the REST APIs were modeled. The benefit of creating unit tests up front proves that the Cúram APIs defined as part of the fit gap meet the needs of each of the REST APIs. It also helps in understanding exactly what needs to be modeled, which eliminates the need for remodeling. Handcrafted structs were created to assist with the unit tests and were removed when the structs were modeled by using Rational® Software Architect Designer.

Related tasks

Appendix on page 57
The following specifies the Cúram APIs that are used for implementing the Cúram REST APIs for the Adult Social Care mobile application.

Modeling the REST APIs

When you complete the REST API design, you must use IBM Rational Software Architect to model the facade, operations and structs.

REST API class diagram

Class diagram representing the modeling of the required structs and operations for the Adult Social Care REST API.

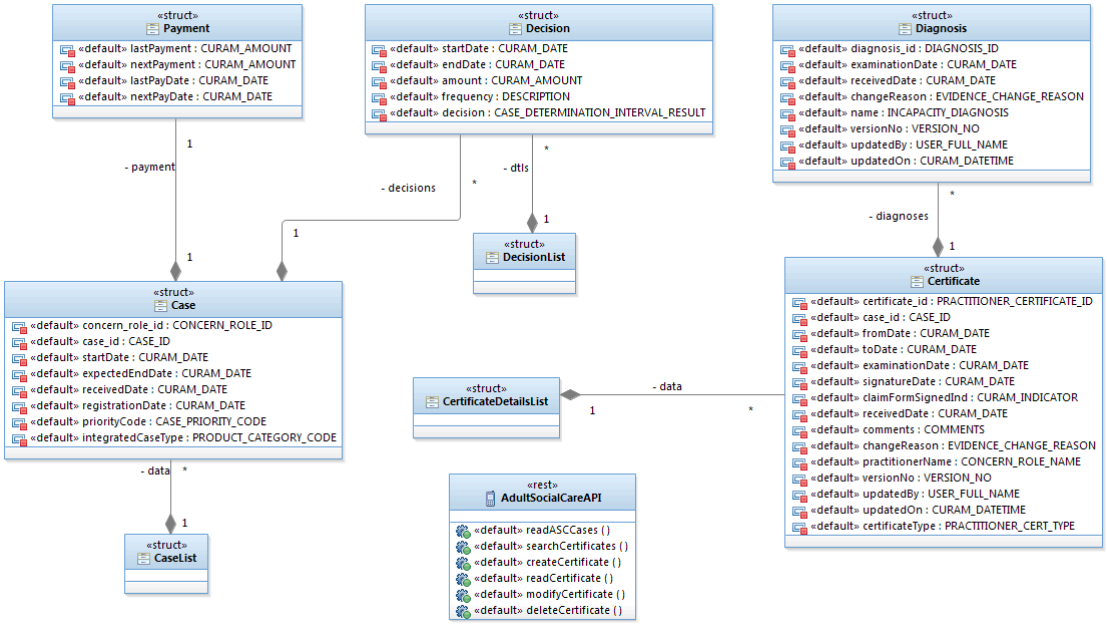


Figure 4: The Adult Social Care REST APIs class diagram.

Modeling the JSON representation

The JSON representation of the Adult Social Care REST API is modeled as a number of structs in IBM® Rational® Software Architect Designer.

Procedure

1. In the model, create the following domain definitions, with the specified types. Also ensure to associate the code-table with the domain definition in the model, where applicable.

Domain Definition Name	Basic Curam Type	Associated Codetable
INCAPACITY_DIAGNOSIS	CURAM_CODETABLE	IncapacityDiagnosis
PRACTITIONER_CERT_TYPE	CURAM_CODETABLE	PractitionerCertType
PRACTITIONER_CERTIFICATE_ID	INTERNAL_ID	N/A
DIAGNOSIS_ID	INTERNAL_ID	N/A

2. Model a new struct, called Case, to represent the `/asc_cases` REST resource. This struct contains nested content, and each piece of nested content requires a new Curam struct to be modeled. The Case struct contains the following parameters, with the relevant domain definitions in the model:

Parameter Name	Domain Definition
case_id	CASE_ID
concern_role_id	CONCERN_ROLE_ID
startDate	CURAM_DATE
expectedEndDate	CURAM_DATE
registrationDate	CURAM_DATE
receivedDate	CURAM_DATE
priorityCode	PRIORITY_CODE
integratedCaseType	INTEGRATED_CASE_TYPE

3. Model a new struct, CaseList, to represent a list of cases. This struct contains no parameters.
4. Create a one to many (1-*) aggregation between the CaseList struct and the Case struct and set the aggregation role to data.
5. Model a new struct, Decision, to represent a case decision. The Decision struct contains the following parameters, with the relevant domain definitions in the model:

Parameter Name	Domain Definition
amount	CURAM_AMOUNT
decision	CASE_DETERMINATION_INTERVAL_RESULT
startDate	CURAM_DATE
toDate	CURAM_DATE
frequency	DESCRIPTION

6. Create a one to many (1-*) aggregation between the Decision struct and the Case struct, and set the aggregation role to decisions. Aggregating the Decision struct in a one to many (1-*) relationship with the Case struct deviates slightly from standard modeling practice, where

the one to many (1-*) relationship would typically be to a DecisionList struct. This in turn would be aggregated in a one to one (1-1) relationship with the Case struct. The rationale behind aggregating Decision directly in a one to many (1-*) relationship with the Case struct is to minimize, as much as possible, the layers inside the JSON representation.

7. Model a new struct, DecisionList, to represent a list of decisions. This struct contains no parameters.
8. Create a one to many (1-*) aggregation between the DecisionList struct and the Decision struct and set the aggregation role to dtls. This struct is not used as part of the resource respresentation but is used in a utility function for retrieving the list of decisions on a case.
9. Model a new struct, Payment, to represent a payment. The Payment struct contains the following parameters, with the relevant domain definitions in the model:

Parameter Name	Domain Definition
nextPayment	CURAM_AMOUNT
lastPayment	CURAM_AMOUNT
nextPayDate	CURAM_DATE
lastPayDate	CURAM_DATE

10. Create a one to one (1-1) aggregation between the Payment struct and the Case struct and set the aggregation role to payments.
11. Model a new struct, called Certificate, to represent a `/asc_cases/{case_id}/certificates` REST resource. The Certificate struct contains the following parameters, with the relevant domain definitions in the model:

Parameter Name	Domain Definition
case_id	CASE_ID
certificate_id	PRACTITIONER_CERTIFICATE_ID
fromDate	CURAM_DATE
toDate	CURAM_DATE
examinationDate	CURAM_DATE
signatureDate	CURAM_DATE
receivedDate	CURAM_DATE
changeReason	EVIDENCE_CHANGE_REASON
practitioner	CASE_PARTICIPANT_ROLE_ID
practitionerName	CONCERN_ROLE_NAME
claimFormSignedInd	CURAM_INDICATOR
householdMember	CASE_PARTICIPANT_ROLE_ID
latestActivity	LOCALIZED_MESSAGE
comments	COMMENTS
versionNo	VERSION_NO

12. Model a new struct, called Diagnosis, to represent a Diagnosis. The Diagnosis struct contains the following parameters, with the relevant domain definitions in the model:

Parameter Name	Domain Definition
diagnosis_id	DIAGNOSIS_ID
examinationDate	CURAM_DATE
receivedDate	CURAM_DATE
evidenceType	CASE_EVIDENCE_TYPE_CODE
changeReason	EVIDENCE_CHANGE_REASON
name	INCAPACITY_DIAGNOSIS
latestActivity	LOCALIZED_MESSAGE

13. Create a one to many (1-*) aggregation between the Diagnosis struct and the Certificate struct and set the aggregation role to diagnoses.
14. Model a new struct, CaseIdentifier. This struct is not part of the resource representation, but it is required for the `asc_cases/{case_id}/certificates/{certificate_id}` GET resource method to represent the case_id query parameter. The CaseIdentifier struct should contain the following parameters:

Parameter Name	Domain Definition
case_id	CASE_ID
_limit	NUMBER_OF_RECORDS

15. Model a new struct, ConcernRoleIdentifier. This struct is not part of the resource representation, but it is required for the `/asc_cases` GET resource method to represent the concern_role_id query parameter. The ConcernRoleIdentifier should contain the following parameters:

Parameter Name	Domain Definition
concern_role_id	CONCERN_ROLE_ID
_limit	NUMBER_OF_RECORDS

16. Model a new struct, PractitionerCertificateKey. This struct is not part of the resources representation, but it is required for the `asc_cases/{case_id}/certificates/{certificate_id}` GET resource method to represent the certificate_id query parameter. The PractitionerCertificateKey should contain the following parameter:

Parameter Name	Domain Definition
certificate_id	PRACTITIONER_CERTIFICATE_ID

Modeling the REST facade

REST API resource methods are implemented by Cúram modeled facade operations.

Before you begin

You must have created the required structs.

About this task

You must create a facade class, with a number of operations to represent the REST API resource methods.

Procedure

1. In the model, create a facade called `AdultSocialCareAPI` with a stereotype of rest.
2. For each entry in the following table, create a new operation on the `AdultSocialCareAPI` facade, with the specified structs defined for input and output:

Facade Operation	Input Struct	Return Struct	Resource Method Mapping	Description
`readASCCases`	ConcernRoleIdentifier	CaseList	*/asc_cases* GET	The ConcernRoleIdentifier struct defines the concern_role_id query parameter that is supported.
`createCertificate`	Certificate	PractitionerCertificateKey	*/asc_cases/{concern_role_id}/certificates* POST	The PractitionerCertificateKey output struct contains the certificate_id of the newly created practitioner certificate.
`readCertificate`	PractitionerCertificateKey	Certificate	*/asc_cases/{concern_role_id}/certificates/{certificate_id}* GET	The PractitionerCertificateKey input struct maps directly to the certificate_id path parameter.
`modifyCertificate`	Certificate	Certificate	*/asc_cases/{concern_role_id}/certificates/{certificate_id}* PUT	The Certificate struct returned contains the updated content.
`deleteCertificate`	PractitionerCertificateKey	Key	*/asc_cases/{concern_role_id}/certificates/{certificate_id}* DELETE	The PractitionerCertificateKey input struct maps directly to the certificate_id path parameter.
`searchCertificates`	CaseHeaderKey	CertificateDetailsList	*/asc_cases/{concern_role_id}/certificates* GET	The CaseIdentifier input struct maps directly to the case_id path parameter.

3. Complete the documentation field for the facade class and all the operations. The documentation is used in the generation of the Swagger document representing the definition of the REST APIs.

What to do next

Model the mandatory properties for each REST resource method.

Modeling the mandatory properties

When you complete the REST API design, you must use IBM® Rational® Software Architect Designer to model the facades, operations and structs.

Procedure

In the model, for each of the operations defined in the `AdultSocialCareAPI`, set the mandatory field values as outlined in the following table:

Facade Operation	Resource Method Mapping	Mandatory Fields Value	Description
`readASCCases`	*/asc_cases* GET	*concern_role_id*	The concern_role_id us used to retrieve the list of cases.
`createCertificate`	*/asc_cases/ {concern_role_id}/ certificates*	*case_id,claimFormSignedInd,practitionerName,receivedDate,signatureDate,ex*	The mandatory fields that are needed to be specified to create a certificate.
`readCertificate`	*/asc_cases/ {concern_role_id}/ certificates/ {certificate_id}*}	*certificate_id*	The certificate_id is used to retrieve the member resource.
`modifyCertificate`		*fromDate,toDate,signatureDate,exa*	The mandatory fields that are needed to be specified to modify a certificate.
`deleteCertificate`		*certificate_id*	The certificate_id is used to retrieve the member resource.
`searchCertificates`		*case_id*	The case_id is used to retrieve the practitioner certificates.

Implementing the REST APIs

After the modeling is complete, you need to generate the Cúram model. Refer to the appendix section for details on how to implement the facade methods that are created for the REST API accelerator.

About this task

The implementation was straightforward due to the Test Driven Development approach that was taken. The entire design and workings of the APIs was driven out by JUnit testing. Implementation entailed moving code from the JUnit tests into the APIs and then updating the tests to call those APIs.

The *asc_cases/{case_id}/certificates* REST API PUT resource was the most complex REST API to implement. The reason for this is that the resource (GET, PUT, POST and DELETE) methods for the diagnosis evidence type needed to be implemented as part of the modify certificate method. This is due to the parent-child nature of the evidence, and the diagnosis evidence is structured as a nested resource of the practitioner certificate.

Related tasks

Analyzing the new REST APIs on page 37
You need to identify any existing APIs within Cúram that can be used to implement the functionality for the new REST APIs.

Appendix on page 57
The following specifies the Cúram APIs that are used for implementing the Cúram REST APIs for the Adult Social Care mobile application.

Generating the Cúram model

When you create a REST API operation in IBM® Rational® Software Architect Designer run the following build targets.

About this task

The `build generated` target generates the code from the model, builds code tables, messages, and events but does not compile handcrafted code. If you run a `build server` target, the `compile.implemented` substep would fail as the handcrafted implementation class is not present. At the very least, you need to provide a dummy implementation of the modeled operations before you run the `build compile.implemented` target.

The `build database` target inserts the security identifiers that are associated with the facade operations into the database.

Procedure

1. From *%CURAM_DIR%\EJBServer*, run: `build generated`.

   %CURAM_DIR% is the Cúram installation directory, which by default is *C:\Merative\Curam\Development*.
2. In Eclipse, you need to refresh and build the EJBServer project. The separate build of the projects in Eclipse is only required if you have the 'Perform build automatically on resource modification' setting switched off. This setting is in the workbench preferences.
3. Open up the modeled implementation class and right click and select **Source** > **Override/Implement Methods** to add the stubs for the modeled operations.
4. From *%CURAM_DIR%\EJBServer*, run: `build compile.implemented database`.

Implementation files for Adult Social Care

The following table includes the location of the classes, which were created to implement the Adult Social Care REST APIs.

Class Name	Location
AdultSocialCareAPI.java	*%CURAM_DIR%\EJBServer\components\AdultSocialCareREST\source\AdultSocialCareREST_src.zip\curam\ascproduct\rest\impl* **Note:** *%CURAM_DIR%* is the Cúram installation directory, which by default is *C:\Merative\Curam\Development*.

The following file includes the constants that are used by the Adult Social Care REST API. They represent the attributes on the following two entities. They are used for the getting and setting the attribute values on the respective dynamic evidence types.

- Practitioner Certificate
- Diagnosis

Class Name	Location
ASCRESTConst.java	*%CURAM_DIR%\EJBServer\components\AdultSocialCareREST\source\AdultSocialCareREST_src.zip\curam\ascproduct\rest\impl*

Configuring the resource configuration files

You must create a REST configuration file in XML format to define the mapping of the REST resource paths to the Cúram façade operations.

Before you begin

Before the REST configuration file can be created, you must model and implement the Cúram façade class, which will have a stereotype of rest.

About this task

The REST configuration file defines all the resource paths for the API, and the methods that are available for each resource. Each resource method is mapped to a Cúram façade operation, and extra configuration values, such as mime-type and cache-control, can be set as required.

Resource configurations specified for Adult Social Care

The following includes the XML and property details, which is specified for the Adult Social Care REST API configurations.

ResourcesConfig.xml

You can view the *ResourcesConfig.xml* and the *RestConfig.properties* files from the *%CURAM_DIR%\EJBServer\components\AdultSocialCare\rest\config* directory. *%CURAM_DIR%* is the Cúram installation directory, which by default is *C:\Merative\Curam\Development*.

The following includes the resource configurations that are specified for theCúram REST API accelerator:

```
<api>
  <version number="v1">

  <resource path="asc_cases">

    <!-- Get all, or the latest, Adult Social Care case record -->
    <method verb="GET">
      <facade class="AdultSocialCareAPI" method="readASCCases" />
    <tags>
      <tag>AdultSocialCare</tag>
    </tags>
    </method>
  </resource>

  <resource path="asc_cases/{case_id}/certificates">

    <!-- Get all, or the latest, Practitioner Certificate records -->
    <method verb="GET">
      <facade class="AdultSocialCareAPI" method="searchCertificates"/>
    <tags>
      <tag>AdultSocialCare</tag>
    </tags>
    </method>

    <!-- Create a Practitioner Certificate record -->
    <method verb="POST">
      <facade class="AdultSocialCareAPI" method="createCertificate"/>
    <tags>
      <tag>AdultSocialCare</tag>
    </tags>
    </method>
  </resource>

  <resource path="asc_cases/{case_id}/certificates/{certificate_id}">

    <!-- Get a specific Practitioner Certificate record -->
    <method verb="GET">
      <facade class="AdultSocialCareAPI" method="readCertificate"/>
    <tags>
      <tag>AdultSocialCare</tag>
    </tags>
    </method>

    <!-- Modify a certificate record -->
    <method verb="PUT">
      <facade class="AdultSocialCareAPI" method="modifyCertificate"/>
    <tags>
      <tag>AdultSocialCare</tag>
    </tags>
    </method>

    <!-- Delete a certificate record -->
    <method verb="DELETE">
      <facade class="AdultSocialCareAPI" method="deleteCertificate"/>
    <tags>
      <tag>AdultSocialCare</tag>
    </tags>
    </method>
  </resource>

  <!-- New resources can be added below -->
  </version>
</api>
```

RestConfig.properties

The following includes the rest configuration properties that are specified for theCúram REST API accelerator:

```
title=Partner Enablement REST API
description=This is the Partner Enablement REST API.
```

Building and testing REST APIs

After you model the facades and create the REST configuration file, you need to generate and test the REST APIs.

Generating the Cúram REST APIs

Complete the following steps for building the Cúram REST APIs and run the API resources in the development environment by using Tomcat to view the Swagger document.

Procedure

1. From the *%CURAM_DIR%* directory, update the *SetEnvironment.bat* and *SetEnvironment.sh* files.

   %CURAM_DIR% is the Cúram installation directory, which by default is *C:\Merative\Curam\Development*.
2. Add the *CATALINA_HOME* environment variable.
   This environment variable defines the home directory of the Tomcat installation, and is used to automatically deploy the REST API resources into Tomcat.
3. From the *%CURAM_DIR%/EJBServer* directory, run the following build target:

   ```
   build rest
   ```

   This build target combines the defined REST resources from each component and deploys the REST API to the Tomcat web server in the development environment.
4. From Eclipse, start your application server and Tomcat as normal.
5. Using a web browser, open the following URL to confirm successful deployment of the REST API:

   ```
   http://localhost:9080/Rest/api/definitions
   ```

   where 9080 is the default Tomcat port.

Results

A list of the Swagger documents for each version of the API defined is displayed. Select a version to open the Swagger document for that version.

Testing a Cúram REST API

A number of different testing mechanisms were used to test the REST APIs, such as unit testing, by using a Swagger-enabled tools, and by using the mobile application with the Cúram application.

Testing a REST API for Adult Social Care

To test that the Cúram REST API accelerator is built successfully, complete the following steps to view the JSON representation for one of the REST APIs.

Procedure

1. From the Adult Social Care accelerator, run through the performing the Adult Social Care caseworker scenario to create an Adult Social Care case for James Smith.
2. From a web browser, enter the following URL to confirm whether the *asc_cases* GET resource method is working. A JSON object with an array of case details is displayed.

```
http://localhost:9080/Rest/v1/asc_cases?concern_role_id=101&_limit=1
```

Results

The following example JSON representation is displayed successfully in the web browser.


```
{
  "data" : [
    {
      "concern_role_id" : "101",
      "case_id" : "8644096534784245760",
      "startDate" : "2015-07-26T23:00:00.000+0000",
      "expectedEndDate" : null,
      "receivedDate" : "2015-07-26T23:00:00.000+0000",
      "registrationDate" : "2015-07-26T23:00:00.000+0000",
      "priorityCode" : {
        "tableName" : "CasePriority",
        "value" : "CP1",
        "description" : "High"
      },
      "integratedCaseType" : {
        "tableName" : "ProductCategory",
        "value" : "PC24000",
        "description" : "Adult Social Care"
      },
      "decisions" : [
        {
          "startDate" : "2015-08-30T23:00:00.000+0000",
          "endDate" : null,
          "amount" : 0.00,
          "frequency" : "",
          "decision" : {
            "tableName" : "CaseDetIntervalResult",
            "value" : "CDIR2",
            "description" : "Not Eligible"
          }
        },
        {
          "startDate" : "2015-07-26T23:00:00.000+0000",
          "endDate" : "2015-08-29T23:00:00.000+0000",
          "amount" : 132.23,
          "frequency" : "Weekly",
          "decision" : {
            "tableName" : "CaseDetIntervalResult",
            "value" : "CDIR1",
            "description" : "Eligible"
          }
        }
      ],
      "payment" : {
        "lastPayment" : 0.00,
        "nextPayment" : 132.23,
        "lastPayDate" : null,
        "nextPayDate" : "2015-07-26T23:00:00.000+0000"
      }
    }
  ]
}
```


Cúram REST API unit testing

Cúram REST resources are HTTP URL endpoints, so you can take a number of different approaches to test your REST APIs.

It is good practice to develop unit tests for REST APIs, and JUnit is one example tool that provides an approach for such testing.

In addition to JUnit, it can be worth considering the following open source libraries to make writing automated tests simpler:

- Jackson for JSON handling
- Apache HTTP Client.

In the absence of these libraries, simple tests can be performed that uses the Java Standard Edition alone, for example:

```
/**
 * Tests the JSON response when the /asc_cases resource is executed for
 * concern_role_id=24011 and the _limit parameter is set to 1.
 *
 * @throws Exception
 */
public void test_json_response() throws Exception {

  final URL url = new URL(
    "http://localhost:9080/Rest/v1/asc_cases?concern_role_id=24011&_limit=1");
  final HttpURLConnection conn = (HttpURLConnection) url.openConnection();

  // Verify the response code is what is expected
  assertEquals(200, conn.getResponseCode());

  final StringBuilder responseBuilder = new StringBuilder();
  final BufferedReader in = new BufferedReader(new InputStreamReader(
    conn.getInputStream()));

  while (in.ready()) {
    responseBuilder.append(in.readLine());
  }

  // Get the expected response from a flat file
  final String responseFileLocation = "C:\\";
  final String responseFile = "Response.txt";
  final String file = responseFileLocation + responseFile;

  final BufferedReader fileBufferedReader = new BufferedReader(
      new FileReader(file));
  final String expectedResponse = fileBufferedReader.readLine();

  // Check if the expected response is what was retrieved
  assertEquals(expectedResponse, responseBuilder.toString());
}
```

Integrating with a mobile application

You can integrate the Cúram REST APIs with other applications, either within your enterprise or with external systems. In this accelerator, the Cúram REST APIs are integrated with a mobile application.

About this task

The Adult Social Care mobile application, which is built by using IBM MobileFirst, was used to integrate with the REST APIs. Adapters were created within IBM MobileFirst, which connects the mobile application with the Cúram application server.

Documentation for how the mobile application was built is out of scope for this accelerator. Steps are included on how to install the mobile application, where you can see yourself how the application was implemented. There are also steps on how to create an adapter to integrate with a Cúram REST API.

Installing IBM MobileFirst development environment

Complete the following steps for installing the Eclipse Marketplace widget and the IBM MobileFirst Studio in Eclipse.

Before you begin

For installing IBM MobileFirst, you must have an existing instance of the Eclipse development environment. You need to install a compatible version of Eclipse such as Juno SR2, Kepler SR1, Kepler SR2, Luna SR1, or Luna SR2.

Procedure

Installing the Eclipse Marketplace widget

Some Eclipse versions come without the Eclipse marketplace widget, which is required to install IBM MobileFirst. Follow these steps to add the Eclipse Marketplace widget only if it does not exist already in your Eclipse.

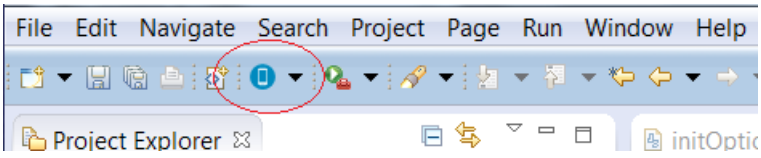
1. From Eclipse, click **Help** > **Install New Software**.
2. From the **Work with** input field, select the drop-down and click **All Available Sites**.
3. In filter text input field type **Marketplace**.
4. Under General Purpose Tools, select **Marketplace Client**.
5. Click **Next** and finish the installation.

Installing MobileFirst Studio in Eclipse

6. From Eclipse, click **Help** > **Eclipse Marketplace**.
7. From the search bar, enter **Mobile First Platform** and click **Go**.
8. Scroll down to IBM MobileFirst Platform Studio and click **Install**.
9. Select **IBM MobileFirst Platform Studio** if it is not already selected and click **Confirm**.
10. **Accept** the terms and conditions, click **Finish**.
11. If you get a security warning click **OK**.
12. Restart Eclipse when prompted.

What to do next

You are able to access the MobileFirst menu from Eclipse. It is denoted by a mobile phone icon.



Installing the Adult Social Care mobile accelerator

The Adult Social Care mobile application accelerator is available for download fis available for download only to members of Merative, such as Lab Services developers who are assigned

to work on a Cúram project on a customer site. Complete the following steps to get the mobile application up and running with IBM MobileFirst.

Procedure

1. Extract the Adult Social Care mobile accelerator at the root of the Cúram installation *%CURAM_DIR%*, which by default is *C:\Merative\Curam\Development*. A folder that is named *Mobile* is added to *%CURAM_DIR%* directory.
   In this documentation, the variable *%MOBILE%* denotes the directory of the mobile application.
2. From Eclipse, click **File** > **Import** > **General** > **Existing Projects into Workspace**.
3. From the **Select root directory** option, click **Browse** and navigate to the *%CURAM_DIR%\Mobile* directory.
4. From the list of available applications, select **ASCMobile** and click **Finish**.

Related concepts

Installing the REST API accelerator on page 21
Download and install the REST API accelerator. The REST API accelerator is available for download only to members of Merative, such as Lab Services developers who are assigned to work on a Cúram project on a customer site.

Installing third-party libraries

The following third-party libraries are required to be installed for the Adult Social Care mobile application user interface.

About this task

For this task, the path *%MOBILE_APPLICATION%* refers to the location: *%CURAM_DIR%\Mobile\ASCMobileApplication\ASCMobile\apps\ASCMobileApplication\common*. *%CURAM_DIR%* is the Cúram installation directory, which by default is *C:\Merative\Curam\Development*.

Table 7: Third-part libraries

The following table lists all third-party libraries that are required and their version numbers.

Library name	Version no.
jQuery UI	1.11.0
Foundation	5.0

Procedure

Installing jQuery UI

1. From a web browser, go to the following URL: JQuery-UI.
2. Select **jQuery UI 1.11.0 (concatenated JS and CSS files)**.
3. Save and extract the file.
4. From the extracted folder, copy the *jquery-ui.js* file to the *%MOBILE_APPLICATION%\js* directory.

5. From the extracted folder, copy the *jquery-ui.css* file to the *%MOBILE_APPLICATION%\css* directory.

Installing Foundation

6. From a web browser, go to the following URL: Foundation.
7. Browse to the download page, and click **Download Everything**.
8. Save the compressed download file.
9. Extract the file to a temporary folder.
10. From the extracted folder, copy the content from the *css* directory to the *%MOBILE_APPLICATION%\css\lib* directory.
11. From the extracted folder, copy the content from the *js* directory to the *%MOBILE_APPLICATION%\js\lib* directory.

Deploying the mobile application

From your IBM MobileFirst development environment, you need to deploy the mobile application and adapters before you can preview the mobile application that uses the simulator. Complete the following steps to view the Adult Social Care mobile application.

Procedure

1. From Eclipse, right-click on the *ASCMobile* directory and click **Open MobileFirst Console**.
   You can ignore the message in red in the Eclipse console, which includes the host name and other console warnings. The Eclipse console displays (in green) **Application 'ASCMobileApplication' with all environments build finished.** Once the deployment is complete, the IBM MobileFirst Platform Operations Console is opened in an internet browser where you can see the mobile application and the adapters deployed.
2. From the *Home>ASCMobile* directory, select **Applications** to view the mobile application deployed.
3. From the *Home>ASCMobile>Applications* directory, select **ASCMobileApplication**.
4. From the **Common Resources** section, select **Preview** to view the mobile application through the IBM MobileFirst web simulation environment.

Performing the Adult Social Care mobile application caseworker scenario

Detailed steps are provided for you to walk through the sample mobile application, and experience the full end-to-end business work flow.

Before you begin

You need to install the Adult Social Care mobile application accelerator and start your Cúram application as normal.

Related tasks

Installing the Adult Social Care mobile accelerator on page 51

The Adult Social Care mobile application accelerator is available for download fis available for download only to members of Merative, such as Lab Services developers who are assigned to work on a Cúram project on a customer site. Complete the following steps to get the mobile application up and running with IBM MobileFirst.

Searching for a client

Complete the following steps to find a client who is already registered as a person on the system.

Procedure

1. From the mobile application landing page, enter the client's full name in the search for a client field and click **Search**.

   Table 8: Search for a client value

Name	Value
Full Name	James Smith

2. From the **Search Results** page, if the search criteria is met, click on the person link to open the **Client Profile** page.

Editing a practitioner certificate

Complete the following steps to edit the practitioner certificate evidence record.

Procedure

1. From the **Client Profile** window, click **Edit**.
2. From the **Edit Practitioner Certificate** window, modify the following details and click **Save**.

Name	Description	Value
Change Reason	The reason why the evidence needs to be changed.	Reported by Client.
Specified Certification To Date	The last day that is covered by the practitioner certification.	Current date + 8 weeks.
Comments	Comments that are captured by the caseworker.	Extended the certification by 6 months due to new diagnosis.

What to do next

You can check the Adult Social Care eligibility worker Cúram workspace to see your changes.

Creating a diagnosis

Complete the following steps to create a diagnosis that is associated with the practitioner certificate.

Procedure

1. From the **Client Profile** window, click **New Diagnosis**.
2. From the **New Diagnosis** window, enter the following values and click **Save**.

Name	Description	Value
Received Date	The date that the practitioner certificate evidence was received by the organization.	Current date
Examination Date	The date on which the medical examination took place.	Current date
Diagnosis Type	The code for the physical disorder or illness the claimant is suffering.	Cardiovascular disease

What to do next

You can check the Adult Social Care eligibility worker Cúram workspace to see your changes.

Integrating the mobile application with a Cúram REST API

You can integrate an existing Cúram REST API with an IBM MobileFirst application by using a IBM MobileFirst adapter. Complete the following steps for creating an adapter to integrate with a REST API and connect with the Cúram application server. The following tasks describe how to integrate a GET and PUT Cúram REST request with an IBM MobileFirst application and discusses how to handle date-time formats between Cúram and the mobile application.

Related information

Overview of MobileFirst adapters

Integrating a custom GET request with a mobile application

The following task describes how to integrate a custom Cúram GET request with an IBM MobileFirst mobile application. For the purposes of this task, a custom GET request from the REST API accelerator is used as an example.

Procedure

1. From Eclipse, open your IBM MobileFirst application and create a new IBM MobileFirst HTTP adapter. For more information, see Creating an IBM MobileFirst adapter.
2. From the new adapters XML file, create a procedure that is called `getEvidenceByCaseID` as follows:

```
<procedure name="getEvidenceByCaseID"/>
```

3. From the new adapters JavaScript file, implement the `getEvidenceByCaseID` function to retrieve data from the existing Cúram REST API as follows:

```
function getEvidenceByCaseID(case_id) {

 var input = {
     method : 'get',
     returnedContentType : 'json',
     path : '/Rest/v1/asc_cases/' + case_id + '/certificates?_limit=1',
 };

 return WL.Server.invokeHttp(input);
}
```

4. From the adapter's XML file, enter the following port and domain details to point to the Cúram application server, where `9080` is the default Tomcat port.

Table 9: Adapter connection settings

Port	Domain	Description
9080	*localhost*	Using the default port 9080 on the localhost domain.

5. From the IBM MobileFirst project, right-click the adapter's folder, click **Run as** > **Call MobileFirst Adapter** to test the adapter.
6. From the adapter wizard window, select the `getEvidenceByCaseID` function and enter the following parameter details:

Table 10: Adapter parameter settings

Parameter Name	Parameter Value Description
case_id	Enter the *case_id* for the case, which is associated with the evidence to be retrieved.

7. Click **Finish**.

Integrating a custom PUT request with a mobile application

The following task describes how to integrate a custom Cúram PUT request with an IBM MobileFirst mobile application. For the purposes of this task, a PUT request from the Cúram REST API accelerator is used as an example.

Procedure

1. From Eclipse, open your MobileFirst application and create a new MobileFirst HTTP adapter.

   For more information, see Creating a MobileFirst adapter.

2. From the new adapters XML file, create a procedure that is called `putEvidence` as follows:

```
<procedure name="putEvidence"/>
```

3. From the new adapters JavaScript file, implement the `putEvidence` function to send data by using the existing Cúram REST API as follows:

   For more information about sending REST requests to Cúram, see the 'Request Header' documentation.

```
function putEvidence(certificate_id, case_id, data) {
 var input = {
   method : 'put',
   returnedContentType : 'json',
   path : '/Rest/v1/asc_cases/' + case_id + '/certificates/' + certificate_id,
              //Cúram PUT, POST and DELETE requests require a 'Referer' header to
 be included in all requests.
              //The Referer header should contain the value 'curam://' followed by
 an identifier for the application.
   headers : { 'Referer': 'curam://acceleratorApp' },
   body: {
   contentType: 'application/json; charset=utf-8',
   content: data,
   }
 };
 return WL.Server.invokeHttp(input);
}
```

4. From the adapters XML file, enter the following port and domain details to point to the Cúram application server.

   Table 11: Adapter connection settings

Port	Domain	Description
9080	*localhost*	Using the default port 9080 on the localhost domain.

5. From the MobileFirst project, right-click the adapter's folder, click **Run as** > **Call MobileFirst Adapter** to test the adapter.

6. From the adapter wizard window, select the `putEvidence` function and enter the following parameter details:

   Table 12: Adapter parameter settings

Parameter Name	Parameter Value Description
certificate_id	Enter the *certificate_id* for the evidence to be modified.
case_id	Enter the *case_id* for the case, which is associated with the evidence.
evidence	JSON representation for the evidence that is to be modified.

7. Click **Finish**.

   For more information about request headers, see the *Cúram™ REST API Guide*.

The Cúram date-time format

The date-time format in Cúram supports the ISO 8601 format. The ISO 8601 format is not always suitable for consumption by the user interface of an application.

You need to write your own functions to control the format of any date-time information that is being retrieved or sent as an outgoing request for your application.

For more information about the date-time format, see the *Cúram™ REST API Guide*.

1.5 Appendix

The following specifies the Cúram APIs that are used for implementing the Cúram REST APIs for the Adult Social Care mobile application.

Retrieving the case details Cúram APIs

For the Adult Social Care mobile application, you are required to get the case, decision, and payment details. The REST API, `/asc_cases`, returns all this information. The following sample code defines the existing Cúram APIs used to retrieve the case, decision, and payment details.

About this task

The following include the list of attributes that are required to be retrieved for the client and displayed on the mobile application.

Case Attributes	Decision Attributes	Payment Attributes
• *startDate* • *expectedEndDate* • *dateReceived* • *priorityCode*	List of decision details: • *startDate* • *endDate* • *amount* • *decision* • *frequency*	• *nextPayment* • *nextPaymentDate* • *lastPayment* • *lastPayDate*

Procedure

1. The following API is used to retrieve all the cases for a person.

```
final CaseHeaderConcernRoleDetailsList1 list =
  MaintainCaseFactory.newInstance()
    .getCasesByConcernRoleID1(participantKey);
```

2. For the Adult Social Care mobile application, you are only interested in the Adult Social Care integrated Cases and the Incapability Benefit product deliveries. You need to iterate through the list and filter it to retrieve the integrated case of type Adult Social Care and the product delivery cases of type Incapability Benefit.
3. For the mobile application, there is a requirement to retrieve the latest Adult Social Care case, therefore you need to sort the list by start date with a descending order.
4. Iterate through the sorted list to process the number of cases the *_limit* input parameter allows. Within the iteration, you need to call the following APIs to read the case details, retrieve the list of decisions, and payment details for the associated product delivery.

Table 13: APIs to get the case, decision, and payment details

API	Description
`final CaseHeaderDtls icCaseHeaderDtls = CachedCaseHeaderFactory.newInstance() .read(caseHeaderKey);`	This API is used to read the case details. • *receivedDate* • *expectedEndDate* • *priorityCode* • *registrationDate*
`final CaseDeterminationDecisionDetailsList list = CaseDeterminationFactory.newInstance() .listDecisionPeriodsForDetermination(key);`	This API is used to retrieve a list of case decision details for a specified case. Iterate through the list to assign the decision values, which are added to the returned list of decisions. • *startDate* • *endDate* • *decision* • *amount* • *frequency*
`final SimulateInd simulateInd = new SimulateInd(); simulateInd.simulateInd = false; simulatePaymentResult = SimulatePaymentFactory.newInstance() .simulatePayment(caseIDStruct, simulationDate, simulateInd);`	This API is used to get the details of the next payment, including amount and next payment date. The simulate indicator is set to false as you do not want to save this simulation. • *nextPayment* • *nextPayDate*
`final AmountEffectiveDate lastPaymentDetails = InstructionLineItemFactory.newInstance() .readLastPaymentAndEffectiveDate (iLICaseStatusAndTypeKey);`	This API is used to get the details for the last payment, including the amount and payment date. • *lastPayment* • *lastPayDate*

The following include the resource URLs that can be used to retrieve either the list of cases or a specific number of cases.

Table 14: Resource URLs

Resource URL	Description
`/asc_cases`	Retrieves a list of Adult Social Care case records.
`/asc_cases?concern_role_id=<concern_role_id>&_limit=1`	Retrieves the latest Adult Social Care case record.

Dynamic evidence Cúram APIs

The following include the APIs used to create, retrieve, update, and delete dynamic evidence details that are configured for the Adult Social Care mobile application.

Retrieving the parent-child evidence details

The `/asc_cases/{case_id}/certificates/{certificate_id}` REST API GET resource retrieves the practitioner certificate (parent evidence) and a list of associated diagnosis (child evidence) details. The following Cúram APIs can be used in the implementation of this API.

About this task

The following include the list of practitioner certificate and diagnosis attributes that are needed for display on the mobile application.

Certificate Attributes	Diagnosis Attributes
• *fromDate* • *toDate* • *practitionerName* • *certificateType* • *examinationDate* • *signatureDate* • *claimFormSignedInd* • *updatedOn* • *updatedBy* • *receivedDate* • *changeReason* • *comments* • *versionNo*	List of diagnoses details: • *name* • *examinationDate* • *updatedOn* • *updatedBy* • *receivedDate* • *evidenceType* • *changeReason*

Procedure

1. The following Cúram APIs are needed to read the dynamic evidence details.

API	Description
`final EvidenceServiceInterface esl = EvidenceGenericSLFactory.instance(key.certificate_id);`	This API is used to get the evidence service interface, which is used to read the evidence details.
`final EvidenceCaseKey evKey = new EvidenceCaseKey(); evKey.evidenceKey.evidenceID = key.certificate_id; evKey.evidenceKey.evType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE; readEvidenceDetails = esl.readEvidence(evKey);`	This API is used to read the practitioner certificate evidence details based on the dynamic evidence type and certificate identifier.

2. You need to convert the dynamic evidence attributes that are returned into their respective Cúram types and assign to the practitioner certificate details.

Sample code	Description
`certificate.fromDate = (Date) DynamicEvidenceTypeConverter .convert(dynamicEvidence .getAttribute(ASCConst.kCertFromDate));`	You need to get the following practitioner certificate evidence attributes values. • *certificate_id* • *fromDate* • *toDate* • *practitionerName* • *certificateType* • *examinationDate* • *signatureDate* • *claimFormSignedInd* • *comments*
`certificate.versionNo = dynamicEvidence .getVersionNo();`	You can retrieve the version number for the evidence that is retrieved from the `readEvidence` REST API. • *versionNo*

3. The following Cúram APIs are used to retrieve the remaining practitioner certificate details.

API	Description
`final EvidenceDescriptorDtls descriptor = EvidenceDescriptorFactory.newInstance() .readByRelatedIDAndType(relatedIDAndEvidenceTypeKey); certificate.receivedDate = descriptor.receivedDate; certificate.changeReason = descriptor.changeReason; certificate.case_id = descriptor.caseID;`	This API is used to read the *receivedDate* and *changeReason* attributes, which are not stored on the practitioner certificate dynamic evidence entity. • *receivedDate* • *changeReason* • *case_id*
`final EvidenceDescriptorKey descriptorKey = new EvidenceDescriptorKey(); descriptorKey.evidenceDescriptorID = descriptor.evidenceDescriptorID; final EvidenceChangeHistoryDtlsList historyList = EvidenceChangeHistoryFactory.newInstance() .searchInstanceHistoryByEvidenceDescriptorID(descriptorKey); certificate.updatedBy = getUpdatedBy(historyList.dtls.item(0)); certificate.updatedOn = historyList.dtls.item(0).changeDateTime;`	The `readUpdatedByAndUpdatedOn` protected method is used to read the *updatedBy* and *updatedOn* attributes, which are not stored on the practitioner certificate dynamic evidence entity. The *updatedBy* uses the `getUpdatedBy` utility to retrieve the full name of the user who last updated the evidence. • *updatedOn* • *updatedBy*

4. A `readChildEvidence` protected method is used to read the list of child diagnosis evidence records for its associated parent practitioner certificate evidence. The following sample code is used to get the list of diagnosis details.

```
final EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
eiEvidenceKey.evidenceID = key.certificate_id;
eiEvidenceKey.evidenceType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE;
final ChildList childList =
  EvidenceRelationshipFactory.newInstance()
    .getChildKeyList(eiEvidenceKey);
```

5. You need to loop through the returned list and read the diagnosis details. A `readDiagnosis` protected method is used to

API	Description
`final EvidenceServiceInterface esl = EvidenceGenericSLFactory .instance(key.evidenceKey.evidenceID);`	This API is used to get the evidence service interface, which is used to read the evidence details.
`final EvidenceCaseKey evKey = new EvidenceCaseKey(); evKey.evidenceKey.evidenceID = key.evidenceKey.evidenceID; evKey.evidenceKey.evType = CASEEVIDENCE.ASCDIAGNOSIS; final ReadEvidenceDetails readEvidenceDetails = esl.readEvidence(evKey); final DynamicEvidenceDataDetails dynamicData = readEvidenceDetails.dtls;`	This API is used to read the practitioner certificate evidence details based on the dynamic evidence type and certificate identifier. The `readEvidence` API performs a read on the *DYNAMICEVIDENCEDATA* and the *DYNAMICEVIDENCEDATATTRIBUTE* entities.

6. You need to convert the dynamic evidence attributes that are returned into their respective Cúram types and assign to the diagnosis details.

Sample code	Description

`diagnosis.name = dynamicData`
 	
`.getAttribute(ASCConst.kDiagnosisName).getValue();`
`diagnosis.examinationDate =`
`(Date) DynamicEvidenceTypeConverter`
`.convert(dynamicData`
 	
`.getAttribute(ASCConst.kCertExaminationDate));`
	You need to get the following diagnosis evidence attributes values.
	• *name*
	• *examinationDate*

`diagnosis.receivedDate =`
 	
`readEvidenceDetails.descriptor.receivedDate;`
`diagnosis.changeReason =`
 	
`readEvidenceDetails.descriptor.changeReason;`
`diagnosis.versionNo =`
 	
`readEvidenceDetails.dtls.getVersionNo();`
`diagnosis.diagnosis_id =`
`readEvidenceDetails.dtls.getID();`
	Set the following attributes with the details returned from the *readEvidenceDetails*.
	• *receivedDate*
	• *changeReason*
	• *versionNo*
	• *diagnosis_id*

`final EvidenceDescriptorKey`
`descriptorKey =`
`new EvidenceDescriptorKey();`
`descriptorKey.evidenceDescriptorID =`
 	
`readEvidenceDetails.descriptor.evidenceDescriptorID;`
 	
`final EvidenceChangeHistoryDtlsList`
`historyList =`
`EvidenceChangeHistoryFactory`
`.newInstance()`
 	
`.searchInstanceHistoryByEvidenceDescriptorID(descriptorKey);`
 	
`diagnosis.updatedBy =`
 	
`getUpdatedBy(historyList.dtls.item(0));`
`diagnosis.updatedOn =`
 	
`historyList.dtls.item(0).changeDateTime;`
	The `readUpdatedByAndUpdatedOn` protected method is used to read the *updatedBy* and *updatedOn* attributes, which are not stored on the practitioner certificate dynamic evidence entity. The *updatedBy* uses the `getUpdatedBy` utility to retrieve the full name of the user who last updated the evidence.
	• *updatedOn*
	• *updatedBy*

7. Add the diagnosis details to the certificate, which is being returned as part of the REST API.

Retrieving all the parent-child evidence details

The `/asc_cases/{case_id}/certificates` REST API GET resource retrieves the list of practitioner certificate and associated diagnosis details. The steps describe the logic, which

is created for the Adult Social Care mobile application and the sample code for the Cúram API, which can be used for this API.

Procedure

1. This API is used to read the list of evidence descriptor details for the case. The evidence descriptor contains all the information across all evidence types.

```
final ListAllActiveEVDInstanceWorkspaceDtls list = EvidenceFactory.newInstance()
  .listAllActiveEVDInstanceWorkspaceDtls(searchKey);
```

2. You need to iterate through the evidence descriptor list to retrieve the list of practitioner certificate details. Within the iteration, you are calling the `readCertificate`method to get the certificate details. For more information, see Retrieving the parent-child evidence details on page 60.
3. For the mobile application, there is a requirement to retrieve the latest practitioner certificate details. Therefore, you need to sort the list of practitioner certificate details by start date with a descending order.
4. Iterate through the sorted list to get the latest practitioner certificate details. Iterate through the sorted list and return the number of records according to the *_limit* parameter. If *_limit* is set to zero, then all records are returned otherwise the specified number of records are returned. The mobile application specifies the input parameter as *_limit=1* as the requirement is to get the latest record. The following specified example resource URLs:

Table 15: Resource URLs

Resource URL	Description
asc_cases/{concern_role_id}/certificates	Retrieves a list of practitioner certificate records.
asc_cases/{concern_role_id}/certificates?_limit=1	Retrieves the latest practitioner certificate record. If the *_limit=2*, retrieves the latest 2 records from the list.

Creating a practitioner certificate

The *asc_cases/{case_id}/certificates* REST API POST resource is used to create a practitioner certificate record.

Procedure

1. The following sample code is needed to map the certificate for modifying the evidence.

Sample code	Description
`final GenericSLDataDetails genericDtls = mapCertificateDetails(details);`	The `mapCertificateDetails` is a protected method to map the certificate details to the evidence details. For more information, see Mapping the practitioner certificate evidence details on page 68.

2. The following Cúram APIs are called to create the evidence record.

API	Description
`final EvidenceTypeKey evType = new EvidenceTypeKey(); evType.evidenceType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE; final EvidenceServiceInterface evidenceServiceInterface = EvidenceGenericSLFactory.instance (evType, details.receivedDate);`	This API is used to get the evidence service interface, which is used to create evidence.
`// Set the genericDtls to create the evidence final GenericSLDataDetails genericDtls = new GenericSLDataDetails(); genericDtls.setData(evidence); genericDtls.setDescriptor(descriptor); genericDtls.setCaseIdKey(descriptor.caseID); // Create the evidence record final ReturnEvidenceDetails evidenceDetails = evidenceServiceInterface.createEvidence(genericDtls);`	This API is used to create the evidence record, which passes in the evidence and descriptor details.

Updating a practitioner certificate

The *asc_cases/{case_id}/certificates* REST API PUT resource is created to modify a practitioner certificate record. As the diagnosis is a child of the practitioner certificate, you need to create, update, or delete a diagnosis record as part of this REST API.

Procedure

1. The following sample code is needed to map the certificate for modifying the evidence.

Sample code	Description
`final GenericSLDataDetails genericDtls = mapCertificateDetails(details);`	The `mapCertificateDetails` is a protected method to map the certificate details to the evidence details. For more information, see Mapping the practitioner certificate evidence details on page 68.

2. Checks are done to see whether the user is trying to create, update, or delete a diagnosis record.

Sample code	Description
`createUpdateDeleteDiagnosis(details);`	The `createUpdateDeleteDiagnosis` protected method checks to see whether the user is trying to create, update, or delete a diagnosis record. For more information, see Creating, updating, or deleting a child evidence record on page 68.

3. The following Cúram APIs are used to update the dynamic evidence details.

API	Description
`final EvidenceTypeKey evType = new EvidenceTypeKey(); evType.evidenceType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE; final EvidenceServiceInterface evidenceServiceInterface = EvidenceGenericSLFactory.instance (evType, details.receivedDate);`	This API is used to get the evidence service interface, which is used to modify the evidence details.
`final ReturnEvidenceDetails evidenceDetails = evidenceServiceInterface.modifyEvidence(genericDtls);`	This API is used to modify the practitioner certificate evidence details.
`return readCertificate(certKey);`	Return the practitioner certificate details, which are returned from the `readCertificate` method. For more information, see Retrieving the parent-child evidence details on page 60.

Mapping the practitioner certificate evidence details

Before you create or update evidence, you are required to convert the details in to evidence specific format. This sample code includes the mapping details that is required for creating and updating a practitioner certificate record.

Sample code	Description
`final String practitionerName = certDetails.practitionerName; evidence.getAttribute(ASCConst.kPractitionerName) .setValue(practitionerName);`	Set the practitioner certificate evidence attributes. • *practitionerName* • *fromDate* • *toDate* • *certificateType* • *examinationDate* • *signatureDate* • *claimFormSignedInd* • *comments*
`evidence.setID(certDetails.certificate_id); evidence.setVersionNo(certDetails.versionNo);`	Set the *ID* and *versionNo* as they are needed for modifying evidence.
`final EvidenceDescriptorDetails descriptor = new EvidenceDescriptorDetails(); descriptor.evidenceType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE; descriptor.receivedDate = certDetails.receivedDate; descriptor.caseID = certDetails.case_id; descriptor.relatedID = certDetails.certificate_id; descriptor.changeReason = certDetails.changeReason;`	Set the evidence descriptor details.
`final GenericSLDataDetails genericDtls = new GenericSLDataDetails(); genericDtls.setData(evidence); genericDtls.setDescriptor(descriptor); genericDtls.setCaseIdKey(evidenceCaseKey.caseIDKey); genericDtls.setEvidenceCaseKey(evidenceCaseKey);`	Set the *genericDtls*, which is needed as an input parameter for modifying the evidence.

Creating, updating, or deleting a child evidence record

As the diagnosis evidence is a nested array in the resource representation of the practitioner certificate evidence. The diagnosis creation, update, and delete operations need to be done as part of the *`asc_cases/{case_id}/certificates/{certificate_id}`* REST API PUT resource. As a result, logic is required to understand which action the user is trying to perform. This logic is described here.

Procedure

1. Add the following check to see whether the user wants to create a diagnosis record. Loop through the diagnosis list that is passed in to check to see whether the *diagnosis_id* equals zero. If so, you need to create the diagnosis record. The following includes sample code for creating a diagnosis record.

Table 16: Create diagnosis

Sample code	Description
`final GenericSLDataDetails genericDtls = mapDiagnosisDetails(diagnosis, caseKey, certKey);`	This API is used to map the details that are passed in to dynamic evidence data details format. For more information, see Mapping the diagnosis details on page 73.

`final EvidenceTypeKey evType = new EvidenceTypeKey();`
`evType.evidenceType = CASEEVIDENCE.ASCDIAGNOSIS;`
 	
`final EvidenceServiceInterface evidenceServiceInterface =`
 	
`EvidenceGenericSLFactory.instance(evType, diagnosis.receivedDate);`
	This API is used to get the evidence service interface, which is used to create evidence.

`// Create the evidence record`
`final ReturnEvidenceDetails evidenceDetails =`
 	
`evidenceServiceInterface.createEvidence(genericDtls);`
 	
`// Return the key`
`diagnosisKey.diagnosis_id =`
 	
`evidenceDetails.evidenceKey.evidenceID;`
	The `createEvidence` method is used to create the evidence record, which passes in the evidence and descriptor details.

2. Add the following check to see whether the user wants to delete a diagnosis record.

Sample code	Description

`// Checks to see if a diagnosis record`
`// needs to be deleted`
`final List<Long> diagnosisKeys =`
`  new ArrayList<Long>();`
 	
`// Loop through the diagnosis details`
`// brought in, read the child evidence`
`keys`
`for (final Diagnosis diagnosis :`
`details.diagnoses) {`
`  // Add the diagnosis keys to the`
`array list`
 	
`diagnosisKeys.add(diagnosis.diagnosis_id);`
`}`
	Loop through the diagnosis details that are passed in, read the child evidence keys. Add the child evidence keys to an array list, which is used further on to check whether a diagnosis record does not exist for the practitioner certificate identifier that is passed into the REST API. Therefore, you delete the record.

`// Retrieve the child records for`
`// this certificate`
`final EIEvidenceKey eiEvidenceKey =`
`  new EIEvidenceKey();`
`eiEvidenceKey.evidenceID =`
`  details.certificate_id;`
`eiEvidenceKey.evidenceType =`
 	
`CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE;`
`final ChildList childList =`
 	
`EvidenceRelationshipFactory.newInstance()`
`  .getChildKeyList(eiEvidenceKey);`
	Retrieve the list of child diagnosis records for the parent certificate.

3. Loop through the child list.
   a) Check to see whether the user is trying to delete a diagnosis record.

Table 17: Delete diagnosis

Sample code	Description

`// If a diagnosis record does not exist`
`// being passed in delete the record.`
`if (!diagnosisKeys.contains(childList.list.dtls.item(i).childID))`
`{`
`final DiagnosisKey key = new DiagnosisKey();`
`key.diagnosis_id = childList.list.dtls.item(i).childID;`
`// remove the child record`
`deleteDiagnosis(key);`
`}`
	If a diagnosis record does not exist which is being passed in, then delete the diagnosis record.

The following sample code is from the `deleteDiagnosis` protected method:
`final EvidenceDescriptorKey evKey = new EvidenceDescriptorKey();`
`final RelatedIDAndEvidenceTypeKey relatedKey = new RelatedIDAndEvidenceTypeKey();`
`relatedKey.evidenceType = CASEEVIDENCE.ASCDIAGNOSIS;`
`relatedKey.relatedID = key.diagnosis_id;`
`final EvidenceDescriptorDtls descriptor = EvidenceControllerFactory.newInstance().readEvidenceDescriptorByRelatedIDAndType(relatedKey);`
`evKey.evidenceDescriptorID = descriptor.evidenceDescriptorID;`
`// Remove the diagnosis evidence record`
`EvidenceControllerFactory.newInstance().removeEvidence(evKey);`
	The `removeEvidence` Cúram API is used to remove the evidence record and added an **InEdit** status.

`// Apply Changes`
`final curam.core.sl.infrastructure.struct.EvidenceKey evidenceKey = new curam.core.sl.infrastructure.struct.EvidenceKey();`
`evidenceKey.evidenceID = relatedKey.relatedID;`
`evidenceKey.evidenceType = CASEEVIDENCE.ASCDIAGNOSIS;`
`evidenceKey.evidenceDescriptorID = descriptor.evidenceDescriptorID;`
`evidenceKey.correctionSetID = descriptor.correctionSetID;`
`evidenceKey.successionID = descriptor.successionID;`
`EvidenceControllerFactory.newInstance().applyRemoval(evidenceKey);`
	The `applyRemoval` Cúram API is used to apply the deletion.

a) Else check to see whether the user wants to modify a diagnosis record.

Table 18: Modify diagnosis

Sample code	Description

`final CaseKey caseKey = new CaseKey(); caseKey.caseID = details.case_id;`
 	
`final Diagnosis diagnosis = new Diagnosis();`
 	
`final EvidenceServiceInterface esl = EvidenceGenericSLFactory`
 	
`.instance(childList.list.dtls.item(i).childID);`
`final EvidenceCaseKey evKey = new EvidenceCaseKey();`
 	
`evKey.evidenceKey.evidenceID = childList.list.dtls.item(i).childID;`
`evKey.evidenceKey.evType = CASEEVIDENCE.ASCDIAGNOSIS;`
 	
`final ReadEvidenceDetails readEvidenceDetails = esl.readEvidence(evKey);`
`final DynamicEvidenceDataDetails originalData = readEvidenceDetails.dtls;`
	From looping though the child list of evidence record keys, read the evidence details from the database to get the original evidence details.

`// Loop through the modified data being passed in`
`for (final Diagnosis updatedData : details.diagnoses) {`
 	
`final GenericSLDataDetails result = mapDiagnosisDetails(updatedData, caseKey);`
 	
`// if any incoming attribute has changed value,`
`// an update is required`
`final boolean updateRequired = isUpdateRequired(originalData, result.getData());`
 	
`if (updateRequired) {`
 	
`final Diagnosis returnedDiagnosisDetails = modifyDiagnosis(updatedData, caseKey, new PractitionerCertificateKey());;`
`}`
	Loop through the diagnosis details, which are being passed in. The details that are passed in need to be in mapped to they dynamic evidence data details format.
	
	For more information, see Mapping the diagnosis details on page 73.
	
	An `isUpdateRequired` utility method checks to see whether there is a difference between the original details that are retrieved from the database and the details, which are being passed in. If there is a difference, the `modifyDiagnosis` method is called.

Mapping the diagnosis details

Before you create or update evidence, you are required to convert the details in to evidence specific format. This sample code includes the mapping details that is required for creating and updating a diagnosis record.

Sample code	Description
`final DynamicEvidenceDataDetails evidence = DynamicEvidenceDataDetailsFactory.newInstance(CASEEVIDENCE.ASCDIAGNOSIS, diagnosis.receivedDate);`	This API is used to get the evidence details by type and effective date as there can be multiple versions available

`evidence.getAttribute(ASCConst.kDiagnosisName) .setValue(diagnosis.name);`
 	
`final String examinationDate = formatDate(diagnosis.examinationDate);`
`evidence.getAttribute(ASCConst.kDiagnosisExaminationDate) .setValue(examinationDate);`
	The following diagnosis dynamic evidence attributes need to be set.
	• *name*
	• *examinationDate*

`// Set the parent evidence key`
`final EvidenceKey evKey = new EvidenceKey();`
`evKey.evidenceID = certKey.certificate_id;`
`evKey.evType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE;`
 	
`// Set the genericDtls to create the evidence`
`final GenericSLDataDetails genericDtls = new GenericSLDataDetails();`
`genericDtls.setData(evidence);`
`genericDtls.setDescriptor(descriptor);`
`genericDtls.setCaseIdKey(caseID.caseID);`
`genericDtls.addParent(CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE, evKey);`
	Before you create the evidence, you need to set the evidence descriptor details.

Deleting a practitioner certificate

The *asc_cases/{case_id}/certificates* REST API PUT resource deletes a practitioner certificate record and its associated diagnosis records.

About this task

For the Adult Social Care mobile application, the requirement is to delete a practitioner certificate record and its associated child diagnosis records. In order for you to delete the practitioner certificate record, you are required to delete the associated diagnosis evidence records first or else an error message is displayed. For better user experience, the REST resource DELETE method includes the deletion of the child records first before the deletion of the practitioner certificate.

Procedure

1. The following APIs are used to remove the child diagnosis evidence records associated with the practitioner certificate evidence. This functionality was separated out into its own protected method that is called `removeChildEvidences`.

API	Description

`// Retrieve the child records for this certificate`
`final EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();`
`eiEvidenceKey.evidenceID = certKey.certificate_id;`
`eiEvidenceKey.evidenceType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE;`
`final ChildList childList = EvidenceRelationshipFactory.newInstance().getChildKeyList(eiEvidenceKey);`
	This API is used to get the list of child evidence records.

2. You need to loop through the *childList* and call the following Cúram APIs to remove the child evidence:

API	Description

`final RelatedIDAndEvidenceTypeKey relatedKey = new RelatedIDAndEvidenceTypeKey();`
`relatedKey.relatedID = child.childID;`
`relatedKey.evidenceType = child.childType;`
`final EvidenceDescriptorDtls descriptor = EvidenceControllerFactory.newInstance().readEvidenceDescriptorByRelatedIDAndType(relatedKey);`
	This API is used to read the evidence descriptor unique identifier, which is needed to delete an evidence record.

`final EvidenceDescriptorKey descriptorKey = new EvidenceDescriptorKey();`
`descriptorKey.evidenceDescriptorID = descriptor.evidenceDescriptorID;`
`EvidenceControllerFactory.newInstance().removeEvidence(descriptorKey);`
	This API is used to remove an evidence record.

`final EvidenceKey evKey = new EvidenceKey();`
`evKey.evidenceDescriptorID = descriptor.evidenceDescriptorID;`
`evKey.evidenceType = descriptor.evidenceType;`
`evKey.evidenceID = relatedKey.relatedID;`
`evKey.successionID = descriptor.successionID;`
`evKey.correctionSetID = descriptor.correctionSetID;`
`EvidenceControllerFactory.newInstance().applyRemoval(evKey);`
	This API is needed to apply the evidence removal changes.

3. Once the child evidence is created, you can now delete the parent evidence record. The following Cúram APIs are used to delete the practitioner certificate record.

API	Description

`final RelatedIDAndEvidenceTypeKey relatedKey = new RelatedIDAndEvidenceTypeKey(); relatedKey.evidenceType = CASEEVIDENCE.ASCPRACTITIONERCERTIFICATE; relatedKey.relatedID = key.certificate_id;`
 	
`final EvidenceDescriptorDtls descriptor =`
 	
`EvidenceControllerFactory.newInstance()`
 	
`.readEvidenceDescriptorByRelatedIDAndType( relatedKey);`
	This API is used to read the evidence descriptor unique identifier, which is needed to delete an evidence record.

`final EvidenceDescriptorKey evDescriptorKey = new EvidenceDescriptorKey(); evDescriptorKey.evidenceDescriptorID = descriptor.evidenceDescriptorID;`
 	
`EvidenceControllerFactory.newInstance().removeEvidence(evDescriptorKey);`
	This API is used to remove an evidence record.

`final EvidenceKey evKey = new EvidenceKey(); evKey.evidenceDescriptorID = descriptor.evidenceDescriptorID; evKey.evidenceType = descriptor.evidenceType; evKey.evidenceID = relatedKey.relatedID; evKey.successionID = descriptor.successionID; evKey.correctionSetID = descriptor.correctionSetID; EvidenceControllerFactory.newInstance().applyRemoval(evKey);`	This API is used to apply the evidence removal changes.

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

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

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those

websites are not part of the materials for this Merative product and use of those websites is at your own risk.

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

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

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

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.