Cúram 8.1.2

Using the Data Mapping Engine Guide

Note

Before using this information and the product it supports, read the information in Notices on page 39

Edition

This edition applies to Cúram 8.1, 8.1.1, and 8.1.2.

Contents

Note.. iii

Edition... v

1 Developing with the Data Mapping Engine..9

1.1 Data mapping overview..9

Data mapping environment and process.. 9

How data is stored in the datastore...10

Creating logical maps...12

1.2 Writing mapping specifications for static evidence and PDFs...12

Writing mapping specifications...12

Mapping condition expressions... 13

Mapping code table values...14

Mapping to multiple target entities..14

Matching one parent entity to several child entities..15

Matching patterns and following associations in the datastore.................................15

Mapping household member data...17

1.3 Writing mapping configurations for static evidence and PDFs..17

Writing mapping configurations for the evidence application builder...................................... 18

Writing mapping configurations for the PDF application builder... 21

1.4 Configuring PDF application forms.. 25

1.5 Writing mapping specifications and configurations for dynamic evidence..............................26

Simple mapping specification and mapping configuration for dynamic evidence....................26

Mapping parent-child dynamic evidence..27

1.6 Mapping to third parties.. 29

Creating participants..30

Sample mapping schema for creating participants...30

Sample mapping configuration for creating participants.. 33

1.7 Data mapping schemas..34

Schema for mapping specifications...35

Schema for mapping configurations..36

1.8 Troubleshooting and error logging... 37

Handling validation issues in data mapping...38

Notices.. 39

Privacy policy...40

Trademarks.. 40

1 Developing with the Data Mapping Engine

Data mapping is the process by which data from the Cúram datastore is mapped into Cúram evidence entities. Data is mapped by the data mapping engine by using mappings that are configured in the data mapping editor. Mappings are created for a program on an intake application.

Use the Cúram data mapping engine to map citizen data that is captured during intake and screening processing to:

- Evidence and non-evidence entities in cases, or
- Filled-out application forms in PDF format

The data mapping editor allows you to rapidly create mappings to evidence. The data mapping editor saves the mappings in an XML mapping language. Details of the XML mapping language are provided to help you to understand how existing mappings are run and to understand more advanced mappings. The mapping language details also help you to create mappings to PDF forms and to maintain older mappings that might not be compatible with the data mapping editor. For more information about the data mapping editor, see the *Data Mapping Editor Guide*.

Related information

www.w3.org/XML/

1.1 Data mapping overview

An overview of data mapping which describes the data mapping environment and development process, how data is stored in the datastore, and the process of creating logical maps. The data mapping engine works with the Universal Access Responsive Web Application to help citizens screen themselves for benefits.

Related information

Data mapping environment and process

An overview of the data mapping environment, how its elements interact, and the data mapping development process.

The following diagram illustrates the data mapping environment and how its elements interact.

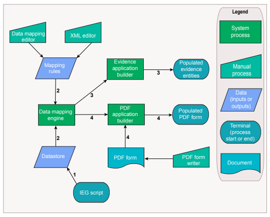

Figure 1: Data mapping environment

1. During the screening and intake process, a citizen submits their information. The IEG script stores the data that is gathered from citizens in the datastore.

2. Mapping rules are defined by using the data mapping editor or an XML editor. The data mapping engine reads the mapping rules and the datastore data.
3. The data mapping engine and the evidence application builder run the mapping rules to transform the datastore data into evidence entities
4. The data mapping engine and the PDF application builder run the mapping rules to populate fields on a PDF form. PDF form writers can create custom PDF forms by using tools such as Adobe™ Acrobat Pro, and Acro Software Inc. CutePDF®. If a custom PDF form is not used, the generic PDF builder maps the application data to a generic PDF form.

Development process

Data mapping development is a collaboration between business analysts and developers. The role of the business analyst is to logically map citizen data to either fields on a PDF application form or to evidence entities. The role of developers is to translate the business analysts' efforts by implementing the mapping in XML or in the data mapping editor.

The following is a summary of the development process.

1. Determine your required output, case evidence or a completed PDF application form.
2. Examine how a citizen's data is stored in the datastore during the intake and screening process.
3. Create logical maps between the forms of data. A logical map breaks down information about a person who is stored in the datastore to the case evidence entities.
4. Configure evidence types and PDF forms. You can customize the default evidence types that are provided or create new evidence types. PDF forms contain named fields, which you can map data to and change.
5. Associate the PDF form with the intake application. Associate the evidence types with the application case.
6. Define your mapping specification and configuration with the mapping editor or in an XML editor.
7. When your mapping is complete, associate the mapping with a program.
8. Extract the mapping configuration data to the file system.

How data is stored in the datastore

During intake and screening, IEG scripts capture information about a citizen and stores it in the datastore according to a predefined schema. Use this information to learn about how data is stored in the datastore so that logical maps can be created.

When mapping citizen data to case evidence or PDF application forms, first you must identify the desired output for the citizen's data, that is, as case evidence or a completed application form. Then, you must examine how a citizen's data is stored in the datastore during the intake and screening process.

Mapping data to case evidence allows case processing to use the data to determine eligibility. Mapping data to PDF application forms speeds up the process of completing the forms for manual submission. When you know the information that is required in the case evidence or the PDF application forms, and the information stored in the datastore, you can create logical maps between the forms of data.

The purpose of the screening and intake process is to help citizens to apply for benefits. At a minimum, an application form or case evidence requires the names of the members in a household and their relationships to the person who is applying for benefits. IEG scripts capture this information stores it in the datastore according to a predefined schema.

The image shows an example of a data structure in the datastore for a single benefit application. The application includes the members of a household and their relationships.

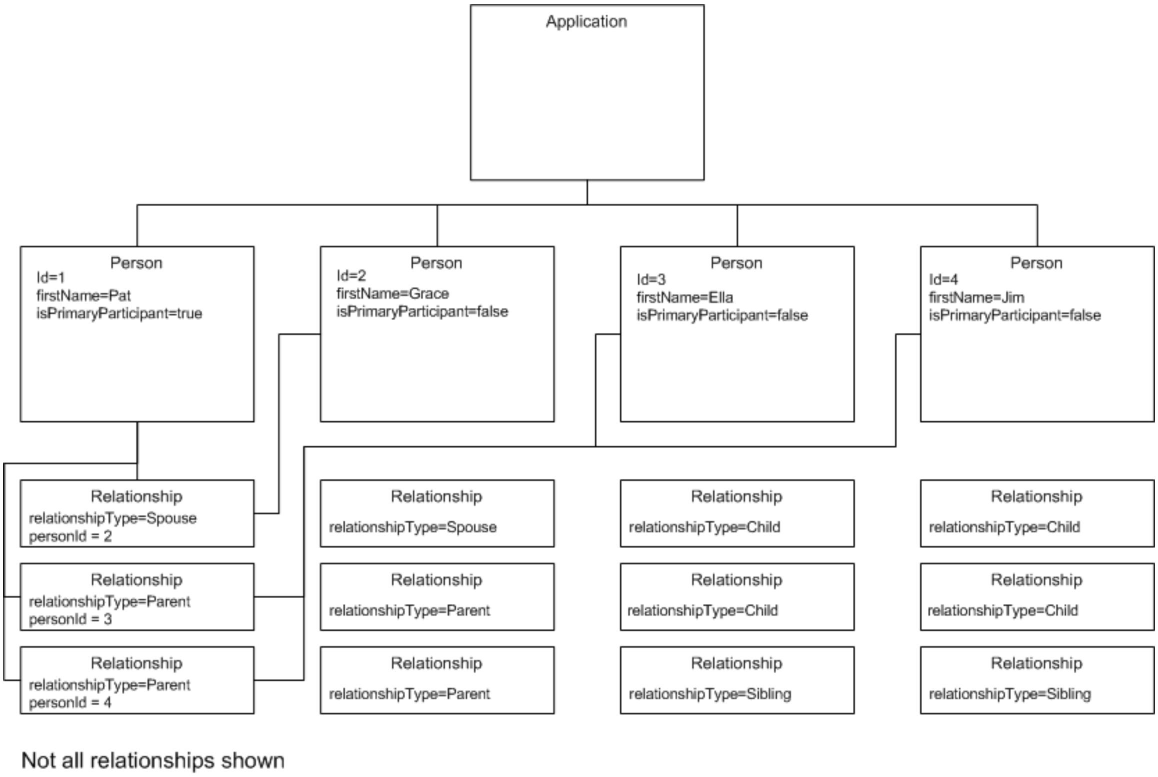

Figure 2: Example of data structure in the datastore

Related concepts

Creating logical maps

Business analysts create logical maps to break down the information about a person that is stored in the datastore into case evidence entities. For example, household member, living arrangement, and disability. Developers use the logical data map as the requirements specification for writing mapping specifications and mapping configurations.

Part of creating a logical map is to recognize business rules which can affect the way that the datastore maps data. For example, if the citizen indicates that they are blind and disabled, the business rules dictate that two disability evidence records must be created for the citizen (one for blindness and one for disability).

1.2 Writing mapping specifications for static evidence and PDFs

Use the code samples to learn how to write mapping specifications for static evidence and PDFs. A mapping specification describes how to map the data that is stored in a specific structure to a different one.

Each mapping specification refers to a source, where the data is coming from, and a target, where the data is going to.

Related concepts

Related information

www.w3.org/XML/

Writing mapping specifications

The data mapping engine uses mapping specifications to transform data in the datastore into another form. This section provides a simple mapping specification that maps a datastore entity to an evidence entity.

All of the data that is contained in a citizen's intake and screening application can be retrieved by reading the application entity, the data contained in its children, its children's children, and so on. To complete the transformation of data, the data mapping engine traverses the data in the datastore and applies the rules that are expressed in XML.

Simple mapping specification

The following is a simple mapping specification that maps a Person entity in the datastore to a Household Member evidence entity.

```
<?xml version="1.0" encoding="UTF-8"?>
<map xmlns="http://www.curamsoftware.com/schemas/GUMBO/Map"
    name="TestMapping">
   <map-entity source="Person">
       <target-entity name="HouseholdMember"
       id="HouseholdMemberTarget">
           <map-attribute from="isNativeAmerican"
           to="natAlaskOrAmerInd"/>
           <map-attribute from="comments" to="comments"/>
       </target-entity>
   </map-entity>
</map>
```

Line 4 indicates the source of the mapping while line 5 indicates the target. This rule can be paraphrased as 'For each Person entity encountered in the CDS, create a corresponding HouseholdMember entity'. The `<target-entity>` element contains two `<map-attribute>` elements on lines 6 and 7.

The `<map-attribute>` element on line 6 states that the `isNativeAmerican` attribute on the Person entity is mapped to the `natAlaskOrAmerInd` attribute on the `HouseholdMember` entity. Attributes are not mapped unless there is a specific `<map-attribute>` element. This is why line 6 states that the comments attribute in Person is mapped to the `comments` attribute in `HouseholdMember`.

Sometimes you might want to specify that a mapping occurs only in particular circumstances. For example, a `HeadOfHousehold` entity must only be created in the target system when the mapping encounters a `Person` entity in the CDS that has an `isPrimaryParticipant` indicator set to true. The previous code sample can be expanded to include this rule as follows.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <map xmlns="http://www.curamsoftware.com/schemas/GUMBO/Map"
3     name="TestMapping">
4     <map-entity source="Person">
5         <target-entity name="HouseholdMember"
          id="HouseholdMemberTarget">
6             <map-attribute from="isNativeAmerican"
              to="natAlaskOrAmerInd"/>
7             <map-attribute from="comments" to="comments"/>
8         </target-entity>
9     </map-entity>
10    <condition expression="Person.isPrimaryParticipant==true">
11        <target-entity name="HeadOfHousehold/>
12    </condition>
13 </map>
```

Related information

www.w3.org/XML/

Mapping condition expressions

Use the sample code to learn how to map condition expressions. In the following sample code, the values "Yes", "No" and "Unanswered" are represented as code table values that are used to record whether a person is a US citizen.

The value `ITYN4001` corresponds to the client answering "Yes" to this question. Note that the use of `&quot;`, this is because the quote symbols `""` cannot be used directly in XML. The syntax for conditionally mapping attributes is the same.

```
<condition expression="Person.isBlind==&quot;ITYN4001&quot;">
  <target-entity
    name="Disability"
    id="BlindDisabilityTarget"
  >
    <set-attribute
      name="disabilityType"
      value="DT1"
    />
   </target-entity>
</condition>
```

Mapping code table values

For some mappings, you can translate code table values that are recorded in the datastore into code table values that are used in the target model.

To do this, use a section at the start of the mapping script to specify the code table mappings. In this sample, values from the `CITIZENSTATUS` code table are mapped to values in the `AlienStatus` code table.

```
<map-code-table source-codetable="CITIZENSTATUS"
target-codetable="AlienStatus">
  <map-value source="US1" target="AS4"/>
  <map-value source="US2" target="AS1"/>
</map-code-table>
```

Mapping to multiple target entities

Sometimes you might need to create a group of target entities. For example, you might do this when you create a group of evidence entities where one is a parent, and the others are children of the parent evidence entity.

This sample code outlines how to create groups of related target entities.

```

        <target-entities>
    <target-entity
      name="BusinessAsset" id="BusinessAssetTarget"
      type="parent"
    >
      <map-attribute
        from="resourceAmount"
        to="amount"
      />
      <map-attribute
        from="amountOwed"
        to="amountOwed"
      />
    </target-entity>
    <target-entity
      name="Ownership" id="OwnershipTarget"
      type="child"
    >
      <set-attribute
        name="percentageOwned"
        value="100.0"
      />
    </target-entity>
</target-entities>
```

In the sample code, two entities are created. The `BusinessAsset` entity is parent evidence, while the `Ownership` entity is a child. Modelling the mapping in this way ensures that the correct Parent/Child evidence entity patterns are respected when the Evidence Application Builder creates the evidence.

Matching one parent entity to several child entities

In the previous section, we created group of target entities where the child and parent entities were related to each other.

In some cases, you might need to create only one parent entity for the whole case. All subsequent child entities are related to the same parent entity. For example:

```
<target-entities>
  <target-entity name="HholdMealsGroup" type="parent"
    attachment="case" id="MealGroup">
    <set-attribute name="groupName" value="sample"/>
  </target-entity>

  <target-entity name="MealGroupMember" type="child"
    id="MealGroupMember">
  </target-entity>
</target-entities>
```

In the sample code, a `HholdMealsGroup` and a `MealGroupMember` are created the first time the rule is executed. Each subsequent time the rule is executed, only a `MealGroupMember` is created and is associated with the same `HholdMealsGroup` entity.

Matching patterns and following associations in the datastore

The following example illustrates how to use the mapping engine to fill out a PDF form. In this example, the customer requires the PDF form to look like the sample in Table 1.

Table 1: Sample PDF form

Name	**Employer Name**	**Start Date**	**Annual Pay Before Taxes**
Pat	The Gingerman Bakery	1/2/2004	30000
Grace	Jarmin Pharmaceutical	1/3/2002	50000

Each field in the PDF form has a unique identity. For example, the field that contains the name, Pat, is identified as Job0.Name. The field containing 30000 is identified as Job0.Salary.

Consider how the datastore might store the intake information.

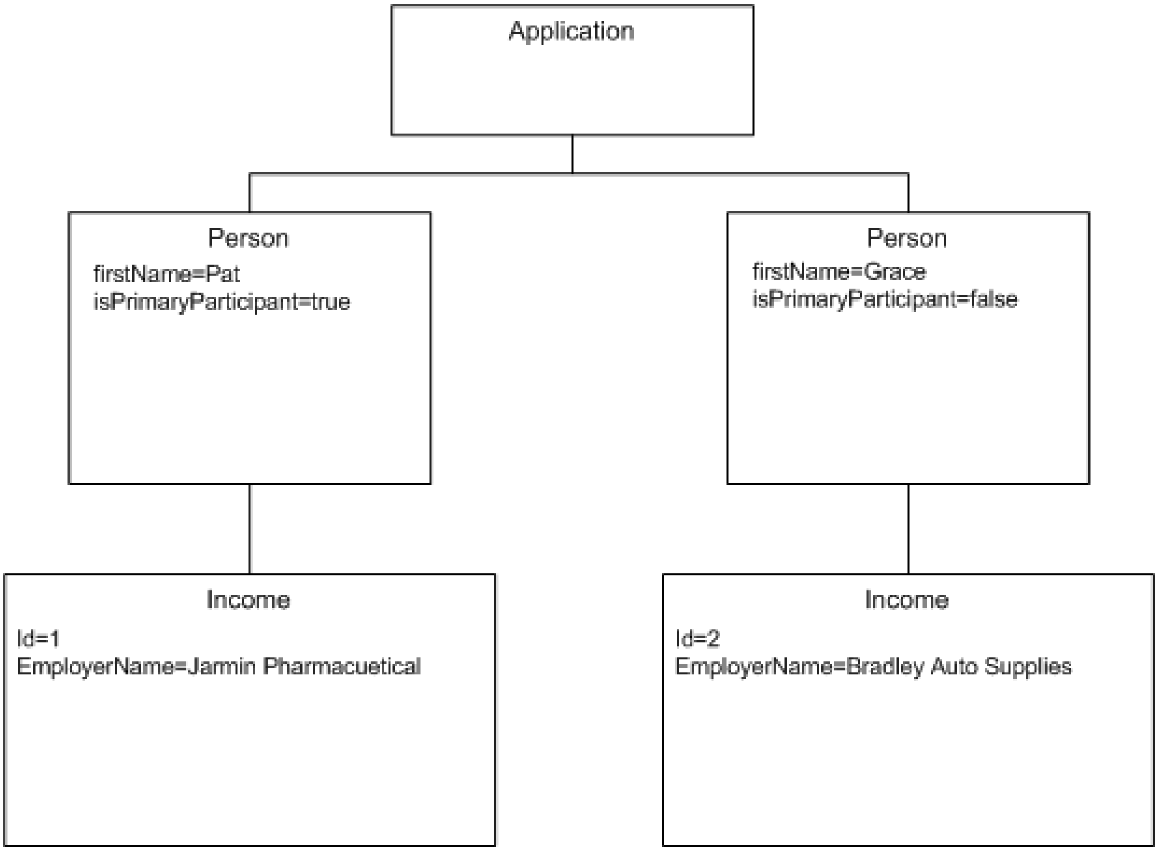
Application
Person
firstName=Pat
isPrimaryParticipant=true
Person
firstName=Grace
isPrimaryParticipant=false
Income
Id=1
EmployerName=Jarmin Pharmacuetical
Income
Id=2
EmployerName=Bradley Auto Supplies

Figure 3: Job income in datastore

To populate the Name field in the PDF form, the mapping specification must contain a rule that states that for each Income belonging to a Person, output the Person's `firstName` to the Name field.

This code sample shows how this is expressed in the mapping language.

```
1 <map-entity source="Person">
2  <map-entity source="Income">
3    <target-entity name="Job" id="JobTarget">
4      <map-attribute from="firstName" to="Name" entity="Person"/>
5      <map-attribute from="employerName" to="Employer"/>
6      …
7    </target-entity>
8  </map-entity>
9 </map-entity>
```

The mapping rule can be paraphrased as 'For each Income entity that is contained within a Person entity, create a Target entity of type `Job`. The `Name` attribute in the `Job` entity is mapped from the `firstName` attribute in the `Person` entity that contains the `Income` entity that is being mapped.'

The syntax `entity="Person"` on line 4 means that the `firstName` attribute belongs to the `Person` entity, not the `Income` entity. A more complex example of this type of mapping specification involves following associations or links from one entity to another.

Mapping household member data

The table shows how relationships are typically expressed in an application form. The requirement is to map the CDS entities to a prefilled application form that is similar to the one shown.

The difficult part is to fill in the field called 'How is this person related to you?' This field is represented by `"RelType"` in the sample code.

Table 2: Members of the household

Name	How is person related to you?	Date of birth	Social Security Number
Grace	Spouse	1/2/1981	209-57-9943
Ella	Child	1/3/2002	987-23-1190

This sample code shows how you can write the required mapping.

```
1  <condition expression="Person.isPrimaryParticipant == true">
2   <map-entity source="Person">
3     <map-entity source="Relationship">
4       <follow-association source="personID">
5         <target-entity name="Householder" id="Householder">
6           <map-attribute from="firstName" to="Name"/>
7           <map-attribute from="relationshipType" to="RelType"
            entity="Relationship"/>
8         </target-entity>
9       </follow-association>
10    </map-entity>
11  </map-entity>
12 </condition>
```

This can be paraphrased as 'For each Relationship that is contained in the primary participant, follow the association to the `Person` that is referred to by that Relationship. Map the `firstName` attribute of this `Person` entity to the `Name` field. Map the `relationshipType` attribute of the `Relationship` entity to the `RelType` field.'

The key to understanding the example is on line 7, where the `RelType` field is mapped from an attribute in the `Relationship` entity.

1.3 Writing mapping configurations for static evidence and PDFs

Use the code samples to learn how to write mapping configurations that are used by the evidence application builder and the PDF application builder.

The mapping specification contains the rules that are required to transform the data from one form to another. The mapping configuration provides extra information that is required to turn the transformed data into case evidence or completed PDF application forms.

Related concepts

Writing mapping configurations for the evidence application builder

When configured to use the evidence application builder, the mapping engine's work has two phases. In the first phase, the mapping engine creates an integrated case that includes case members, and concern role relationships between the case members. The second phase of the mapping process creates the evidence entities.

The case members are populated by finding entities in the datastore called Person. The data mapping engine treats all common data store entities called Person as a person or prospect person on a case. For example, in phase one of the mapping, the evidence application builder creates an integrated case with four case members.

- Pat is called the primary participant.
- A pair of concern role relationship records are created to indicate that Grace and Pat are married.
- The system creates all other concern role relationships as well (for parental and sibling relationships).
- Address and phone number records are also created.

Phase two of the mapping process creates the evidence entities. Code samples of these are included in the remainder of this section.

Simple mapping configuration

This sample code provides a simple mapping configuration for the evidence application builder. The evidence application builder is configured to create `HouseholdMember` and `HeadOfHousehold` evidence.

For example:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <application-builder-config
  xmlns="http://www.curamsoftware.com/schemas/GUMBO/ApplicationBuilderConfig">
3   <evidence-config package="curam.evidence">
4     <entity name="HouseholdMember"/>
5     <entity name="HeadOfHousehold"/>
6   <evidence-config package="curam.evidence">
7 </application-builder-config>
```

On line 3, the base Java package name is specified as *`curam.evidence`*. The evidence application builder uses this information to infer the following about `HouseholdMember`:

1. The name of the evidence service layer class is `curam.evidence.service.HouseholdMember`.
2. The name of name of the operation on this class used to create the evidence is `createHouseholdMemberEvidence()`.
3. The name of the class passed in to this call as an argument is `curam.evidence.entity.struct.HouseholdMemberEvidenceDetails`.

The evidence application builder uses this information to construct the `HouseholdMember` evidence for the current person being processed.

Above is based on the assumption that the evidence is coded according to certain patterns. This is guaranteed to be the case if the evidence generator is used to generate the evidence. It is possible for the evidence application builder to work with hand-coded evidence provided it follows the patterns that is used by the evidence generator.

Handling casePartcipantDetails fields

To execute the `createHouseholdMemberEvidence()` operation on the `HouseholdMember`, the CDME must populate the `caseParticipantDetails` field of the `HouseholdMemberEvidenceDetails` struct.

This code snippet illustrates this.

```
public final class HouseholdMemberEvidenceDetails
implements java.io.Serializable, curam.util.type.DeepCloneable {

  /** Attribute of the struct. */
  public  curam.core.sl.struct.CaseIDKey caseIDKey;

  /** Attribute of the struct. */
  public  curam.core.sl.struct.CaseParticipantDetails
    caseParticipantDetails;

  /** Attribute of the struct. */
  public  curam.core.sl.struct.EvidenceDescriptorDetails descriptor;

  /** Attribute of the struct. */
  public  curam.evidence.entity.struct.HouseholdMemberDtls dtls;
 …
}
```

The members of the `dtls` struct are mainly populated through the `<set-attribute>` and `<map-attribute>` elements in the mapping specification. For example, the following line in the mapping specification populates the field `natHawOrPaIsInd` with a value in the `dtls` struct:

```
<map-attribute
        from="nativeAlaskanOrAmericanIndian"
        to="natHawOrPaIsInd"
      />
```

The `caseParticipantDetails` field is often present in an `EvidenceDetails` struct. In this example, a case participant is created for Grace and the `caseParticipantDetails` refers to this case participant. The Data Mapping Engine does this automatically whenever it finds a field called `caseParticipantDetails` on the `EvidenceDetails` struct.

However, sometimes there are variations required in the handling of case participants, for example, when the `EvidenceDetails` struct contains additional case participants that refer to third parties. Consider the following:

```
public final class AnnuityEvidenceDetails
implements java.io.Serializable, curam.util.type.DeepCloneable {
  /** Attribute of the struct. */
  public  curam.core.sl.struct.CaseIDKey caseIDKey;

  /** Attribute of the struct. */
  public  curam.core.sl.struct.CaseParticipantDetails
    instCaseParticipantDetails;

  /** Attribute of the struct. */
  public  curam.core.sl.struct.EvidenceDescriptorDetails descriptor;

  /** Attribute of the struct. */
  public  curam.evidence.entity.struct.AnnuityDtls dtls;

  /** Attribute of the struct. */
  public  curam.evidence.entity.struct.AnnuityCaseParticipantDetails
    annuityCaseParticipantDetails;
```

In the example, the case participant who owns the annuity is referred to in the `AnnuityCaseParticipantDetails` struct that is aggregated under the field name `annuityCaseParticipantDetails`. The institution that holds the annuity is described in the `CaseParticipantDetails` struct and is aggregated under the field name `instCaseParticipantDetails`. This variation can be catered for by using the following evidence application builder configuration:

```
1    <entity
2      case-participant-class-name="curam.core.sl.struct.CaseParticipantDetails"
3      case-participant-relationship-name="annuityCaseParticipantDetails"
      name="Annuity"
4    >
5      <ev-field
6        aggregation="instCaseParticipantDetails"
7        referenced-as="participantName"
8        target-name="participantName"
9      />
10     <ev-field
11       aggregation="instCaseParticipantDetails"
12       referenced-as="address"
13       target-name="address"
14     />
15  </entity>
```

Lines 2 and 3 tell the Evidence Application Builder that the `caseParticipantDetails` for this evidence entity are referred to by the field name `annuityCaseParticipantDetails` using the struct `CaseParticipantDetails`. The lines 5-9 tell the Evidence Application Builder that the field `participantName` of the aggregated struct `instCaseParticipantDetails` can be referenced in the mapping specification as `"participantName"` (line 7). Similarly for the institutional address in lines 10-14. By using the following sample code, you can map the name and address of the institution that holds the annuity:

```
<target-entity name="Annuity" id="AnnuityTarget">
 <map-attribute
   from="institutionName"
   to="participantName"
 />
 <map-attribute
   from="institutionAddress"
   to="address"
 />
</target-entity>
```

In some cases, it might be too onerous to ask a client to fill out all of this third party information as part of an online intake. Instead, you can use the mapping specification to default these values and they can be filled out properly at the interview stage.

This sample code shows you how to default the values for a third-party participant like a financial institution.

```
1 <target-entity name="Annuity" id="AnnuityTarget">
2   <map-attribute
3          from="resourceAmount"
4          to="annuityValue"
5        />
6        <set-attribute
7          name="participantName"
8          value="Unknown"
9        />
10       <set-attribute
11         name="address"
12         value="curam.blankaddress"
13       />
14     </target-entity>
```

On line 12, the value `curam.blankaddress` causes a blank address to be inserted for the participant.

Setting target entity identifiers

As with a number of the previous code samples, the `<target-entity>` element on line 1 of sample 13, includes an `id` attribute called `AnnuityTarget`. This attribute is optional, however, it is good practice to include an `id` in all `<target-entity>` elements.

Including the `id` element allows the data mapping engine to distinguish between different distinct mappings from the same entity to the same target entity type. Consider the following example. The Person entity in the common datastore has two Boolean indicators: `isBlind` and `hasDisability`. As shown in this sample code, both map to the same target entity type, Disability.

```
1  <map-entity source="Person">
1    <condition expression="Person.isBlind==true">
2      <target-entity
3        id="DisabilityBlind"
4        name="Disability"
5      >
6        <set-attribute
7          name="disabilityType"
8          value="DT1"
9        />
10     </target-entity>
11   </condition>
12   <!-- Create an empty disability record. -->
13   <condition expression="Person.hasDisability==true">
14     <target-entity
15       id="DisabilityUnspecified"
16       name="Disability"
17     />
18   </condition>
19 </map-entity>
```

The first target on lines 1-11 ensures that if an applicant indicates that they are blind then a disability record of type blindness is created. The second target, lines 13-18, checks the `hasDisability` indicator. If it is set to true, a Disability record of unspecified type is created. By giving the two mappings a distinct `id`, the mapping engine can tell the two mappings apart. Without the `id`, the second mapping will not be processed.

Writing mapping configurations for the PDF application builder

A PDF form contains a number of fields of various types. Each field has a unique name. Cúram Workspace Services use this unique name to reference the field so that it can write data to that field. You must name the fields and set their properties according to certain conventions.

If you do not follow the conventions, the PDF application builder cannot map the data to the fields.

Note: This section describes the custom PDF application builder that allows you to map to a tailored PDF form. Customers can also use the generic PDF builder. For more information, see the *Merative SPM Universal Access Guide*.

Grouping fields into sections

The basic convention is that fields are grouped into sections. The sections do not necessarily need to correspond to sections on the form, but in most cases they will.

This code sample is an extract from a PDF application builder configuration. You see a section called Applicant that contains a group of fields that capture an applicant's details.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <application-builder-config
  xmlns="http://www.curamsoftware.com/schemas/GUMBO/ApplicationBuilderConfig">
3     <pdf-config>
4         <section name="Applicant">
5             <field name="Name" type="append" append-separator=" "/>
6             <field name="SSN"/>
7             <field name="DateofBirth"/>
8             <field name="Gender" type="button-radio"/>
9             <field name="USCitizen" type="button-radio"/>
10            <field name="blackOrAfricanAmerican"
                type="button-checkbox"/>
11            <field name="nativeAlaskanOrAmericanIndian"
                type="button-checkbox"/>
12            <field name="asian" type="button-checkbox"/>
13            <field name="nativeHawaiianOrPacificIslander"
                type="button-checkbox"/>
14            <field name="whiteOrCaucasian"
                type="button-checkbox"/>
15            <field name="EthnicOrigin" type="button-radio"/>
16        </section>
17    </pdf-config>
18</application-builder-config>
```

Based on line 4 of the sample code, the PDF application builder expects the target PDF form to contain a field called "Applicant.Name", a text field. Line 8 refers to a field on the PDF Form called "Applicant.Gender", which is a radio button. Lines 10-14 refer to fields that are check boxes

Filling text fields

In the previous section, line 6 of the sample refers to a standard text field, that is, `"SSN"`. Use the following code sample to learn about what the corresponding mapping might look like.

```
<target-entity name="Applicant">
  <map-attribute from="ssn" to="SSN"/>
</target-entity>
```

Line 5 in the sample in the previous section is different. The type is marked as `"append"`. This means that the same text field can be written to multiple times and each time the mapping engine writes to the text field, the result appends to the current value of the text field instead of overwriting it.

Each time an append occurs, the new data is separated from the old data by the append-separator, in this case a single space character. Taking a mapping file like that shown in Sample 16, and combining it with the mapping configuration shown in Sample 14 results in the Applicant Name field being populated with the Applicant's first name, middle initial and surname, for example, 'Pat A Kayek'.

```
<target-entity name="Applicant">
  <map-attribute from="firstName" to="Name"/>
  <map-attribute from="middleInitial" to="Name"/>
  <map-attribute from="lastName" to="Name"/>
</target-entity>
```

Appending text fields are also useful for creating a comma separated list of items. Consider a field that asks the client to provide a list of people in their household who are pregnant. An extract from the mapping XML might look like the following sample code.

```
<condition expression="Person.isPregnant == true">
   <target-entity name="Pregnancy">
     <map-attribute from="firstName" to="Pregnancy"/>
     <set-attribute name="HasPregnancies" value="Yes"/>
   </target-entity>
 </condition>
```

The corresponding mapping configuration is shown in the next code sample. Each time the mapping engine processes a person in the household for which the isPregnant indicator is set to true, the first name of that person is appended to the Pregnancy.Pregnancies field.

```
<section name="Pregnancy">
  <field name="Pregnancies" type="button-checkbox"/>
  <field name="Pregnancy" type="append" append-separator=", "/>
</section>
```

Filling repeated sections and code table descriptions

Some PDF forms contain repeated sections, for example. 'List the details of all the people in your household'. Use this information to learn about how to fill repeated sections and code table descriptions.

You must name the PDF fields according to the correct conventions. For example, fields that are used to capture data about household members might be named as follows.

Table 3: Fields in a PDF Form for recording household members

Name	How Is Person Related to You?	Date of Birth	Social Security Number
OtherPerson0.Name	OtherPerson0.RelType	OtherPerson0.DateOfBirth	OtherPerson0.SSN
OtherPerson1.Name	OtherPerson1.RelType	OtherPerson1.DateOfBirth	OtherPerson1.SSN
OtherPerson2.Name	OtherPerson2.RelType	OtherPerson2.DateOfBirth	OtherPerson2.SSN

This sample code outlines the corresponding mapping configuration.

```
1  <section name="Person" type="multiple">
2   <field name="Name" type="append" append-separator=" "/>
3   <field name="RelType" codetable-class="RelationshipTypeCode"/>
4   <field name="DateofBirth"/>
5  </section>
```

> **Note:** In line 1, the attribute `type="multiple"` causes the section to be repeated. Note the`codetable-class` attribute on line 3 in the sample. This is a useful attribute that causes code table values to be translated into localized descriptions. By using it in the above context, the script author ensures that the second column in the form is populated with localized values like Parent and Sibling instead of meaningless codes like RT1 or RT3.

Check boxes

A check box is a single field that can either be checked or cleared. The PDF application builder assumes that setting a check box field to the value 'Yes' causes it to be checked and setting it to 'No' causes it to be cleared.

As the PDF form author, you must ensure that this convention is adhered to. Whenever the mapping engine maps a boolean value to a check box field, it is automatically mapped as follows: True maps to 'Yes' and false maps to 'No'.

Radio buttons

In a PDF form, a collection of radio buttons are treated as a single field. Radio buttons allow only one item to be selected at a time. The individual radio button items are selected by writing a particular value to the radio button. To determine which item is selected, you must specify an export value for each item in the PDF form.

In Curam, a typical use for a radio button is to use export values to denote a number of code table items.

Consider the example of a radio button that is used to denote Male or Female. The code table values for Male and Female are 'SX1' and 'SX2' respectively. The PDF author creates a single radio button field called 'Applicant.Gender'. The 'Male' item is denoted by the export value 'SX1' while the female item is denoted by the export value 'SX2'.

This sample code shows the mapping.

```
<target-entity name="Applicant">
  <map-attribute from="gender" to="Gender"/>
</target-entity>
```

This sample code shows the corresponding mapping configuration.

```
<section name="Applicant">
   <field name="Gender" type="button-radio"/>
 </section>
```

Choice combos

A choice combo is a drop-down list of items where the user can choose one item. The name of the item provides enough information for the user of the form to decide which item to choose.

As an example, let's suppose you want to provide a drop-down to represent the alien status of a person. Cúram has an AlienStatus code table with codes that correspond to the following descriptions:

- Alien
- US Citizen
- Undocumented Alien
- Refugee
- Non Citizen National

To create the alien status drop-down, you must create a choice combo and set the item text for each item in the drop-down to the above values.

In addition, you must ensure that the code table description and not the code table code is sent to the form. This sample code outlines how to do this.

```
<section name="AlienPerson" type="multiple">
    <field name="CitizenshipStatus" type="choice-combo" codetable-class="AlienStatus"/>
  </section>
```

1.4 Configuring PDF application forms

You can configure the PDF application form for manual submission during an intake.

> **Note:** Ensure that you use a PDF form and not a PDF document. The PDF must contain a form and the form must contain fields. If you are using the PDF application builder to map data to the form, each field on the form must have a unique name, for example, PersonalDetails.surname versus Child1Details.surname.

You can use a program like Adobe™ Acrobat Pro or Acro Software Inc. CutePDF® to edit your form. If you are using Adobe™ Acrobat Pro, ensure that you save the form as an AcroForm not an XFA.

1. Upload your PDF form to the Cúram Universal Access Responsive Web Application.
    1. Log in as Administrator and click **Universal Access** from the shortcuts.
    2. Click **PDF Forms** and then upload your PDF form and name it.
2. Associate an intake application type with your PDF form.
    1. Go to Intake Applications.
    2. Select the relevant intake application type and click **Edit**.
    3. From the PDF Form drop-down list, select your uploaded PDF form.
3. Specify PDF mappings for each program that you are interested in.
    1. Ensure that you have written a PDF mapping specification and a PDF application configuration in XML format. For more information, see 1.2 Writing mapping specifications for static evidence and PDFs on page 12 and 1.3 Writing mapping configurations for static evidence and PDFs on page 17.
    2. Click **Programs** in Universal Access Administration. A list of programs is shown.
    3. Select a program and then select **Mappings** > **New Mapping**.
    4. Select **PDF Form Creation**.
    5. Upload your PDF mapping configuration file and PDF mapping specification file.
4. Test your mapping by completing an intake for the relevant program. At the end of the intake, select the link to display the PDF file.

Related information

https://www.adobe.com/acrobat.html

https://www.acrosoftware.com/index.htm

1.5 Writing mapping specifications and configurations for dynamic evidence

Use the code samples to learn how to map citizen data that is captured during an intake to dynamic evidence entities. A dynamic evidence type can have multiple evidence type versions over its lifetime. You must provide the correct mapping specification and configurations for the currently effective evidence type version.

By its nature, dynamic evidence is not modeled. There is single entity, or a set of entities, that contain data for all dynamic evidence types. At any point in time, only one evidence type version is effective. All metadata elements, such as attributes and relationships, are defined at the evidence type version level.

To write mapping specifications and configurations for dynamic evidence, you must have knowledge of the basic concepts of dynamic evidence. It is assumed that you have a good understanding of evidence nature, dynamic evidence type definition, evidence version definition, and XML metadata of dynamic evidence.

Related concepts

Related information

www.w3.org/XML/

Simple mapping specification and mapping configuration for dynamic evidence

The simple mapping specification maps data from `HouseHoldMember` entity in the datastore to the `HouseHoldMember` dynamic evidence entity that was defined in the previous section. The simple mapping configuration is defined for the `HouseHoldMember` dynamic evidence type.

Simple mapping specification

The attribute name in the `to` field must match the `name` field of the `DataAttribute` element in the dynamic evidence metadata. This means that you must ensure that the attributes to which the data will be mapped are present in the metadata of the dynamic evidence type.

```
?xml version="1.0" encoding="UTF-8"?>
<map xmlns="http://www.curamsoftware.com/schemas/GUMBO/Map"
                                          name="EvidenceMapping">
    <map-entity source="HouseHoldMember">
        <target-entity name="HouseHoldMember">
            <map-attribute from="blkOrAfrAmerInd"
                                          to="blkOrAfrAmerInd" />
            <map-attribute from="ssnStatus" to="ssnStatus" />
            <map-attribute from="startDate" to="startDate" />
            <map-attribute from="endDate" to="endDate" />
            <map-attribute from="comments" to="comments" />
        </target-entity>
    </map-entity>
</map>
```

Simple mapping configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<application-builder-config >
    <evidence-config package="curam.gumbo.evidence">
        <entity name="HouseHoldMember" ev-type-code="DE_HMEMBER"/>
    </evidence-config>
</application-builder-config>
```

You must specify the dynamic evidence type code in the `ev-type-code` attribute so that the system can determine whether the evidence is static or dynamic. If you leave this attribute blank or it contains an invalid entry, the system assumes that the type of evidence is static and will proceed to map the data.

Mapping parent-child dynamic evidence

Use the code samples to learn how to map evidence that contains a parent-child relationship. Sample parent-child dynamic evidence metadata for an Adoption evidence type is provided with its corresponding mapping specification, and mapping configuration.

Simple parent-child dynamic evidence metadata

The following metadata code sample includes attributes for the Adoption dynamic evidence type. The Adoption metadata has two attributes that describe that this entity has two related `CaseParticipant` fields. The field `"caseParticipantRoleID"` is a primary `CaseParticipantRole` and `"parCaseParticipantRoleID"` is an associated `CaseParticipantRole`.

```
<?xml version="1.0" encoding="UTF-8"?>
<EvidenceTypeVersion xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../../
DynamicEvidence/source/curam/dynamicevidence/definition/impl/
                         xmlresources/DynamicEvidenceMetadata.xsd">
    <Model>
        <Attributes>
            <Attribute>
                <DataAttribute name="adoptionFinalizedDate">
                    <DomainType dataType="Date" />
                </DataAttribute>
            </Attribute>
            <Attribute>
                <RelatedCPAttribute name="caseParticipantRoleID"
                        participantType="Person" volatile="true" />
            </Attribute>
            <Attribute>
                <RelatedCPAttribute name="parCaseParticipantRoleID"
                                        participantType="Person" />
            </Attribute>
        </Attributes>
    </Model>
    <UserInterface>
        <Cluster>
            <RelCPCluster fullCreateParticipant="true">
                <StandardField source="caseParticipantRoleID" />
            </RelCPCluster>
        </Cluster>
        <Cluster>
            <RelCPCluster fullCreateParticipant="true">
                <StandardField source="parCaseParticipantRoleID" />
            </RelCPCluster>
        </Cluster>
    </UserInterface>
</EvidenceTypeVersion>
```

The `AdoptionPayment` metadata has a relationship with its parent dynamic evidence as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<EvidenceTypeVersion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="file://DynamicEvidenceMetadata.xsd"
 javaHookClassNameForCalculatedAttributes="curam.dynamicevidencetest.
 hook.impl.AdoptionPaymentCalcualtedAttributeHook"
 useHookForCalculatedAttributes="true">
    <Model>
        <Attributes>
            <Attribute>
                <DataAttribute name="amount">
                    <DomainType dataType="Money" />
                </DataAttribute>
            </Attribute>
            <Attribute>
                <CalculatedAttribute name="parentName">
                    <DomainType dataType="String" />
                </CalculatedAttribute>
            </Attribute>
        </Attributes>
        <Relationships>
            <Relationship>
                <MandatoryParent name="adoptions"
                           evidenceTypeCode="DET004" />
            </Relationship>
        </Relationships>
    </Model>
</EvidenceTypeVersion>
```

Simple parent-child mapping specification

The following mapping specification shows a parent-child relationship. Several attributes are also defined for the Adoption entity, which are used to create a new participant. The values for these attributes are read from the Curam DataStore.

```
<?xml version="1.0" encoding="UTF-8"?>
<map xmlns="http://www.curamsoftware.com/schemas/GUMBO/Map"
                                     name="ParentChildMapping">
    <map-entity source="Adoption">
        <target-entities>
            <target-entity name="Adoption" type="parent" id="parent">
                <map-attribute from="adoptionFinalizedDate"
                                       to="adoptionFinalizedDate" />
                <map-attribute from="adParentName"
                                                to="adParentName" />
                <map-attribute from="adParentStreet1"
                                             to="adParentStreet1" />
                <map-attribute from="adParentStreet2"
                                             to="adParentStreet2" />
                <map-attribute from="adParentCity"
                                                to="adParentCity" />
                <map-attribute from="adParentState"
                                               to="adParentState" />
                <map-attribute from="adParentZipCode"
                                             to="adParentZipCode" />
            </target-entity>
            <target-entity name="AdoptionPayment"
                                            type="child" id="child">
                <set-attribute name="amount" value="2200" />
            </target-entity>
        </target-entities>
    </map-entity>
</map>
```

Simple mapping configuration for parent-child relationship

The `<def-create-participant>` and `<create-participant>` elements are introduced in this section.

Important: The `<entity>` element includes a new attribute called `"dyn-evidence-primary-cpr-field-name"`. You must specify the attribute name of the primary `CaseParticipantRole` that is defined in metadata in this attribute.

In this code sample, `"caseParticipantRoleID"` is the primary `CaseParticipantRole` that is defined in the Adoption entity. Similarly, the related `CaseParticipantRole` attribute name (`"parCaseParticipantRoleID"` in this example) is defined in the `name` field in the `<create-participant>` element. For static evidence, the same `name` field in the `<create-participant>` element is used to reference the corresponding aggregation name.

```
<?xml version="1.0" encoding="UTF-8"?>
<application-builder-config xmlns="http://www.curamsoftware.com/
                            schemas/GUMBO/ApplicationBuilderConfig">
    <evidence-config package="curam.evidence">
     <def-create-participant id="AdoptedParentDetails" type="RL13">
            <participant-name-field name="firstName"
                                    from="adParentName" order="1" />
            <participant-address type="AT3">
                <address-field name="addressLine1"
                                           from="adParentStreet1" />
                <address-field name="addressLine2"
                                           from="adParentStreet2" />
                <address-field name="city" from="adParentCity" />
                <address-field name="state" from="adParentState" />
                <address-field name="zip" from="adParentZipCode" />
            </participant-address>
      </def-create-participant>
       <entity name="Adoption" ev-type-code="DET004"
       dyn-evidence-primary-cpr-field-name="caseParticipantRoleID">
            <create-participant refid="AdoptedParentDetails"
                         name="parCaseParticipantRoleID" role="" />
        </entity>
        <entity name="AdoptionPayment" ev-type-code="DET005"/>
    </evidence-config>
</application-builder-config>
```

1.6 Mapping to third parties

Many evidence types reference third parties. You must insert third parties onto the case as new case participants with their own unique case participant role. The application builder configuration schema includes elements and attributes that you can use to create a new participant and map an address to that participant.

The participant is either a representative or a prospect person. For example, a Pregnancy record can have an associated father. If the father is absent, they can be recorded as a Prospect Person case participant. In another example, Student evidence needs to be associated with a school. The school is entered as a Representative case participant. These new cases need to be created as needed during the mapping. Also, these cases need to contain as much information as possible to smooth the intake process for the assigned caseworkers who process the case.

Mapping addresses is necessary so that a new or existing participant can be associated with a new piece of evidence. One of the challenges in creating new participants is mapping their addresses. Address fields that are stored in the datastore, such as `ADD1`, must be correctly aggregated into a correctly formatted address structure to ensure that the participant is properly created.

The logic for creating a new participant and mapping an address to that participant is isolated from the evidence application builder.

Related concepts

Creating participants

There are two parts to creating participants: defining the participant creation behavior and creating the participant.

Defining participant creation behavior

The element `<def-create-participant>` is part of the `<evidence-config>` element. This element is used to define behavior for creating a participant. This same behavior can be reused by multiple `<entity>` definitions through the `id`. You must define the data type of all attributes that are referenced in this sample code as 'String'.

```
<def-create-participant id="SchoolRepresentative" type="RL13">
  <participant-name-field name="firstName" from="participantName"
                                                    order="1"/>
       <participant-address type="AT3">
            <address-field name="addressLine1" from="street1"/>
            <address-field name="addressLine2" from="street2"/>
            <address-field name="city" from="city"/>
            <address-field name="state" from="state"/>
            <address-field name="zip" from="zipCode"/>
        </participant-address>
</def-create-participant>
```

Creating a participant

The `<create-participant>` element has been added into `<entity>` element. This element instructs the application builder to create the participant as defined in the participant creator definition.

```
<entity case-participant-class-name="curam.core.sl.struct.
 CaseParticipantDetails"case-participant-relationship-name=
                                "curam.none" name="Student">
          <create-participant refid="SchoolRepresentative"
               name="schCaseParticipantDetails" role="SCH"/>
</entity>
```

Sample mapping schema for creating participants

Follow this schema when you write the mapping specification.

In the Education entity, the attributes are mapped directly to the Student evidence entity. The `schoolName`, `schoolStreet1`, `schoolStreet2` attributes and so on, are used to create a new participant and address.

```
<?xml version="1.0" encoding="UTF-8"?>
<map xmlns="http://www.curamsoftware.com/schemas/GUMBO/Map"
  from-schema="GumboDS" name="TestMapping"to-schema="CGISS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="...\EJBServer\components\
  WorkspaceServices\lib\Mapping.xsd"><map-entity source="Person">
     <target-entity id="householdMember" name="HouseholdMember">
      <map-attribute from="ssnStatus" to="ssnStatus"/>
      <map-attribute from="blackOrAfricanAmerican"
                                to="blkOrAfrAmerInd"/>
      <map-attribute from="nativeAlaskanOrAmericanIndian"
                                to="natHawOrPaIsInd"/>
      <map-attribute from="asian" to="asianInd"/>
      <map-attribute from="nativeHawaiianOrPacificIslander"
```

```
                                to="natHawOrPaIsInd"/>
      <map-attribute from="whiteOrCaucasian"
                                 to="whiteOrCaucInd"/>
      <map-attribute from="isMigrantOrSeasonalFarmWorker"
                              to="migrantFWorkerInd"/>
    </target-entity>
    <target-entity id="livingArrange" name="LivingArrange">
      <map-attribute from="accommodationType"
                                   to="livingArrangeType"/>
    </target-entity>
  </map-entity>
  <map-entity source="Education">
        <condition expression=
                           "Education.highestGrade!
=&quot;&quot;">
            <target-entity id="highestGrade" name="Student">
                <map-attribute from="highestGrade"
                                         to="highGradeCompleted"/
>
                <map-attribute from="attendanceFrequency"
                                              to="studentStatus"/
>
                <map-attribute from="schoolName"
                                            to="participantName"/
>
                <map-attribute from="schoolStreet1" to="street1"/
>
                <map-attribute from="schoolStreet2" to="street2"/
>
                <map-attribute from="schoolCity" to="city"/>
                <map-attribute from="schoolState" to="state"/>
                <map-attribute from="schoolZipCode" to="zipCode"/
>
            </target-entity>
        </condition>
    </map-entity>
  <map-entity source="HealthInsuranceExpense">
    <target-entity id="healthInsuranceExpense"

 name="MedicalInsurance">
      <map-attribute from="policyNumber" to="policyNumber"/>
      <map-attribute from="groupNumber" to="groupPolicyNumber"/>
        <map-attribute from="policyHolderParticipantName"
              to="policyHolderParticipantName"/>
        <map-attribute from="policyHolderStreet1"
                                        to="policyHolderStreet1"/
>
        <map-attribute from="policyHolderStreet2"
                                        to="policyHolderStreet2"/
>
        <map-attribute from="policyHolderCity"
                                           to="policyHolderCity"/
>
        <map-attribute from="policyHolderState"
                                          to="policyHolderState"/
>
        <map-attribute from="policyHolderZipCode"
```

```
                                        to="policyHolderZipCode"/
>
        <map-attribute from="groupParticipantName"
                                       to="groupParticipantName"/
>
        <map-attribute from="groupStreet1" to="groupStreet1"/>
        <map-attribute from="groupStreet2" to="groupStreet2"/>
        <map-attribute from="groupCity" to="groupCity"/>
        <map-attribute from="groupState" to="groupState"/>
        <map-attribute from="groupZipCode" to="groupZipCode"/>
        <map-attribute from="insuranceProvider"
                                          to="insuranceProvider"/
>
        <map-attribute from="InsProviderStreet1"
                                         to="InsProviderStreet1"/
>
        <map-attribute from="InsProviderStreet2"
                                         to="InsProviderStreet2"/
>
        <map-attribute from="InsProviderCity"
                                            to="InsProviderCity"/
>
        <map-attribute from="InsProviderState"
                                           to="InsProviderState"/
>
        <map-attribute from="InsProviderZipCode"
                                         to="InsProviderZipCode"/
>
        <map-entity source="HealthInsuranceExpenseRelationship">
        <target-entity id="healthInsuranceExpenseRelationship"

 name="Coverage">
          <map-attribute from="personID"
                                      to="caseParticipantRoleID"/
>
        </target-entity>
      </map-entity>
    </target-entity>
  </map-entity>
</map>
```

Sample mapping configuration for creating participants

Follow this XML configuration when you write the mapping specification.

```xml
<?xml version="1.0" encoding="UTF-8"?><application-builder-config xmlns=
"http://www.curamsoftware.com/schemas/GUMBO/ApplicationBuilderConfig">
 <evidence-config package="curam.evidence">
      <def-create-participant id="SchoolRepresentative" type="RL13">
          <participant-name-field name="firstName" from=
                                       "participantName" order="1"/>
          <participant-address type="AT3">
              <address-field name="addressLine1" from="street1"/>
              <address-field name="addressLine2" from="street2"/>
              <address-field name="city" from="city"/>
              <address-field name="state" from="state"/>
              <address-field name="zip" from="zipCode"/>
          </participant-address>
      </def-create-participant>
      <def-create-participant id="MedicalInsurancePolicyHolder"
                                                        type="RL7">
          <participant-name-field name="firstName"
                     from="policyHolderParticipantName" order="1"/>
          <participant-address type="AT3">
              <address-field name="addressLine1"
                                       from="policyHolderStreet1"/>
              <address-field name="addressLine2"
                                       from="policyHolderStreet2"/>
              <address-field name="city" from="policyHolderCity"/>
              <address-field name="state" from="policyHolderState"/>
              <address-field name="zip" from="policyHolderZipCode"/>
          </participant-address>
      </def-create-participant>
      <def-create-participant id="MedicalInsuranceGroup"
                                                      type="RL13">
          <participant-name-field name="firstName"
                           from="groupParticipantName" order="1"/>
          <participant-address type="AT3">
              <address-field name="addressLine1"
                                             from="groupStreet1"/>
              <address-field name="addressLine2"
                                             from="groupStreet2"/>
              <address-field name="city" from="groupCity"/>
              <address-field name="state" from="groupState"/>
              <address-field name="zip" from="groupZipCode"/>
          </participant-address>
      </def-create-participant>
      <def-create-participant id="MedicalInsuranceProvider"
                                                      type="RL13">
          <participant-name-field name="firstName"
                              from="insuranceProvider" order="1"/>
          <participant-address type="AT3">
              <address-field name="addressLine1"
                                       from="InsProviderStreet1"/>
              <address-field name="addressLine2"
                                       from="InsProviderStreet2"/>
              <address-field name="city" from="InsProviderCity"/>
              <address-field name="state" from="InsProviderState"/>
              <address-field name="zip" from="InsProviderZipCode"/>
          </participant-address>
      </def-create-participant>

    <entity name="HouseholdMember"/>
    <entity name="HeadOfHousehold"/>
    <entity case-participant-class-name="curam.core.sl.struct.
     CaseParticipantDetails"case-participant-relationship-name=
                                  "curam.none" name="Student">
        <create-participant refid="SchoolRepresentative"
                 name="schCaseParticipantDetails" role="SCH"/>
    </entity>
    <entity name="MedicalInsurance">
        <create-participant refid="MedicalInsurancePolicyHolder"
               name="plchdrCaseParticipantDetails" role="MIPH"/>
        <create-participant refid="MedicalInsuranceGroup"
                 name="groupCaseParticipantDetails" role="GPP"/>
        <create-participant refid="MedicalInsuranceProvider"
                   name="insCaseParticipantDetails" role="MIP"/>
    </entity>
</application-builder-config>
```

1.7 Data mapping schemas

The XML schemas to use for writing mapping specifications and mapping configurations.

Schema for mapping specifications

The schema to follow when writing mapping specifications.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.curamsoftware.com/schemas/GUMBO/Map"
    xmlns:mp="http://www.curamsoftware.com/schemas/GUMBO/Map"
    elementFormDefault="qualified">

    <xs:simpleType name="TargetEntityRoleType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="parent"/>
            <xs:enumeration value="child"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="AttachmentType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="case"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="MapAttributeType">
        <xs:attribute name="from" type="xs:NCName" use="required"/>
        <xs:attribute name="to" type="xs:NCName" use="required"/>
        <xs:attribute name="mapping-function" type="xs:string"
          use="optional"/>
        <xs:attribute name="mapping-rule" type="xs:string"
          use="optional"/>
        <xs:attribute name="entity" type="xs:NCName" use="optional"/>
    </xs:complexType>

    <xs:complexType name="SetAttributeType">
        <xs:attribute name="name" type="xs:NCName"/>
        <xs:attribute name="value" type="xs:string"/>
    </xs:complexType>

    <xs:element name="set-attribute" type="mp:SetAttributeType"/>

    <xs:complexType name="TargetEntityType">
        <xs:sequence>
            <xs:element name="map-attribute" type="mp:MapAttributeType"
minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:set-attribute" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:condition" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:NCName"/>
        <xs:attribute name="type" type="mp:TargetEntityRoleType"/>
        <xs:attribute name="attachment" type="mp:AttachmentType"/>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
    </xs:complexType>

    <xs:element name="target-entity" type="mp:TargetEntityType"/>

    <xs:complexType name="ConditionType">
        <xs:choice>
            <xs:element ref="mp:target-entity" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:target-entities" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:set-attribute" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:map-entity" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:choice>
        <xs:attribute name="expression" type="xs:string"/>
    </xs:complexType>

    <xs:element name="condition" type="mp:ConditionType"/>

    <xs:complexType name="MapEntityType">
        <xs:sequence>
            <xs:element ref="mp:target-entity" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:target-entities" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element  ref="mp:condition" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:map-entity" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element ref="mp:follow-association" minOccurs="0"
```

Schema for mapping configurations

The schema to follow when writing mapping configurations.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 Copyright Merative US L.P. 2012
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="
  http://www.curamsoftware.com/schemas/GUMBO/ApplicationBuilderConfig"
    xmlns:abc="
 http://www.curamsoftware.com/schemas/GUMBO/ApplicationBuilderConfig"
    elementFormDefault="qualified">

    <xs:complexType name="EvFieldType">
        <xs:attribute name="referenced-as" type="xs:NCName" use="optional"/>
        <xs:attribute name="target-name" type="xs:NCName" use="optional"/>
        <xs:attribute name="aggregation" type="xs:NCName" use="optional"/>
        <xs:attribute name="is-reference-attribute" type="xs:boolean"
   use="optional"/>
        <xs:attribute name="map-as-concernrole-id" type="xs:boolean"
   use="optional"/>
    </xs:complexType>

    <xs:complexType name="ParticipantCreatorType">
        <xs:attribute name="refid" type="xs:string" use="required"/>
        <xs:attribute name="name" type="xs:NCName" use="required"/>
        <xs:attribute name="role" type="xs:string" use="required"/>
    </xs:complexType>

    <xs:complexType name="ParticipantNameFieldType">
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="from" type="xs:NCName" use="required"/>
        <xs:attribute name="order" type="xs:positiveInteger" use="optional"/>
    </xs:complexType>

    <xs:complexType name="AddressFieldType">
        <xs:attribute name="name" type="xs:NCName" use="required"/>
        <xs:attribute name="from" type="xs:NCName" use="required"/>
    </xs:complexType>

    <xs:complexType name="ParticipantAddressType">
        <xs:sequence>
            <xs:element ref="abc:address-field" minOccurs="0"
   maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="type" type="xs:string" use="required"/>
    </xs:complexType>

    <xs:element name="participant-name-field"
 type="abc:ParticipantNameFieldType"/>
    <xs:element name="participant-address"
 type="abc:ParticipantAddressType"/>
    <xs:element name="ev-field" type="abc:EvFieldType"/>
    <xs:element name="create-participant" type="abc:ParticipantCreatorType"/>
    <xs:element name="address-field" type="abc:AddressFieldType"/>

    <xs:complexType name="EntityType">
        <xs:sequence>
            <xs:element ref="abc:ev-field" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="abc:create-participant" minOccurs="0" maxOccurs=
    "unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:NCName"/>
        <xs:attribute name="package" type="xs:string"/>
        <xs:attribute name="case-participant-relationship-name" type="xs:NCName"/>
        <xs:attribute name="case-participant-class-name" type="xs:NCName"/>
        <xs:attribute name="ev-type-code" type="xs:NCName" use="optional"/>
        <xs:attribute name="dyn-evidence-primary-cpr-field-name" type="xs:NCName"
   use="optional"/>
        <xs:attribute name="target-entity-type" type="xs:NCName" use="optional"/>

    </xs:complexType>

    <xs:complexType name="ParticipantCreatorDefinitionType">
        <xs:sequence>
            <xs:element ref="abc:participant-name-field"
   minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="abc:participant-address" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="type" type="xs:string" use="required"/>
    </xs:complexType>
```

1.8 Troubleshooting and error logging

During mapping development, unexpected errors can occur. You can configure the application to increase the level of error logging and you can review the list of default error codes. Guidance is also provided to help you to address validation issues.

To increase the granularity of the error messages, set the application property *`curam.workspaceservices.application.processing.logging.on`* to TRUE.

Error codes

To help you to troubleshoot, the following is the list of error codes and their code table descriptions that the mapping engine returns.

```
APROCER001 An error occurred creating a person.
APROCER002 An error occurred creating a prospect person.
APROCER003 An error occurred creating a prospect person.
APROCER004 An error occurred creating a case.
APROCER005 An error occurred while performing a "map-attribute" mapping.
APROCER006 An error occurred while performing a "set-attribute" mapping.
APROCER007 An error occurred while performing a "map-address" mapping.
APROCER008 General mapping failure.
APROCER009 Error creating evidence.
APROCER010 More than one PDF form is registered against the program type.
APROCER011 Error setting the alternate id type for a Prospect Person.
APROCER012 Invalid alternate ID value.
APROCER013 Error the Evidence Application Builder has not been correctly configured.
APROCER014 Evidence type not listed in the Mapping Configuration.
APROCER015 No parent evidence entity found.
APROCER016 An error occurred when trying to unmarshal the application XML.
APROCER017 An error occurred when trying to set a field value.
APROCER018 An error occurred when trying to create the PDF document.
APROCER019 An error occurred when trying to create the PDF document. A form code could
 not be mapped to a codetable description.
APROCER020 An error occurred when trying a WorkspaceServices mapping extension handler.
APROCER021 Missing source attribute in datastore entity.
APROCER022 An attribute in an expression is not valid.
APROCER023 Application builder configuration error.
APROCER024 Failed creating DataStoreMappingConfig, no name specified.
APROCER025 Failed creating DataStoreMappingConfig, the name is not unique.
APROCER026 The mapping to datastore had to be abandoned because the schema is not
 registered.
APROCER027 There was a problem parsing the Mapping Specification.
APROCER028 General mapping error. Mapping XML included.
APROCER029 Cannot have multiple primary participants.
APROCER030 No programs have been applied for.
APROCER031 An error occurred while attempting to map to Person data.
APROCER032 An error occurred while attempting to map to Relationship data.
APROCER033 An error occurred while creating Cases.
APROCER034 An error occurred while creating evidence.
APROCER035 No programs have been applied for.
APROCER036 An error occurred reading data from the datastore.
APROCER037 Specified case type does not exist.
APROCER038 Specified case type does not exist.
APROCER039 Duplicate SSN entered.
APROCER040 Duplicate SSN entered.
APROCER041 There was a problem with the workflow process.
APROCER042 No primary participant has been identified as part of the intake process.
```

The set of error codes that is returned by the application matches the set that is defined in the code table file *`CT_ApplicationProcessingError.ctx`*. The range of codes that are reserved for internal processing is APROCER001 – APROCER500. This means that customers can use the range APROCER501 – APROCER999 if they want to log errors in custom processing, for example, in an extension mapping handler.

You can find the error messages that appear in the logs in the following message files:

- *`EJBServer\components\WorkspaceServices\message\Mapping.xml`*
- *`EJBServer\components\WorkspaceServices\message\Mapping.xml`*

Handling validation issues in data mapping

Use this information to learn about how to handle validation issues with your data mappings.

As part of the intake and screening process, a balance must be struck between validating evidence so that it can be inserted sensibly, while also ensuring that the citizen is not asked unnecessary questions. One approach is to ensure that during intake, the evidence is created with minimal validations, and defaulted or temporary values that can be updated later.

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.

Privacy policy

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

Trademarks