IBM Cúram Social Program Management
Version 7.0.2

Using the Data Mapping Engine

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 30

Contents

List of Figures

List of Tables

Chapter 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 Cúram Data Mapping Engine by using mappings that are configured in the Cúram Data Mapping Editor. Mappings are created for a program on an intake application.

Introduction

Purpose

The purpose of this guide is to describe how to use the Cúram Data Mapping Engine (CDME) to map citizen data captured during intake and screening processing to either:

- Evidence and non-evidence entities within Cúram cases, or
- Filled-out Application forms in PDF format

Both options streamline the citizen's application for benefit programs. Mapping data to case evidence allows that data to be used to determine eligibility as part of Cúram case processing. Mapping data to PDF application forms speeds up the process of completing these forms for manual submission.

This guide should be used in conjunction with the IBM Cúram Data Mapping Editor Guide. The Data Mapping Editor is the simplest tool for rapidly creating mappings to evidence. The Data Mapping Editor saves these mappings in an XML mapping language. This guide describes the details of the XML mapping language which is useful for understanding how existing mappings are executed, for understanding more advanced mappings, for creating mapping to PDF forms and for maintaining older mappings that may not be compatible with the data mapping editor.

Audience

Data mapping is a collaborative project between business analysts and developers. The role of business analysts 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 writing mapping specifications and mapping configurations in XML.

Prerequisites

The following table lists the prerequisites for the Data Mapping Engine and provides recommended reading to gain knowledge about these prerequisites:

Table 1: Prerequisites for Cúram Data Mapping Engine		
Prerequisite	**Recommended Reading**	**Target Audience**
XML	http://www.w3.org/XML/	Developers
Cúram Cases	Case entity descriptions in the Core reference model; `Cúram Integrated Case Management Guide`	Business Analysts
	`Inside Eligibility and Entitlement Using Cúram Express Rules`	Developers
Cúram Participants	Participant entity descriptions in the Core reference model; `Cúram Participant Guide`	Business Analysts

1

Table 1: Prerequisites for Cúram Data Mapping Engine (continued)		
Prerequisite	**Recommended Reading**	**Target Audience**
Common Datastore (CDS)	`Creating Datastore Schemas`	Developers
IBM Cúram Universal Access	`Universal Access Customization Guide`	Developers
Cúram Evidence	Evidence entity descriptions in the Core reference model; Cúram Evidence Guide	Business Analysts
	`Designing Cúram Evidence Solutions`	Developers
PDF Forms	http://www.adobe.com/products/acrobat/?promoid=BPDDU	Developers

Chapters in this Guide

The following chapters are in this guide:

Chapter 2 - Understanding Data Mapping
This chapter describes how the Cúram Data Mapping Engine converts citizen data (captured in the Universal Access Portal and stored in the Cúram Datastore) to fields on PDF application forms or evidence entities within cases.

Chapter 3 - Writing Mapping Specifications and Configurations for Static Evidence and PDFs
This chapter describes how to write mapping specifications and mapping configurations in XML for static evidence and pdfs. Mapping specifications map data from one form to another, e.g., from entities in the CDS structure to entities in the database. Mapping configurations describe how to populate PDF applications or convert data to evidence entities.

Chapter 4 - Writing Mapping Specifications and Configuration for Dynamic Evidence
This chapter describes how to write mapping specifications and mapping configurations in XML for dynamic evidence.

Chapter 5 - Mapping to Third Parties
This chapter describes how to tackle the mapping of Third Parties as Case Participants onto cases.

Appendix A - Schema for Mapping Specifications
This appendix defines the grammar to follow when writing mapping specifications in XML.

Appendix B - Schema for Mapping Configurations
This appendix defines the grammar to follow when writing mapping configurations in XML.

Appendix C - Compliancy
This appendix describes how to develop in a compliant manner.

Understanding Data Mapping

Introduction

The Cúram Data Mapping Engine (CDME) works with the Cúram Universal Access Portal to help citizens screen themselves for benefits. During the screening and intake process, a citizen will submit his or her information. CDME is responsible for converting the citizen's information into either a PDF application form (filled out with citizen's details) or a Cúram case with new evidence entities.

The CDME works as follows. When the citizen submits his or her information, that information gets stored in the Cúram Datastore (CDS). CDME reads the data and uses rules from a mapping specification to transform the data into something readable by an application builder. A PDF application builder uses

mapping configuration to determine how the citizen's data will appear on a PDF application form. An evidence application builder uses a mapping configuration to call on the evidence API in order to create the new evidence entities for the new case.

Understanding How Data Is Stored in the CDS

When mapping citizen data to case evidence or PDF application forms, the first task is to examine the desired output for the citizen's data, i.e., as case evidence or a completed application form. The next step is to examine how a citizen's data is stored in the CDS as part of the intake and screening process. Once you know the information required in case evidence or PDF application forms and the information stored in the CDS, you can then create logical maps between the forms of data.

The purpose of the screening and intake process is to help citizens apply for benefits. At the very minimum, an application form or case evidence will require the names of the members in a household and their relationships to the person applying for benefits. IEG scripts capture this information about a citizen and store it in the CDS according a pre-defined schema. displays an example of a data structure in the CDS for a single benefit application and includes the members of a household and their relationships.
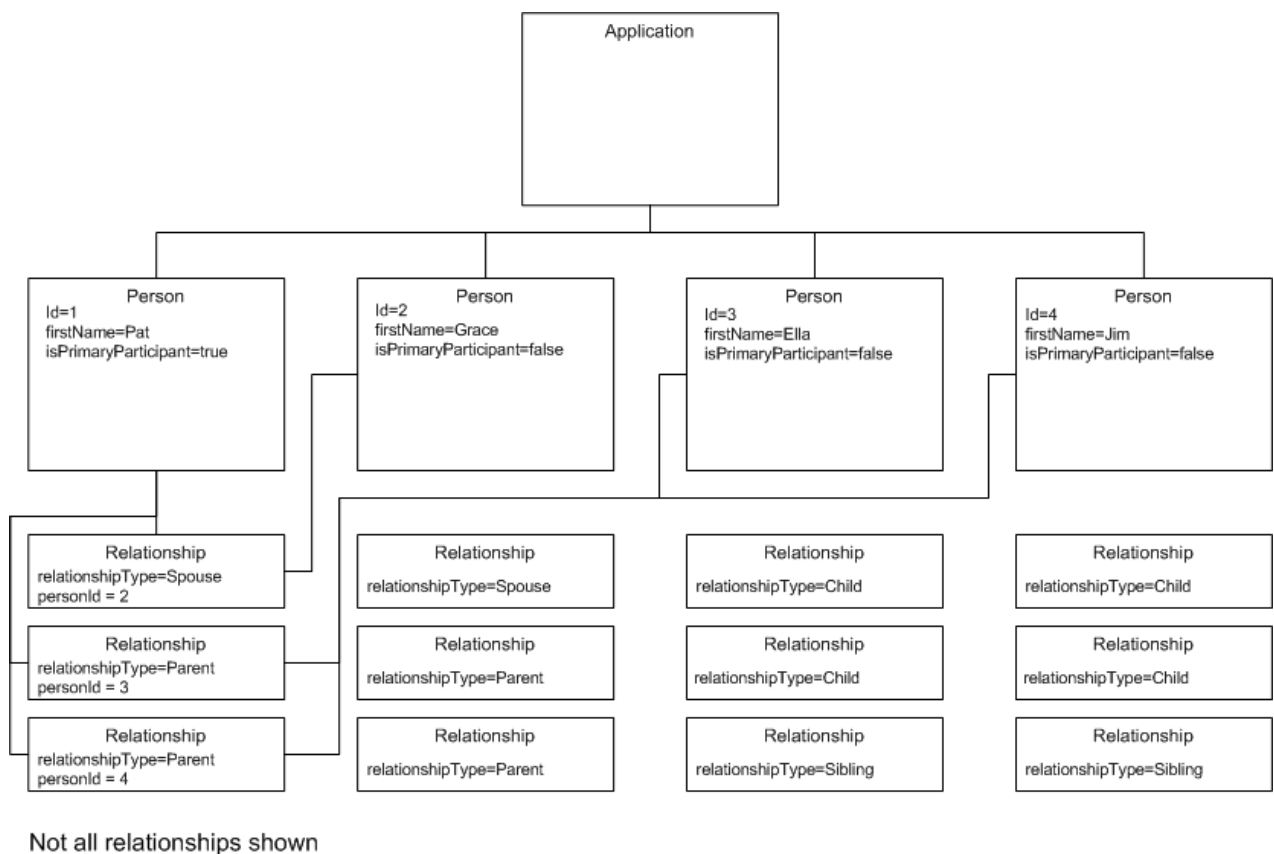
Figure 1: Example of Data Structure in Cúram Datastore

Creating Logical Maps

A logical map breaks down information about a person stored in CDS into the case evidence entities: household member, living arrangement, and disability.

Part of creating a logical map is to recognize business rules which may effect the way in which CDME maps data. For example, if the citizen indicates that he or she is blind and disabled, the business rules dictate that two disability evidence records should be created for the citizen (one for blindness and one for disability).

Considering Validation Issues in Data Mapping

As part of the intake and screening process, it is necessary to strike a balance between validating evidence so that it can be inserted sensibly while also insuring that the citizen is not asked unnecessary questions. One approach to considering validation issues in data mapping is to ensure that during intake, the evidence is created with minimal validations, defaulted or temporary values which can be updated later.

Writing Mapping Specifications and Configurations for Static Evidence and PDFs

Introduction

Developers can use the logical data map as the requirements specification for writing mapping specifications and mapping configurations. A mapping specification describes how to map data stored in a particular structure to a different one. Every mapping specification refers to a source, where the data is coming from, and a target, where the data is going to.

While the mapping specification contains the rules required to transform the data from one form to another, more information is required to turn the transformed data into case evidence or completed PDF application forms. This additional information is supplied in the mapping configuration. This chapter provides examples on how to write mapping specifications and mapping configurations.

Writing Mapping Specifications

Mapping specifications are used by the CDME to transform data in the CDS into another form. All the data 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. The Data Mapping Engine traverses the data in the CDS and applies rules expressed in XML to complete the transformation of data.

Simple Mapping Specification

This simple mapping specification maps a person entity in CDS to a household member evidence entity:

```
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 </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 <map-attribute> element specific. This is why line 6 states that the comments attribute in Person is mapped to the comments attribute in HouseholdMember.

In some cases, it is necessary to specify that a mapping only occurs under particular circumstances. For example, a HeadOfHousehold entity should 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 sample above 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>
```

Mapping Condition Expressions

In this example, the values Yes, No and Unanswered are represented as code table values and are used to record whether the person is a US Citizen or not. The value ITYN4001 corresponds to the client answering "Yes" to this question. Note the use of ", this is because the quote symbols "" cannot be used directly in XML. The syntax for conditionally mapping attributes is exactly the same.

```
1    <condition expression="Person.isBlind==&quot;ITYN4001&quot;">
2      <target-entity
3        name="Disability"
4        id="BlindDisabilityTarget"
5      >
6        <set-attribute
7          name="disabilityType"
8          value="DT1"
9        />
10     </target-entity>
11   </condition>
```

Mapping Code Tables Values

In some mappings, code table values recorded in the CDS can be translated directly into codetable values used in the target model. In these cases, a section at the beginning of the mapping script can be used to specify the codetable mappings. For example:

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

In Sample 5, values from the CITIZENSTATUS codetable are being mapped to values in the AlienStatus codetable.

Mapping to Multiple Target Entities

Sometimes it is necessary to create a group of Target Entities together. This is usually done when creating a group of evidence entities, one of which is a parent and the others are children of that parent evidence entity. See the sample below for an example of how to create groups of related target entities.

```
    1
    <target-entities>
```

```
2      <target-entity
3        name="BusinessAsset" id="BusinessAssetTarget"
4        type="parent"
5      >
6        <map-attribute
7          from="resourceAmount"
8          to="amount"
9        />
10       <map-attribute
11         from="amountOwed"
12         to="amountOwed"
13       />
14     </target-entity>
15     <target-entity
16       name="Ownership" id="OwnershipTarget"
17       type="child"
18     >
19       <set-attribute
20         name="percentageOwned"
21         value="100.0"
22       />
23     </target-entity>
24   </target-entities>
```

In this example, 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 is created by the Evidence Application Builder.

Matching One Parent Entity to Several Child Entities

In "Mapping to Multiple Target Entities" on page 5 , a group of target entities were created in which the child and parent entities were related to each other. In some cases, it is necessary to create only one parent entity for the whole case. All subsequent child entities are related to the same parent entity. An example of how to do this is shown below.

```
1 <target-entities>
2   <target-entity name="HholdMealsGroup" type="parent"
      attachment="case" id="MealGroup">
3     <set-attribute name="groupName" value="sample"/>
4   </target-entity>
5
6   <target-entity name="MealGroupMember" type="child"
      id="MealGroupMember">
7   </target-entity>
8 </target-entities>
```

In this example, a HholdMealsGroup and a MealGroupMember is 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 CDS

In the following scenario the customers have requested that the mapping engine be used to fill out a PDF form that looks something like this:

Table 2: 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 on this PDF form has a unique identity. For example, the field containing the name, Pat, is identified as Job0.Name. The field containing 30000 is identified as Job0.Salary.

Consider how the information from the intake might be stored in the CDS:
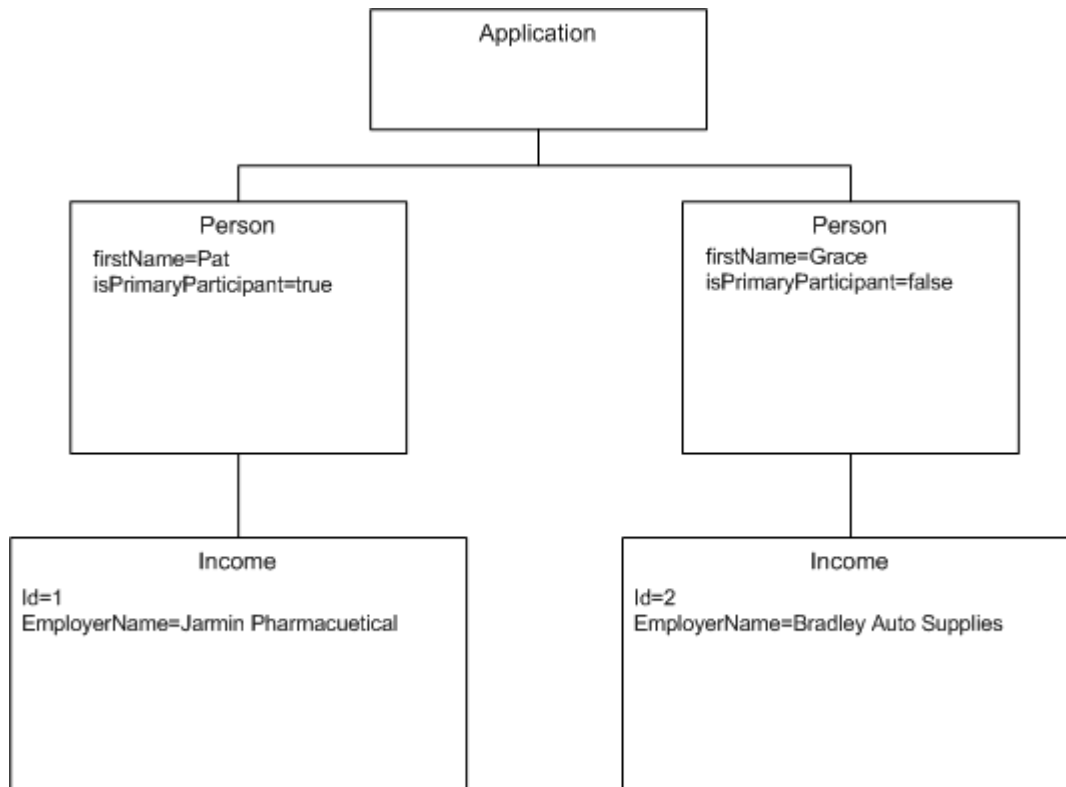
Figure 2: Job Income in CDS

In order to fill out the Name field on the on the above PDF form, the mapping specification must contain a rule which states that for each Income belonging to a Person, output the Person's firstName to the Name field. In the mapping language, this can be expressed as follows:

```
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>
```

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

Note the use of the syntax entity="Person" on line 4 to denote that the firstName attribute comes from 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 Members of a Household

The table below depicts how relationships are typically expressed in an application form. The requirement is to map the CDS entities to a pre-filled application form similar to the one shown below. The difficult part in this case is to fill in field titled "How is this Person Related to You?" This field goes by the shorthand "RelType" in this example.

Table 3: 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

In this example, the required mapping is written as follows:

```
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 contained within the primary participant, follow the association to the Person 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.

Writing Mapping Configurations

This section provides samples of mapping configurations for both the evidence application builder and the PDF application builder.

For the Evidence Application Builder

When configured to use the Evidence Application Builder, the work of the Mapping Engine is divided into two phases. In the first phase, the Mapping Engine creates an Integrated Case including Case Members and creates Concern Role Relationships between the Case Members. The Case Members are populated by finding Entities in the Data Store called "Person". The Data Mapping Engine is designed to treat all Common Data Store Entities called Person as referring to a Person or Prospect Person on a Cúram Case.

For example, in phase 1 of the mapping the Evidence Application Builder will create an integrated case with four case members where:

• 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 2 of the mapping process is concerned with creating evidence entities, samples of which are included in the remainder of this section.

Simple Mapping Configuration

The following example illustrates a configuration for the Evidence Application Builder:

```
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>
```

In this example, the Evidence Application Builder has been configured to create HouseholdMember and HeadOfHousehold evidence. 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.

All of the 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 as long as it follows the patterns used by the evidence generator.

Handling of caseParticipantDetails Fields

In order to execute the createHouseholdMemberEvidence() operation on the HouseholdMember, the CDME has to populate the caseParticipantDetails field of the HouseholdMemberEvidenceDetails struct, an extract of which is shown below:

```
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, by and large, filled out through the <set-attribute> and <map-attribute> elements in the Mapping Specification. For example, the following line in the mapping specification leads to the field natHawOrPaIsInd being populated 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 refer to this Case Participant. The Data Mapping Engine does this automatically whenever it finds a field called caseParticipantDetails on the EvidenceDetails struct. Sometimes, however, there are variations required in the handling of case

participants, for example, when the Evidence Details 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 this example, the Case Participant who owns the Annuity is referred to in the AnnuityCaseParticipantDetails struct 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 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. Using the following example, it is possible map the name and address of the institution holding the Annuity:

```
1  <target-entity name="Annuity" id="AnnuityTarget">
2   <map-attribute
3     from="institutionName"
4     to="participantName"
5   />
6   <map-attribute
7     from="institutionAddress"
8     to="address"
```

```
 9    />
10 </target-entity>
```

In some cases, it may be considered too onerous to ask the client to fill out all this kind of third party information as part of an online intake. Instead the Mapping Specification can be used to default these values and they can be filled out properly at the interview stage. Here is an example of 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>
```

The value curam.blankaddress on line 12 causes a blank address to be inserted for the participant.

Setting Target Entity Identifiers

On line 1 of Sample 13, as with a number of the previous samples the <target-entity> element has included an id attribute "AnnuityTarget". Although this attribute is optional, it is good practice to include an id in all <target-entity> elements. This 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 Data Store has two Boolean indicators: isBlind and hasDisability. Both map to the same target entity type, Disability, as follows:

```
 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, lines13-18, checks the hasDisability indicator and if it is set to true then 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.

For the PDF Application Builder

A PDF form contains a number of fields of various types. Each field has a unique name. The Cúram Workspace Services uses this unique name to reference the field so that it can write data to that field. In order to make this process work, the author of the PDF Form needs to name these fields and set properties of these fields according to certain conventions. If these conventions are followed, the PDF Application Builder will be able to map data to those fields.

Note: This section describes the custom PDF Application Builder, it allows mapping to a tailored PDF Form. Customers can also avail of the Generic PDF Builder, see the section 'How to Customize the Generic PDF for Processed Applications' in the Chapter Customizing the Handling of Submitted Applications of the Cúram Universal Access Customization Guide.

Sections and Fields

The most basic convention is that fields are grouped into "Sections". These Sections do not necessarily correspond to sections on the form but in many cases they will. For example:

```
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>
```

Sample 14 shows an extract from a PDF Application Builder Configuration. It refers to a section called applicant. Based on Sample 14 line 4, 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". This field is a radio button, whereas lines 10-14 all refer to fields that are checkbox buttons.

Filling Text Fields

Line 6 of Sample 14 refers to a plain standard text field. The corresponding mapping might look something like this:

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

Line 5 is somewhat 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 is appended to the current value of the text field rather than 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 will result in the field Applicant.Name being filled with the Applicants first name, middle initial and surname e.g. "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 typically look something like this:

```
<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 Figure 18. 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>
```

Repeated Sections and Code Table Descriptions

Some forms contain repeated sections, for example "List the details of all the people in your household" or "List all your sources of Income from Work". The PDF Application Builder is designed to deal with this provided the PDF Author names the fields according to the correct conventions. For example, fields used to collect data about household members might be named as follows:

Table 4: 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

The corresponding Mapping Configuration would be written as follows:

```
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, line 1 the attribute type="multiple" is what causes the section to be repeated. Note the codetable-class attribute on line 3 in this sample. This is a very useful attribute that causes codetable values to be translated into localized descriptions. By using it in the above context the script author ensures that the

second column 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 unchecked. The PDF Application Builder assumes that setting a checkbox field to the value "Yes" will cause it to be checked whereas setting it to "No" causes it to be unchecked. The PDF Form Author needs to ensure that this convention is adhered to. Whenever the Mapping Engine maps a boolean value to a checkbox field, it is automatically mapped as follows: True maps to "Yes" and false maps to "No".

Radio Buttons

A collection of Radio Buttons are treated as a single field in a PDF form. Only one item at a time can be selected by the Radio Buttons. The individual items in the Radio button are selected by writing a particular value to the radio button. The PDF Form author can determine which item is selected by specifying an "export value" for each item. A typical use for a Radio button with Cúram 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 codetable 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". The mapping looks like this:

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

The corresponding Mapping Configuration looks like this:

```
<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, imagine that the person designing the form wants to provide a drop down to represent Alien Status of a Person. Curam has an AlienStatus codetable which has codes corresponding to the following descriptions:

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

The PDF form designer, creates a Choice Combo and sets the item text for each item in the drop down to the values above. To ensure that the code table description rather than the code table code gets sent to the form, the following configuration is required:

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

How to configure the PDF Application Form

First load your PDF form into the Universal Access Portal. It's important to use a PDF *Form* not just a PDF document. The PDF must contain a form and the form must contain fields. If you want to use the PDF Application Builder, then each field on the form must have a unique name e.g. PersonalDetails.surname vs. Child1Details.surname. You should use a program like Adobe Acrobat Writer Professional or GlobalSCAPE CutePDF Pro to edit your form.

Note, if you're using Adobe Acrobat Writer Professional ensure that you save the form as an AcroForm not an XFA. You can upload your PDF by logging into Curam as Administrator, choosing the Universal Access Administration section and selecting PDF Forms. From here you can upload the form and give it a sensible name. Every Intake Application has an Intake Application Type so the next thing you need to do is ensure that your Intake Application Type is associated with the correct PDF Form. You can do this by going to Intake Applications, select the relevant Intake Application Type and selecting Edit. Click on the drop down for PDF Form and your newly loaded PDF Form should appear on this list. Select it. Next you need to specify PDF Mappings for the individual programs you are interested in.

Write a PDF Mapping xml and a PDF Application Configuration xml using the instructions in the previous section in this guide. Go to the Programs menu item in the Universal Access Admin section. A list of programs is displayed. View the program you are interested in. Select the Mapping tab on the top right. Create a new mapping. Make sure that you select "PDF Form Creation" instead of Evidence Creation. Upload the mapping configuration file and mapping specification file. Test your mapping by performing an intake for the relevant program. At the end of the Intake, select the link to display the PDF file.

Writing Mapping Specifications and Configurations for Dynamic Evidence

Introduction

The purpose of this document is to map citizen data captured during the intake process into dynamic evidence entities.

Prerequisites:

It is assumed that you have knowledge of the basic concepts of dynamic evidence. In particular, it is assumed that you have a good understanding of evidence nature, dynamic evidence type definition, evidence version definition and XML meta data of dynamic evidence.

Writing Mapping Specifications and Configurations for Dynamic Evidence

Dynamic evidence by it's nature is not modeled. There is single entity (or a set of entities) that contain data for all dynamic evidence types. A Dynamic Evidence Type will have one or more Evidence Type Versions over its lifetime. At any point in time, only one Evidence Type Version will be effective. All metadata elements (Attributes, Relationships, etc.) will be defined at the Evidence type Version level. So, the developer is responsible for providing the right mapping specification and configurations for the currently effective evidence type version.

Simple Dynamic Evidence Metadata

This simple metadata represents the structure for Household Member dynamic evidence type. The different types of attributes such as Boolean, Codetable, Date and String are defined.below:

```
<?xml version="1.0" encoding="UTF-8"?>
<EvidenceTypeVersion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file://DynamicEvidenceMetadata.xsd">
    <Model>
        <Attributes>
            <Attribute>
                <DataAttribute name="blkOrAfrAmerInd">
                    <DomainType dataType="Boolean" />
                </DataAttribute>
```

```
                    </Attribute>
                    <Attribute>
                        <DataAttribute name="ssnStatus">
                            <DomainType dataType="Codetable">
                                <CodetableOptions codetableName=
                                        "SSNApplicationStatus" />
                            </DomainType>
                        </DataAttribute>
                    </Attribute>
                    <Attribute>
                        <DataAttribute name="startDate">
                            <DomainType dataType="Date" />
                        </DataAttribute>
                    </Attribute>
                    <Attribute>
                        <DataAttribute name="endDate">
                            <DomainType dataType="Date" />
                        </DataAttribute>
                    </Attribute>
                    <Attribute>
                        <DataAttribute name="comments">
                            <DomainType dataType="String" />
                        </DataAttribute>
                    </Attribute>
                </Attributes>
            </Model>
            <Validations>
                <PatternValidations>
                </PatternValidations>
            </Validations>
            <UserInterface />
        </EvidenceTypeVersion>
```

Simple Mapping Specification

This simple mapping specification maps data from HouseHoldMember entity in a DataStore to a HouseHoldMember dynamic evidence entity defined in the previous section.

Please note that the attribute name mentioned in the 'to' field must match the 'name' field of 'DataAttribute' element of dynamic evidence metadata. In other words, a developer should make sure that the attribute to which the data will be mapped are present in the metadata of that 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

The following simple mapping configuration is defined for HouseHoldMember dynamic evidence type.:

```xml
<?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>
```

A developer must specify dynamic evidence type code in the 'ev-type-code' attribute. This is to enable the system to determine whether the evidence is static or dynamic. If this attribute is left blank or 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

This section describes how a mapping would be done for evidences which have a parent-child relationship.

Simple Parent-Child Dynamic Evidence metadata

The following metadata has attributes for the Adoption dynamic evidence type. This Adoption metadata has two attributes which describes that this entity has two related CaseParticipant fields. The field "caseParticipantRoleID" is a primary CaseParticipantRole and "parCaseParticipantRoleID" is an associated CaseParticipantRole.

```xml
<?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 Relationship with its parent dynamic evidence.

```xml
<?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 mapping specification has parent-child relationship and there are few attributes defined for adoption entity and which all are used to create a new participant. The values for these attributes are read from IBM 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 here.The very important thing to note here is that a new attribute 'dyn-evidence-primary-cpr-field-name' has been added into

<entity> element. The purpose of this attribute is that developer needs to specify the attribute name of the primary CaseParticipantRole defined in metadata. In this example, 'caseParticipantRoleID' is the primary CaseParticipantRole which was defined in the Adoption entity. Similarly, the related CaseParticipantRole attribute name ('parCaseParticipantRoleID' in this example) is defined in 'name' field of <create-participant> element. Note: In the case of static evidence, this same 'name' field of <create-participant> element is being used to mention the corresponding aggregation name.

```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="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>
```

Mapping to Third Parties

Introduction

Universal Access Intake allows users to enter details about their circumstances by using intelligent evidence gathering (IEG) scripts. The IEG script inserts the client's details into a data store. Following submission, the data store contents are mapped into Evidence Data for an Intake Case.

Many types of Evidence reference third parties – these third parties must be inserted onto the case as new Case Participants with their own unique Case Participant Role. For example, a Pregnancy record can have an associated Father. If the Father is absent, they can be recorded That Father as a Prospect Person Case Participant. In another example, Student evidence needs to be associated with a School. A 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 process of Intake for the assigned Caseworkers who processes the Case.

The mapping of addresses is made necessary by the need to associate a new or existing participant with a new piece of evidence. The participant is either a representative or a prospect person. One of the particular challenges in creating new participants is in the mapping of addresses. Address fields that are stored in the data store, such as ADD1 must be correctly aggregated into a correctly formatted Cúram Address structure to ensure that the participant can be properly created.

You will so in this chapter that the logic of creating new participant and mapping an address to that participant is isolated from the Evidence Application Builder.

How to Map Third Parties

The Application Builder Configuration schema includes elements and attributes which can be used to create a new participant and map an address to that participant.

Participant Creator Definition

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. The one important thing to be noted here is that the data type of all attributes mentioned here must be defined 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>
```

Create Participant

The <create-participant> element has been added into <entity> element. This element instructs the ApplicationBuilder to create a 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

The following is the schema to follow when writing mapping specification. Note that in the Education entity the attributes mapped directly to the 'Student' evidence entity. Attributes such as schoolName, schoolStreet1, schoolStreet2, etc will be 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

The following is the configuration XML to follow when writing 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>
```

Schema for Mapping Specifications

Schema

The following is 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"
              maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="source" type="xs:NCName"/>
    </xs:complexType>

    <xs:element name="map-entity" type="mp:MapEntityType"/>

    <xs:element name="follow-association" type="mp:MapEntityType"/>

    <xs:complexType name="MapEntitiesType">
        <xs:sequence>
            <xs:element ref="mp:target-entity" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="target-entities" type="mp:MapEntitiesType"/>

    <xs:complexType name="MapCodeTableValueType">
        <xs:attribute name="source" type="xs:string"/>
        <xs:attribute name="target" type="xs:string"/>
    </xs:complexType>
```

```
    <xs:complexType name="MapCodeTableType">
        <xs:sequence>
            <xs:element name="map-value"
              type="mp:MapCodeTableValueType" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="context" type="xs:NCName" use="optional"/>
        <xs:attribute name="source-codetable" type="xs:NCName"/>
        <xs:attribute name="target-codetable" type="xs:NCName"/>
        <xs:attribute name="source-package" type="xs:NCName"
use="optional"/>
        <xs:attribute name="target-package" type="xs:NCName"
use="optional"/>
    </xs:complexType>

    <xs:complexType name="MapType">
        <xs:sequence>
            <xs:element name="map-code-table" type="mp:MapCodeTableType"
              minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="mp:map-entity" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:NCName"/>
        <xs:attribute name="from-schema" type="xs:NCName"/>
        <xs:attribute name="to-schema" type="xs:NCName"/>
    </xs:complexType>

    <xs:element name="map" type="mp:MapType"/>

</xs:schema>
```

Schema for Mapping Configurations

Schema

The following is the schema to follow when writing mapping configurations:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Licensed Materials - Property of IBM

  Copyright IBM Corporation 2012. All Rights Reserved.

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

<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">
```

```xml
        <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>

    <xs:complexType name="EvidenceConfigType">
            <xs:sequence>
                <xs:element name="entity" type="abc:EntityType"

 maxOccurs="unbounded"/>
                <xs:element

name="def-create-participant" type=
                    "abc:ParticipantCreatorDefinitionType"

 minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="package" type="xs:string" use="optional"/>
    </xs:complexType>

    <xs:simpleType name="MutliplicityType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="multiple"/>
            <xs:enumeration value="singleton"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="FieldType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="append"/>
            <xs:enumeration value="button-radio"/>
            <xs:enumeration value="button-checkbox"/>
            <xs:enumeration value="choice-combo"/>
            <xs:enumeration value="choice-multi-list"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="FieldConfig">
        <xs:attribute name="name" type="xs:NCName"/>
        <xs:attribute name="type" type="abc:FieldType" use="optional"/>
        <xs:attribute name="append-separator" type="xs:string" use="optional"/>
        <xs:attribute name="codetable-class" type="xs:string" use="optional"/>
    </xs:complexType>

    <xs:complexType name="SectionConfig">
        <xs:sequence>
            <xs:element name="field" type="abc:FieldConfig"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:NCName" use="required"/>
        <xs:attribute name="type" type="abc:MutliplicityType" use="optional"/>
    </xs:complexType>

    <xs:complexType name="PdfConfigType">
            <xs:sequence>
                <xs:element name="section" type="abc:SectionConfig"

maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="pdf-form-name" type="xs:string"/>
```

```
        </xs:complexType>

        <xs:complexType name="DatastoreConfigType">
            <xs:attribute name="targetSchema" type="xs:string"/>
        </xs:complexType>

        <xs:element name="evidence-config" type="abc:EvidenceConfigType"/>

        <xs:element name="pdf-config" type="abc:PdfConfigType"/>

        <xs:element name="datastore-config" type="abc:DatastoreConfigType"/>

        <xs:complexType name="ApplicationBuilderConfigType">
            <xs:choice>
              <xs:element ref="abc:evidence-config"/>
              <xs:element ref="abc:pdf-config"/>
              <xs:element ref="abc:datastore-config"/>
            </xs:choice>
        </xs:complexType>
        <xs:element name="application-builder-config"

 type="abc:ApplicationBuilderConfigType"/>
</xs:schema>
```

Error logging and diagnostics

Error codes

During mapping development, unexpected errors can occur. You can configure the application to increase the level of error logging.

The application property

```
curam.workspaceservices.application.processing.logging.on
```

can be set to true to increase the granularity of these messages.

The list of error codes (and their codetable description) that the mappings engine returns is as follows:

```
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.
```

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

The actual error messages that appear in the logs can be found in the following message files: EJBServer \components\WorkspaceServices\message\Mapping.xml, EJBServer\components\WorkspaceServices \message\WorkspaceServicesDataMapping.xml

Notices

This information was developed for products and services offered in the United States.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Privacy Policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user's name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Programming Interface Information

This publication documents intended programming interfaces that allow the customer to write programs to obtain the services of IBM Cúram Social Program Management.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at " Copyright and trademark information " at http://www.ibm.com/legal/copytrade.shtml.

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

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

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

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

IBM®

Part Number:

(1P) P/N: