IBM Cúram Social Program Management
Version 6.0.5

# *Cúram Person and Prospect Person Evidence Developers Guide*

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 35

**Revised: March 2014**

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

# Contents

# Figures

# Tables

# Developing with Person and Prospect Person Evidence

Use this information to design a person/prospect person evidence solution. This work involves a consideration of the data, its structure, evidence constraints, and the flow of the data around the system. Some of the information that is stored about persons and prospect persons is held as evidence, which can be shared between cases on the system.

## Introduction

### Purpose

The purpose of this guide is to provide a high level technical understanding of person/prospect person evidence and its components. This guide also outlines the available customization options and extension points and provides instructions on how to implement these customizations.

### Audience

This guide is intended for developers and architects intending to implement a person/prospect person evidence solution.

**Important:** This guide is only applicable to those readers that are using the participant application with person and prospect person dynamic evidence.

### Pre-requisites

The document assumes that the reader is familiar with the following.

- *Cúram Evidence Guide*
- *Cúram Participant Guide*
- *Cúram Dynamic Evidence Configuration Guide*
- *Google Guice*

### Chapters in this Guide

The following list describes the chapters within this guide:

**Person/Prospect Person Evidence Overview**
This chapter provides a high level overview of the key technical aspects of person/prospect person evidence.

**Designing Person/Prospect Person Evidence Solutions**
This chapter outlines some design considerations that should be taken into account when designing a person/prospect person evidence solution.

**Dynamic Evidence Type Data Mappings**
This chapter describes the mapping of data from the dynamic evidence types supplied with the application to legacy database tables.

**Customizing Person/Prospect Person Evidence**
This chapter describes the customization options and extension points available for person/prospect person evidence.

# Person/Prospect Person Evidence Overview

## Person/Prospect Person Data as Evidence

Some of the information stored about persons and prospect persons is held as evidence, which can be shared between cases on the system. A number of Cúram components and technologies come together to enable the storing of person and prospect person evidence and the flow of this evidence through the system:

- Cúram Dynamic Evidence is used to store the captured person/prospect person data and perform basic validations.
- Cúram Express Rules are used to execute complex validations against the captured data.
- The Cúram Evidence Broker can optionally be used to broker the data between cases.
- Cúram Verifications can optionally be used to apply verifications to the captured data when it is brokered between cases.

Depending on the business requirements, some level of configuration, customization or both may be required. This chapter describes at a high level how the system manages person/prospect person data and identifies the points at which configuration and/or customization might be required.

**Note:** Careful consideration should be given to the required business behavior of the system during the design phase of a Cúram implementation. The starting point for developing an understanding of how a system should be configured to support the business requirements should be the Cúram Participant Guide and the Cúram Evidence Guide.

## How is Person/Prospect Person Evidence Managed?

The management of person and prospect person data as evidence is underpinned by the following key foundations:

- Each person and prospect person has an associated person or prospect person case (Participant Data Case) created 'under the hood' following registration.
- Person and prospect person data is stored as evidence on dynamic evidence tables and is described by the dynamic evidence types which are associated with the person and/or prospect person.
- The data recorded as evidence is replicated back to the legacy database tables; the legacy database tables should therefore be in sync with the dynamic evidence.
- When editing a person or prospect person record, the data is retrieved from, and written to, the dynamic evidence tables (and again, replicated back to the legacy database tables).
- In some cases, application screens and processing will still read from the legacy database tables.
- The person and prospect person case types can be configured to have participant data brokered to and from other cases, using the Cúram Evidence Broker.

### Person/Prospect Person Evidence Types

A number of dynamic evidence types are provided and are associated with the person and prospect person participants. These evidence types and their attributes must not be removed or disassociated from the person and prospect person.

Where there is a requirement to manage additional data as person and prospect person evidence, new evidence types can be created as described in the Cúram Dynamic Evidence Configuration Guide and associated with the person and prospect person in the administration component. Equally, new attributes can be added to the existing dynamic evidence types. Where the data being added to new or existing evidence types is already present on an existing legacy database table, additional customization work must be performed:

- The code that replicates the data from the dynamic evidence tables to the legacy database tables (the 'replicator') must be extended to replicate the additional data being stored as evidence.
- Where data already exists on the legacy database tables, this data must be copied to the equivalent dynamic evidence table(s). The code that performs that operation (the 'converter') must be extended to convert the additional data into evidence.

More detail and example implementations for both of these are provided in the chapter 'Customizing Person/Prospect Person Evidence'.

## Evidence Validations

Person and prospect person evidence is validated when created and edited. These validations are implemented in one of two ways:

- Using the Dynamic Evidence Editor validations functionality. For example, mandatory field validations.
- Using Cúram Express Rule Sets. For example, cross-evidence validations.

Where new dynamic evidence types or attributes are added, customers should use one of these mechanisms to add any validations required. This is described in more detail in the Cúram Dynamic Evidence Configuration Guide.

When evidence is brokered to the person and prospect person, these validations are not checked. Brokered evidence is always accepted to prevent it being lost, as there is no concept of 'incoming evidence' for a person and prospect person. When person/prospect person evidence is entered, it is validated immediately. However, evidence brokered in from another case is automatically accepted and activated, even if the validation checks fail. For other case types, when person/prospect person evidence is brokered onto the case, a validation failure will prevent the evidence from being activated.

## Evidence Sharing

The evidence framework provides the ability to share evidence between a person/prospect person, application cases and ongoing cases. The Evidence Broker enables and mediates this sharing of evidence. Evidence sharing is uni-directional and per evidence type. This means that different targets can receive and share an evidence type in different ways. If required, one case type might be able to receive shared evidence, but might not be able to share its own evidence. There are three main functions which will trigger the evidence broker to broadcast evidence:

- When a new person is added to a target case. For example, where person/prospect person evidence such as identification evidence has been configured for sharing to an integrated case and a person is added to an integrated case, the evidence broker will first check to see if that person has any person/prospect person evidence. If evidence is found, the evidence broker then checks for active identification evidence and shares it to the integrated case.
- When evidence changes are made to a source case. For example, when changes are made to a person's identification evidence, the evidence broker will share those changes to the integrated case.

- When a new target case is created. For example, any time a new integrated case is created, the evidence broker will search for person/prospect person identification evidence to be shared. If this evidence is found, the evidence broker shares the identification evidence to the integrated case.

For more detailed information on the Evidence Broker, see the Cúram Evidence Broker Guide.

# Designing Person/Prospect Person Evidence Solutions

When designing a person/prospect person evidence solution the designer should consider the data, its structure, constraints and the flow of that data around the system.

## Data: Dynamic Evidence Types

### Structure

Person/Prospect Person evidence is primarily stored as dynamic evidence and the data structures that represent it are dynamic evidence types. Dynamic evidence types define the data, its type and behaviour such as volatility, calculated attributes etc. Once new dynamic evidence types have been defined they must be activated and associated with the relevant case types, persons and prospect persons. Further information on how to define dynamic evidence types can be found in the Cúram Dynamic Evidence Configuration Guide.

Things to consider:

- Does the evidence vary over time?
- Is the evidence type reciprocal? If so, the evidence type should have participant and related participant attributes.
- What case types should the evidence be available on?
- Consider making the evidence type 'Preferred', if it is to be commonly used. It will allow case workers to quickly create evidence for frequently recorded evidence types.

### Constraints

**Validations:** A number of standard validations, frequently used in evidence processing, are provided in the Dynamic Evidence Editor. More complex validations, such as cross-evidence validations, can be included using Cúram Express Rules. More information on validations can be found in the Cúram Dynamic Evidence Configuration Guide.

Things to consider:

- What validations are required to ensure integrity of data?
- When person/prospect person evidence is entered, it is validated immediately; however, evidence brokered in from another case is automatically accepted and activated, even if the validation checks fail. For other case types, when person/prospect evidence is brokered onto the case, a validation failure will prevent the evidence from being activated.
- Include any validations required to enforce succession constraints.
- Try to use standard validation patterns where possible. Validation rule sets should only be developed if they cannot be implemented using standard validations.

- If developing validation rule sets for more complex validations, be mindful of how data retrieval is performed. If performed incorrectly, this can have a significant impact on performance.

**Important:** System processes rely on the validations shipped with the application and it is not compliant to remove or alter these validations.

**Verifications:** This section is only applicable for those readers licensed to use the verifications component.

Verification is the process of checking the accuracy of evidence. The verification of evidence can take a number of forms; it can be provided by documents, e.g. birth certificates or bank statements, or by verbal means, e.g. telephone calls. When evidence is captured, the verification engine is invoked in order to determine if the evidence requires verification.

**Note:** With the exception of evidence brokered to a person/prospect person record, evidence cannot be activated until all mandatory verification requirements have been met.

For more information on verifications and their configuration, please see the Cúram Verification Guide.

Things to consider:

- Does the evidence require verification?
- What are the rules around verification?
- What information needs to be provided by the client?

## Flow: Evidence Broker

The Evidence Broker is the mechanism that allows evidence to be shared throughout the system. When the Evidence Broker broadcasts evidence to a person/prospect person record, the evidence is automatically accepted and activated on the person/prospect person record, so the user does not have to manually accept and activate evidence. For more information on Evidence Broker and configuration options, please see the Cúram Evidence Broker Guide. For the recommended brokering approach please see the Cúram Evidence Guide.

Things to consider:

- Is the same evidence type used on more than one case type? If so, should changes to this evidence be communicated to other cases?
- Should the target case be set up to automatically accept changes or should the case worker be forced to intervene to decide on whether to accept this incoming evidence?
- In order for system processing to function correctly, it is essential that person/prospect person data recorded outside of the participant manager be shared back to the participant manager.

## Cúram Express Rules: Case Eligibility/Entitlement Calculations

Areas where Cúram Express Rules are used to read participant data from legacy database tables for the purposes of case eligibility and entitlement calculations, should be analyzed in order to decide where this data should be sourced from. There are three options, each of which have their own benefits and limitations:

- Read participant data from the dynamic evidence stored by the participant manager
- Read participant data which has been brokered onto cases
- Continue to read from the legacy tables

### Read participant data from the dynamic evidence stored by the participant manager

Things to consider:

- Working off primary data source
- Changes in evidence causes immediate recalculations
- No opportunity for case worker to review

### Read participant data which has been brokered onto cases

Things to consider:

- This is the recommended option for any new development
- Changes will only effect when evidence is activated
- Evidence type has to be configured to be brokered onto the case

### Continue to read from the legacy tables

Things to consider:

- This option should be considered carefully and is only recommended for upgrading customers.

## Dynamic Evidence Type Data Mappings

The tables below show the data mappings from the dynamic evidence types to the legacy database tables.

**Note:** Replicators perform this mapping and converters perform the reverse mapping.

## Address

*Table 1. Address Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRoleAddress.concernRoleID (calculated using caseParticipantRoleID) |
| address | Address.addressData |
| fromDate | ConcernRoleAddress.startDate |
| toDate | ConcernRoleAddress.endDate |
| addressType | ConcernRoleAddress.typeCode |
| comments | ConcernRoleAddress.comments |

## Bank Account

*Table 2. Bank Account Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRoleBankAccount.concernRoleID (calculated using caseParticipantRoleID) |
| accountName | BankAccount.name |

*Table 2. Bank Account Mapping (continued)*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| accountNumber | BankAccount.accountNumber |
| iban | BankAccount.iban |
| accountType | BankAccount.typeCode |
| sortCode | BankAccount.bankSortCode |
| bic | BankAccount.bic |
| fromDate | BankAccount.startDate |
| toDate | BankAccount.endDate |
| accountStatus | BankAccount.bankAccountStatus |
| jointAccountInd | BankAccount.jointAccountInd |
| comments | BankAccount.comments |

## Birth and Death

*Table 3. Birth and Death Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| person | Person/ProspectPerson.concernRoleID (calculated using caseParticipantRoleID) |
| birthLastName | Person/ProspectPerson.personBirthName |
| mothersBirthLastName | Person/ProspectPerson.motherBirthSurname |
| dateOfBirth | Person/ProspectPerson.dateOfBirth |
| dateOfDeath | Person/ProspectPerson.dateOfDeath |
| comments | N/A |

## Contact Preferences

*Table 4. Contact Preferences Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRole.concernRoleID (calculated using caseParticipantRoleID) |
| preferredLanguage | ConcernRole.preferredLanguage |
| preferredCommunication | ConcernRole.prefCommMethod |
| comments | N/A |

## Email Address

*Table 5. Email Address Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRoleEmailAddress.concernRoleID (calculated using caseParticipantRoleID) |
| emailAddress | EmailAddress.emailAddress |
| fromDate | ConcernRoleEmailAddress.startDate |
| toDate | ConcernRoleEmailAddress.endDate |

*Table 5. Email Address Mapping (continued)*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| emailAddressType | ConcernRoleEmailAddress.typeCode |
| comments | EmailAddress.comments |

## Gender

*Table 6. Gender Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| person | Person/ProspectPerson.concernRoleID (calculated using caseParticipantRoleID) |
| gender | Person/ProspectPerson.gender |
| comments | N/A |

## Identification

*Table 7. Identification Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRoleAlternateID.concernRoleID (calculated using caseParticipantRoleID) |
| alternateID | ConcernRoleAlternateID.alternateID |
| altIDType | ConcernRoleAlternateID.typeCode |
| fromDate | ConcernRoleAlternateID.startDate |
| toDate | ConcernRoleAlternateID.endDate |
| comments | ConcernRoleAlternateID.comments |

## Name

*Table 8. Name Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | AlternateName.concernRoleID (calculated using caseParticipantRoleID) |
| title | AlternateName.title |
| firstName | AlternateName.firstForename |
| middleName | AlternateName.otherForename |
| lastName | AlternateName.surname |
| suffix | AlternateName.nameSuffix |
| initials | AlternateName.initials |
| nameType | AlternateName.nameType |
| comments | AlternateName.comments |

## Phone Number

*Table 9. Phone Number Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRolePhoneNumber.concernRoleID (calculated using caseParticipantRoleID) |
| phoneCountryCode | PhoneNumber.phoneCountryCode |
| phoneAreaCode | PhoneNumber.phoneAreaCode |
| phoneNumber | PhoneNumber.phoneNumber |
| phoneExtension | PhoneNumber.phoneExtension |
| fromDate | ConcernRolePhoneNumber.startDate |
| toDate | ConcernRolePhoneNumber.endDate |
| phoneType | ConcernRolePhoneNumber.typeCode |
| comments | PhoneNumber.comments |

## Relationship

*Table 10. Relationship Mapping*

| Dynamic Evidence Attribute | Database Column |
|---|---|
| participant | ConcernRoleRelationship.concernRoleID (calculated using caseParticipantRoleID) |
| relatedParticipant | ConcernRoleRelationship.relConcernRoleID |
| fromDate | ConcernRoleRelationship.startDate |
| toDate | ConcernRoleRelationship.endDate |
| relationshipType | ConcernRoleRelationship.relationshipType |
| endReason | ConcernRoleRelationship.relEndReasonCode |
| comments | ConcernRoleRelationship.comments |

## Snapshot Tables

When person/prospect person data is registered or maintained, this data will **not** be replicated to the following snapshot tables

- AlternateNameSnapshot
- ConcernRoleAddressSnapshot
- ConcernRoleAlternateIDSnapshot
- ConcernRoleBankAccountSnapshot
- ConcernRoleRelSnapshot
- ConcernRoleSnapshot
- PersonSnapshot
- ProspectPersonSnapshot

# Customizing Person/Prospect Person Evidence

## Introduction

This chapter describes the customization options and extension points available for person/prospect person evidence. Some or all of these may be applicable to you depending on your existing customizations and configurations. There are five main areas to consider, listed below:

- Replicators
- Converters
- Selection of Primary Information
- Reciprocal Evidence
- Participant Data Case Owner

Each of these areas are described in detail and examples are also provided. **Please note, these are samples only.**

## Replicators

### What is a Replicator?

A replicator reflects changes in evidence onto the relevant legacy tables for the purposes of backward compatibility. The replicator takes the dynamic evidence details and converts them to a struct containing the details to be stored on the legacy tables. These details are then written to the relevant database tables, thus ensuring that the information on the legacy tables is in sync with the primary data source, the dynamic evidence. Default replicator implementations are provided for each of the person/prospect person evidence types. These default implementations contain extension points to allow replication to custom fields, which is covered in the following section.

**Note:** Only the last version in a succession set is used to replicate data to the legacy tables.

### Why Extend a Replicator?

In cases where legacy database tables have been extended, it may be necessary to extend a replicator.

### Replicator Extension

It is possible to extend the replicators supplied with the application, to allow replication of person/prospect person evidence to custom database columns. Interfaces are available for each supplied evidence type and can be found in the package curam.pdc.impl, listed below. Custom implementations can be written that make use of these interfaces, depending on the evidence type.

Replicator Extender Interfaces:

- PDCAddressReplicatorExtender
- PDCAlternateIDReplicatorExtender
- PDCAlternateNameReplicatorExtender
- PDCBankAccountReplicatorExtender
- PDCBirthAndDeathReplicatorExtender
- PDCContactPreferencesReplicatorExtender
- PDCEmailAddressReplicatorExtender

- PDCGenderReplicatorExtender
- PDCPhoneNumberReplicatorExtender
- PDCRelationshipsReplicatorExtender

The majority of the interfaces have one method `assignDynamicEvidenceToExtendedDetails`. It accepts two parameters:

- dynamicEvidenceDataDetails - the dynamic evidence details
- details - the struct containing the extended details for the legacy table

`PDCBirthAndDeathReplicatorExtender` and `PDCGenderReplicatorExtender` have two methods, `assignDynamicEvidenceToExtendedPersonDetails` and `assignDynamicEvidenceToExtendedProspectPersonDetails`. `assignDynamicEvidenceToExtendedPersonDetails` accepts two parameters:

- dynamicEvidenceDataDetails - the dynamic evidence details
- details - the struct containing the extended person details for the legacy table

`assignDynamicEvidenceToExtendedProspectPersonDetails` also accepts two parameters:

- dynamicEvidenceDataDetails - the dynamic evidence details
- details - the struct containing the extended prospect person details for the legacy table

## Example: Implementing a Person/Prospect Person Evidence Replicator Extender

The following example outlines how to extend a replicator to map person/prospect person evidence to custom fields. This example provides a very basic implementation of an extension to the `PDCPhoneNumberReplicatorExtender`. In this scenario, the PhoneNumber table has been extended and contains a custom field 'phoneProvider'. The dynamic evidence configuration for Phone Number also contains an attribute 'phoneProvider'. This example assumes that the struct `ParticipantPhoneDetails` has already been extended to include the custom field. For more information on dynamic evidence configuration, please see the Cúram Dynamic Evidence Configuration Guide. The responsibility of the custom replicator extension implementation is to map the dynamic evidence data to the struct attribute that represents the 'phoneProvider' attribute on the extended PhoneNumber table.

**Note:** A mapping of data is all that is necessary; the default implementation performs the actual replication of data.

The steps involved in extending a replicator are:

- Provide a replicator extension implementation that will map the custom data back to the legacy table
- Add a binding to the new replicator extension implementation

**Step 1: Provide a Replicator Extension Implementation:** The first step is to provide a new implementation that implements the relevant replication extension interface for the evidence type and maps the custom data back to the legacy table. The code snippet below demonstrates a custom implementation for `PDCPhoneNumberReplicatorExtender`. It simply assigns the value of the dynamic evidence attribute to the phoneProvider struct attribute. This information will then be inserted along with the other dynamic evidence attributes via the default implementation for `PDCPhoneNumberReplicatorExtender`.

```
public class SampleReplicatorExtenderImpl
  implements PDCPhoneNumberReplicatorExtender {

  public void assignDynamicEvidenceToExtendedDetails(
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails,
    ParticipantPhoneDetails details)
    throws AppException, InformationalException {

    details.phoneProvider =
    dynamicEvidenceDataDetails.getAttribute("phoneProvider").getValue();
  }
}
```

**Step 2: Add a Binding to the New Replicator Extension Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Register the replicator extension implementation
  Multibinder<PDCPhoneNumberReplicatorExtender>
sampleReplicatorExtender =
  Multibinder.newSetBinder(binder(),
PDCPhoneNumberReplicatorExtender.class);

  sampleReplicatorExtender.addBinding().
to(SampleReplicatorExtenderImpl.class);
 }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

## Why Implement a Replicator?

In cases where new dynamic evidence types have been introduced, it may be necessary to implement a new replicator.

## Implementing a Replicator

Replicators can be easily developed to cater for scenarios such as new dynamic evidence types. A detailed example is provided in the next section and outlines the steps and artifacts necessary to get a new replicator up and running. Replicator implementations are invoked via an event based mechanism. When dynamic evidence is activated after an insert, modify or remove, an event is thrown. For new evidence types an event listener needs to be developed to listen for this event and invoke the replication process, this will be discussed in more detail later in this chapter. The next section demonstrates how to implement a replicator.

## Example: Implementing a Person/Prospect Person Evidence Replicator

The following example outlines how to implement a replicator. In this scenario, Sample Foreign Residency has been configured as a new dynamic evidence type. For more information on how to configure a new evidence type, please see the Cúram Dynamic Evidence Configuration Guide. The new Sample Foreign Residency evidence type has the following attributes,

- participant - the case participant role id of the person/prospect person that the evidence is being entered for
- country - the country of residency
- fromDate - the date the residency started

- toDate - the date the residency ended
- reason - the reason for residency in this country

It has been assumed that this dynamic evidence type has been activated and is configured for use with person/prospect person. Until now Sample Foreign Residency information has been stored as static evidence on the SampleForeignResidency database table. It is now necessary to store this information as dynamic evidence. A new replicator may be required to replicate evidence changes to the legacy database table so that this table is in sync with the dynamic evidence.

The steps involved in implementing a replicator are:

- Provide a replicator interface for the dynamic evidence type
- Provide a replicator implementation that will replicate dynamic evidence to the legacy database table
- Implement an event listener that will trigger the replication
- Add a binding to the new event listener implementation

**Step 1: Provide a Replicator Interface:** The new replicator interface should contain three methods -

`replicateInsertEvidence` which replicates activated inserted Sample Foreign Residency evidence to the Sample Foreign Residency legacy database table. It accepts one parameter:

- evidenceDescriptorDtls the activated evidence descriptor details

`replicateModifyEvidence` which replicates activated modified Sample Foreign Residency evidence to the Sample Foreign Residency legacy database table. It accepts two parameters:

- evidenceDescriptorDtls the activated evidence descriptor details
- previousActiveEvidDescriptorDtls the evidence descriptor details for the evidence that was active before the modify

`replicateRemoveEvidence` which replicates activated removed Sample Foreign Residency evidence to the Sample Foreign Residency legacy database table. It accepts one parameter:

- evidenceDescriptorDtls the activated evidence descriptor details

```
@ImplementedBy(SampleForeignResidencyReplicatorImpl.class)
public interface SampleForeignResidencyReplicator {

  public void replicateInsertEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException;

  public void replicateModifyEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls,
    final EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException;

  public void replicateRemoveEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException;
}
```

**Step 2: Provide a Replicator Implementation:** The replicator implementation should provide implementations for the three methods described in the previous

section. These methods should convert the dynamic evidence data to data suitable to be written to the legacy database tables and update the legacy tables for this evidence type.

```
public class SampleForeignResidencyReplicatorImpl
  implements SampleForeignResidencyReplicator {

  protected SampleForeignResidencyReplicatorImpl() {
  }

  public void replicateInsertEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    SampleForeignResidency sampleForeignResidencyObj =
   SampleForeignResidencyFactory.newInstance();
    SampleForeignResidencyDtls sampleForeignResidencyDtls =
   new SampleForeignResidencyDtls();
    UniqueID uniqueIDObj = UniqueIDFactory.newInstance();

    EvidenceControllerInterface evidenceControllerObj =
      (EvidenceControllerInterface) EvidenceControllerFactory
.newInstance();

    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
    eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType = evidenceDescriptorDtls.
evidenceType;

    EIEvidenceReadDtls eiEvidenceReadDtls =
      evidenceControllerObj.readEvidence(eiEvidenceKey);

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails) eiEvidenceReadDtls.
evidenceObject;

    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
      dynamicEvidenceDataDetails.getAttribute("reason")
.getValue();

    sampleForeignResidencyDtls.concernRoleID =
evidenceDescriptorDtls.participantID;
    sampleForeignResidencyDtls.foreignResidencyID =
uniqueIDObj.getNextID();
    sampleForeignResidencyDtls.statusCode =
RECORDSTATUS.NORMAL;

    sampleForeignResidencyObj.insert(sampleForeignResidencyDtls);
  }

  public void replicateModifyEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls,
    final EvidenceDescriptorDtls
previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException {
```

```
    List<SampleForeignResidencyKey> sampleForeignResidencyKeyList =
      new ArrayList<SampleForeignResidencyKey>();

    SampleForeignResidencyDtls sampleForeignResidencyDtls =
   new SampleForeignResidencyDtls();

    EvidenceControllerInterface evidenceControllerObj =
      (EvidenceControllerInterface)
EvidenceControllerFactory.newInstance();

    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
    eiEvidenceKey.evidenceID =
previousActiveEvidDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType =
previousActiveEvidDescriptorDtls.evidenceType;

    EIEvidenceReadDtls eiEvidenceReadDtls =
      evidenceControllerObj.readEvidence(eiEvidenceKey);

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails)
eiEvidenceReadDtls.evidenceObject;

    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.
getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
   dynamicEvidenceDataDetails.getAttribute("reason").getValue();

    SampleForeignResidency sampleForeignResidencyObj =
   SampleForeignResidencyFactory.newInstance();

    SampleForeignResidencyReadMultiKey
sampleForeignResidencyReadMultiKey =
      new SampleForeignResidencyReadMultiKey();
    sampleForeignResidencyReadMultiKey.concernRoleID =
   previousActiveEvidDescriptorDtls.participantID;

    SampleForeignResidencyReadMultiDtlsList
sampleForeignResidencyReadMultiDtlsList =
      sampleForeignResidencyObj.searchByConcernRole
(sampleForeignResidencyReadMultiKey);

    for (SampleForeignResidencyReadMultiDtls
sampleForeignResidencyReadMultiDtls :
   sampleForeignResidencyReadMultiDtlsList.dtls) {

      if ((sampleForeignResidencyReadMultiDtls.countryCode.equals(
     sampleForeignResidencyDtls.countryCode))
        && (sampleForeignResidencyReadMultiDtls.reasonCode.equals(
  sampleForeignResidencyDtls.reasonCode))) {

        SampleForeignResidencyKey sampleForeignResidencyKey =
new SampleForeignResidencyKey();
        sampleForeignResidencyKey.sampleForeignResidencyID =
    sampleForeignResidencyReadMultiDtls.sampleForeignResidencyID;

        sampleForeignResidencyKeyList.add(sampleForeignResidencyKey);
```

```
      }
    }

    for (SampleForeignResidencyKey sampleForeignResidencyKey
 : sampleForeignResidencyKeyList) {

      sampleForeignResidencyDtls = new SampleForeignResidencyDtls();

      eiEvidenceKey = new EIEvidenceKey();
      eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
      eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

      eiEvidenceReadDtls = evidenceControllerObj.readEvidence(eiEvidenceKey);

      dynamicEvidenceDataDetails =
        (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;

      sampleForeignResidencyDtls.countryCode =
        dynamicEvidenceDataDetails.getAttribute("country").getValue();

      sampleForeignResidencyDtls.startDate = (Date)
DynamicEvidenceTypeConverter.convert(
        dynamicEvidenceDataDetails.getAttribute("fromDate"));

      sampleForeignResidencyDtls.endDate = (Date)
DynamicEvidenceTypeConverter.convert(
        dynamicEvidenceDataDetails.getAttribute("toDate"));

      sampleForeignResidencyDtls.reasonCode =
        dynamicEvidenceDataDetails.getAttribute("reason").getValue();

      sampleForeignResidencyDtls.concernRoleID =
evidenceDescriptorDtls.participantID;

      SampleForeignResidencyDtls sampleForeignResidencyReadDtls =
     sampleForeignResidencyObj.read(sampleForeignResidencyKey);

      sampleForeignResidencyReadDtls.assign(sampleForeignResidencyDtls);

      sampleForeignResidencyObj.modify(sampleForeignResidencyKey,
 sampleForeignResidencyReadDtls);
    }
  }

  public void replicateRemoveEvidence(
    final EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    List<SampleForeignResidencyKey> sampleForeignResidencyKeyList =
   new ArrayList<SampleForeignResidencyKey>();

    SampleForeignResidencyDtls sampleForeignResidencyDtls =
   new SampleForeignResidencyDtls();

    EvidenceControllerInterface evidenceControllerObj =
      (EvidenceControllerInterface) EvidenceControllerFactory.newInstance();

    EIEvidenceKey eiEvidenceKey = new EIEvidenceKey();
    eiEvidenceKey.evidenceID = evidenceDescriptorDtls.relatedID;
    eiEvidenceKey.evidenceType = evidenceDescriptorDtls.evidenceType;

    EIEvidenceReadDtls eiEvidenceReadDtls =
      evidenceControllerObj.readEvidence(eiEvidenceKey);

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
      (DynamicEvidenceDataDetails) eiEvidenceReadDtls.evidenceObject;
```

```
    sampleForeignResidencyDtls.countryCode =
      dynamicEvidenceDataDetails.getAttribute("country").getValue();

    sampleForeignResidencyDtls.startDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("fromDate"));

    sampleForeignResidencyDtls.endDate =
   (Date) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails.getAttribute("toDate"));

    sampleForeignResidencyDtls.reasonCode =
   dynamicEvidenceDataDetails.getAttribute("reason").getValue();

    SampleForeignResidency sampleForeignResidencyObj =
   SampleForeignResidencyFactory.newInstance();

    SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
      new SampleForeignResidencyReadMultiKey();
    sampleForeignResidencyReadMultiKey.concernRoleID =
   evidenceDescriptorDtls.participantID;

    SampleForeignResidencyReadMultiDtlsList
 sampleForeignResidencyReadMultiDtlsList =
      sampleForeignResidencyObj.
searchByConcernRole(sampleForeignResidencyReadMultiKey);

    for (SampleForeignResidencyReadMultiDtls
 sampleForeignResidencyReadMultiDtls :
   sampleForeignResidencyReadMultiDtlsList.dtls) {

      if ((sampleForeignResidencyReadMultiDtls.countryCode.equals(
     sampleForeignResidencyDtls.countryCode))
        && (sampleForeignResidencyReadMultiDtls.reasonCode.equals(
  sampleForeignResidencyDtls.reasonCode))) {

        SampleForeignResidencyKey sampleForeignResidencyKey
 = new SampleForeignResidencyKey();
        sampleForeignResidencyKey.sampleForeignResidencyID =
    sampleForeignResidencyReadMultiDtls.sampleForeignResidencyID;

        sampleForeignResidencyKeyList.add(sampleForeignResidencyKey);
      }
    }

    for (SampleForeignResidencyKey sampleForeignResidencyKey
 : sampleForeignResidencyKeyList) {

      sampleForeignResidencyDtls = sampleForeignResidencyObj.
read(sampleForeignResidencyKey);
      sampleForeignResidencyDtls.statusCode
 = RECORDSTATUS.CANCELLED;
      sampleForeignResidencyObj.modify(sampleForeignResidencyKey,
 sampleForeignResidencyDtls);
    }
  }
}
```

**Step 3: Implement an Event Listener:** A new event listener needs to be implemented to listen for events raised of type Sample Foreign Residency that occur as a result of evidence activation. This listener should implement the interface curam.pdc.impl.PDCEvents and provide implementations for the three methods. This is where the replication process can be kicked off as well as any other custom processing that may need to happen.

```
public class SampleForeignResidencyEventsListener
  implements PDCEvents {

  @Inject
  private SampleForeignResidencyReplicator
sampleForeignResidencyReplicator;

  public void insertedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    if (evidenceDescriptorDtls.evidenceType.equals
("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateInsertEvidence
(evidenceDescriptorDtls);
    }
  }

  public void modifiedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls,
    EvidenceDescriptorDtls previousActiveEvidDescriptorDtls)
    throws AppException, InformationalException {

    if (evidenceDescriptorDtls.evidenceType.equals
("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateModifyEvidence
(evidenceDescriptorDtls,
      previousActiveEvidDescriptorDtls);
    }
  }

  public void removedEvidenceActivated(
    EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

 if (evidenceDescriptorDtls.evidenceType.equals
("SAMPLEFOREIGNRESIDENCY")) {
      sampleForeignResidencyReplicator.replicateRemoveEvidence
(evidenceDescriptorDtls);
    }
  }
}
```

**Step 4: Add a Binding to the New Event Listener Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Register the event listener
    Multibinder<PDCEvents> sampleEventListeners =
      Multibinder.newSetBinder(binder(), PDCEvents.class);

    sampleEventListeners.addBinding().to(
      SampleForeignResidencyEventsListener.class);
  }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

# Converters

## What is a Converter?

A converter is a mechanism for converting legacy person/prospect person data to dynamic evidence. When legacy database tables have been populated external to the application, through the use of tools such as the Cúram Data Manager(DMX files), converters can be used to convert this data to dynamic evidence. Default converter implementations are provided for each of the person/prospect person evidence types. These default implementations contain extension points to allow conversion of custom fields, which is covered in the following section.

## Why Extend a Converter?

In cases where legacy database tables have been extended, it may be necessary to extend a converter. Converters are generally only used in a development environment or for upgrade tooling and should not be used as part of everyday processing.

## Converter Extension

The converters provided with the application can be extended to allow conversion of custom database columns to person/prospect person dynamic evidence. Interfaces are available for each evidence type and can be found in the package curam.pdc.impl, these are listed below. Custom implementations can be written that make use of these interfaces, depending on the evidence type.

Converter Extension (Populator) Interfaces:

- PDCAddressEvidencePopulator
- PDCAlternateIDEvidencePopulator
- PDCAlternateNameEvidencePopulator
- PDCBankAccountEvidencePopulator
- PDCBirthAndDeathEvidencePopulator
- PDCContactPreferencesEvidencePopulator
- PDCEmailAddressEvidencePopulator
- PDCGenderEvidencePopulator
- PDCPhoneNumberEvidencePopulator
- PDCRelationshipsEvidencePopulator

The majority of the interfaces have one method `populate`. It accepts varying parameters depending on the evidence types.

`PDCBirthAndDeathEvidencePopulator` and `PDCGenderEvidencePopulator`, interfaces have two methods, `populatePerson` and `populateProspectPerson`.

`populatePerson` accepts four parameters:

- concernRoleKey - unique identifier for the concern role that this evidence is relating to
- caseIDKey - the unique identifier of the Participant Data Case
- personDtls - the struct containing the extended person details from the legacy table
- dynamicEvidenceDataDetails - the dynamic evidence details

`populateProspectPerson` also accepts four parameters:

- concernRoleKey - unique identifier for the concern role that this evidence is relating to
- caseIDKey - the unique identifier of the Participant Data Case
- prospectPersonDtls - the struct containing the extended prospect person details from the legacy table
- dynamicEvidenceDataDetails - the dynamic evidence details

## Example: Implementing a Person/Prospect Person Evidence Populator

The following example outlines how to extend a converter to map custom database columns to person/prospect person evidence. This example provides a very basic implementation of an extension to `PDCPhoneNumberEvidencePopulator`. In this scenario, the PhoneNumber table has been extended and contains a custom column 'phoneProvider'. The dynamic evidence configuration for Phone Number also contains an attribute 'phoneProvider'. The responsibility of the custom populator implementation is to convert the struct attribute that represents the 'phoneProvider' attribute on the extended PhoneNumber table to dynamic evidence data. Fore more information on dynamic evidence configuration, please see the Cúram Dynamic Evidence Configuration Guide.

**Note:** The conversion of data is all that is necessary, the default converters will look after the actual storage of the dynamic evidence.

The steps involved in extending a converter are:

- Provide a populator implementation that will convert the custom field from the legacy table to dynamic evidence data
- Add a binding to the new populator implementation

**Step 1: Provide a Populator Implementation:** The first step is to provide a new implementation that implements the relevant populator interface for the evidence type and converts the custom field from the legacy table to dynamic evidence. The code snippet below demonstrates the custom implementation for the `PDCPhoneNumberEvidencePopulator`, it simply converts the phoneProvider struct attribute to the dynamic evidence equivalent attribute. This dynamic evidence will then be stored along with the other dynamic evidence attributes via the default converter implementation.

```
public class SamplePopulatorImpl
  implements
PDCPhoneNumberEvidencePopulator {

  public void populate(
    ConcernRoleKey concernRoleKey,
CaseIDKey caseIDKey,
    ConcernRolePhoneNumberDtls
concernRolePhoneNumberDtls,
    PhoneNumberDtls phoneNumberDtls,
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails)
    throws AppException, InformationalException {

    DynamicEvidenceDataAttributeDetails phoneProvider =
      dynamicEvidenceDataDetails.getAttribute("phoneProvider");

    DynamicEvidenceTypeConverter.setAttribute(phoneProvider,
      phoneNumberDtls.phoneProvider);
  }
}
```

**Add a Binding to the New Populator Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Register the populator implementation
  Multibinder<PDCPhoneNumberEvidencePopulator> samplePopulator =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberEvidencePopulator.class);

  samplePopulator.addBinding().to(SamplePopulatorImpl.class);
  }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

## Why Implement a Converter?

In cases where new dynamic evidence types have been introduced, it may be necessary to implement a new converter. Converters are generally only used in a development environment or for upgrade tooling and should not be used as part of everyday processing.

## Implementing a Converter

Converter implementations can be developed using the `PDCConverter` interface. The `PDCConverter` interface can be found in the curam.pdc.impl package. This interface has one method `storeEvidence`. It accepts two parameters:

- concernRoleKey - the unique identifier of the concern role
- caseIDKey - the unique identifier of the Participant Data Case.

The next section demonstrates how to implement a converter.

## Example: Implementing a Person/Prospect Person Evidence Converter

The following example outlines how to implement a converter. In this scenario, Sample Foreign Residency has been configured as a new dynamic evidence type. For more information on how to configure a new evidence type, please see the Cúram Dynamic Evidence Configuration Guide. The new Sample Foreign Residency evidence type has the following attributes:

- participant - the case participant role id of the person/prospect person that the evidence is being entered for
- country - the country of residency
- fromDate - the date the residency started
- toDate - the date the residency ended
- reason - the reason for residency in this country

It has been assumed that this dynamic evidence type has been activated and is configured for use with person/prospect person. Sample Foreign Residency information had previously been stored as static evidence on the SampleForeignResidency database table. It is now necessary to store this information as dynamic evidence. A new converter is required which will take this information from the legacy table and convert and store it as dynamic evidence.

The steps involved in implementing a converter are:

- Provide a converter implementation that will convert the legacy data to dynamic evidence
- Add a binding to the new converter implementation

**Step 1: Provide a Converter Implementation:** The code snippet below demonstrates the implementation for the PDCConverter. It retrieves all Sample Foreign Residency information for a person/prospect person from the legacy SampleForeignResidency table, converts this information to a dynamic evidence data structure and stores the resulting information.

```
public class SampleForeignResidencyConverterImpl
  implements PDCConverter {

  @Inject
  private EvidenceTypeDefDAO etDefDAO;

  @Inject
  private EvidenceTypeVersionDefDAO etVerDefDAO;

  public void storeEvidence(ConcernRoleKey concernRoleKey, CaseIDKey caseIDKey)
    throws AppException, InformationalException {

    PDCCaseIDCaseParticipantRoleID pdcCaseIDCaseParticipantRoleID =
      new PDCCaseIDCaseParticipantRoleID();

    ParticipantDataCase participantDataCaseObj =
ParticipantDataCaseFactory.newInstance();
    pdcCaseIDCaseParticipantRoleID.caseID =
      participantDataCaseObj.getParticipantDataCase(concernRoleKey).caseID;

    CaseIDTypeCodeKey caseIDTypeCodeKey = new CaseIDTypeCodeKey();
    caseIDTypeCodeKey.caseID = pdcCaseIDCaseParticipantRoleID.caseID;
    caseIDTypeCodeKey.typeCode = CASEPARTICIPANTROLETYPE.PRIMARY;

    pdcCaseIDCaseParticipantRoleID.caseParticipantRoleID =
      CaseParticipantRoleFactory.newInstance().readByCaseIDAndTypeCode
(caseIDTypeCodeKey).dtls.caseParticipantRoleID;

    SampleForeignResidency sampleForeignResidencyObj =
 SampleForeignResidencyFactory.newInstance();

    SampleForeignResidencyReadMultiKey sampleForeignResidencyReadMultiKey =
      new SampleForeignResidencyReadMultiKey();
    sampleForeignResidencyReadMultiKey.concernRoleID =
 concernRoleKey.concernRoleID;

    SampleForeignResidencyReadMultiDtlsList sampleForeignResidencyList =
      sampleForeignResidencyObj.
searchByConcernRole(sampleForeignResidencyReadMultiKey);

   for (SampleForeignResidencyReadMultiDtls
sampleForeignResidencyReadMultiDtls :
sampleForeignResidencyList.dtls) {

      final EvidenceTypeKey eType = new EvidenceTypeKey();
      eType.evidenceType = "SampleForeignResidency";

      EvidenceTypeDef evidenceType =
        etDefDAO.readActiveEvidenceTypeDefByTypeCode(eType.evidenceType);

      EvidenceTypeVersionDef evTypeVersion =
        etVerDefDAO.getActiveEvidenceTypeVersionAtDate(evidenceType,
 Date.getCurrentDate());

      DynamicEvidenceDataDetails dynamicEvidenceDataDetails =
        DynamicEvidenceDataDetailsFactory.newInstance(evTypeVersion);
```

```
      DynamicEvidenceDataAttributeDetails participant =
        dynamicEvidenceDataDetails.getAttribute("participant");

      DynamicEvidenceTypeConverter.setAttribute(participant,
        pdcCaseIDCaseParticipantRoleID.caseParticipantRoleID);

      DynamicEvidenceDataAttributeDetails country =
        dynamicEvidenceDataDetails.getAttribute("country");

      DynamicEvidenceTypeConverter.setAttribute(country,
        sampleForeignResidencyReadMultiDtls.countryCode);

      DynamicEvidenceDataAttributeDetails fromDate =
        dynamicEvidenceDataDetails.getAttribute("fromDate");

      DynamicEvidenceTypeConverter.setAttribute(fromDate,
sampleForeignResidencyReadMultiDtls.startDate);

      DynamicEvidenceDataAttributeDetails endDate =
        dynamicEvidenceDataDetails.getAttribute("toDate");

      DynamicEvidenceTypeConverter.setAttribute(endDate,
sampleForeignResidencyReadMultiDtls.endDate);

      DynamicEvidenceDataAttributeDetails reasonCode =
        dynamicEvidenceDataDetails.getAttribute("reason");

      DynamicEvidenceTypeConverter.setAttribute(reasonCode,
sampleForeignResidencyReadMultiDtls.reasonCode);

      EvidenceControllerInterface evidenceControllerObj =
        (EvidenceControllerInterface)
EvidenceControllerFactory.newInstance();

      EvidenceDescriptorInsertDtls evidenceDescriptorInsertDtls =
new EvidenceDescriptorInsertDtls();
      evidenceDescriptorInsertDtls.participantID =
 concernRoleKey.concernRoleID;
      evidenceDescriptorInsertDtls.evidenceType =
 eType.evidenceType;
      evidenceDescriptorInsertDtls.receivedDate =
 curam.util.type.Date.getCurrentDate();
      evidenceDescriptorInsertDtls.caseID =
 pdcCaseIDCaseParticipantRoleID.caseID;

      EIEvidenceInsertDtls eiEvidenceInsertDtls =
 new EIEvidenceInsertDtls();

      eiEvidenceInsertDtls.descriptor.
assign(evidenceDescriptorInsertDtls);
      eiEvidenceInsertDtls.descriptor.participantID =
 concernRoleKey.concernRoleID;
      eiEvidenceInsertDtls.descriptor.changeReason =
EVIDENCECHANGEREASON.REPORTEDBYCLIENT;
      eiEvidenceInsertDtls.evidenceObject =
 dynamicEvidenceDataDetails;

      evidenceControllerObj.insertEvidence(eiEvidenceInsertDtls);
    }
  }
}
```

**Step 2: Add a Binding to the New Converter Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

  public void configure() {

  // Register the converter implementation
  Multibinder<PDCConverter> sampleForeignResidencyConverter =
    Multibinder.newSetBinder(binder(), PDCConverter.class);

  sampleForeignResidencyConverter.addBinding().
to(SampleForeignResidencyConverterImpl.class);
  }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

## Selection of Primary Information

Legacy participant manager entities have the notion of primary indicators, where users are able to specify which bank account/phone number etc. represents the primary data when the evidence is created. This is not the case with dynamic evidence types. The user does not specify the primary record; instead there is an algorithm in the background that calculates which should be the primary record. These algorithms are based on a defined business strategy and can be modified, details of which are outlined in the following section.

### Why Change the Selection of Primary Information?

As the identification of the primary record is not user-driven, it may be necessary to modify this selection process, if the default business strategy is not preferable.

### Changing the Selection of Primary Information

The strategies that determine which data should be selected as primary information can be modified through the use of the default primary handler interfaces. An interface is defined for each dynamic evidence type supplied that has a primary identifier on its legacy table, found in the curam.pdc.impl package and are listed below.

Primary handler implementations are invoked via an event based mechanism. When dynamic evidence is activated after an insert, modify or remove operation, an event is thrown. For new evidence types an event listener needs to be developed to listen for this event and invoke the appropriate algorithm that will determine the primary data, this is discussed in more detail later in this chapter. The next section demonstrates how to implement a primary handler.

Primary Handler Interfaces:

- PDCPrimaryAddressHandler
- PDCPrimaryAlternateIDHandler
- PDCPrimaryAlternateNameHandler
- PDCPrimaryBankAccountHandler
- PDCPrimaryEmailAddressHandler
- PDCPrimaryPhoneNumberHandler

### Changing the Selection of Primary Information Example

The following example outlines how to implement a primary handler. In this scenario, the defined business strategy for selecting a primary phone number is to select the phone number with the latest start date.

The steps involved in implementing a primary handler are:

- Provide a primary handler implementation that will identify the primary record
- Add a binding to the new primary handler implementation

**Step 1: Provide a Primary Handler Implementation:** The first step is to provide a new implementation that implements the relevant primary handler interface for the evidence type and identifies the primary record. The code snippet below demonstrates an implementation for `PDCPrimaryPhoneNumberHandler`, it simply takes the phone number with the latest start date and sets it as the primary record.

```
public class SamplePrimaryPhoneNumberHandlerImpl
  implements PDCPrimaryPhoneNumberHandler {

  protected SamplePrimaryPhoneNumberHandlerImpl() {
  }

  public void setPrimaryPhoneNumber
(EvidenceDescriptorDtls evidenceDescriptorDtls)
    throws AppException, InformationalException {

    ConcernRoleKey concernRoleKey = new ConcernRoleKey();
    concernRoleKey.concernRoleID =
evidenceDescriptorDtls.participantID;

    ConcernRolePhoneNumberDtlsList concernRolePhoneNumberDtlsList =
      ConcernRolePhoneNumberFactory.newInstance().
searchByConcernRole(concernRoleKey);

    ConcernRole concernRoleObj = ConcernRoleFactory.newInstance();
    ConcernRoleDtls concernRoleDtls = concernRoleObj.read(concernRoleKey);
    Date currentPrimaryPhoneNumberStartDate = Date.kZeroDate;

    List<SampleSortedPhoneNumber> list =
new ArrayList<SampleSortedPhoneNumber>();

    for (ConcernRolePhoneNumberDtls
concernRolePhoneNumberDtls:concernRolePhoneNumberDtlsList.dtls) {

      PhoneNumberKey phoneNumberKey = new PhoneNumberKey();
      phoneNumberKey.phoneNumberID = concernRolePhoneNumberDtls.phoneNumberID;

      if (concernRolePhoneNumberDtls.phoneNumberID ==
concernRoleDtls.primaryPhoneNumberID) {
        currentPrimaryPhoneNumberStartDate = concernRolePhoneNumberDtls.startDate;
      }

      SampleSortedPhoneNumber sampleSortedPhoneNumber =
        new SampleSortedPhoneNumber(concernRolePhoneNumberDtls);
      list.add(sampleSortedPhoneNumber);
    }

    Collections.sort(list);

    SampleSortedPhoneNumber newPrimaryPhoneNumber = list.get(0);

    if (newPrimaryPhoneNumber.getStartDate().
after(currentPrimaryPhoneNumberStartDate)) {
      concernRoleDtls.primaryPhoneNumberID =
newPrimaryPhoneNumber.getPhoneNumberID();
      concernRoleObj.pdcModify(concernRoleKey, concernRoleDtls);
    }
}
  class SampleSortedPhoneNumber implements
Comparable<SampleSortedPhoneNumber> {
    private long phoneNumberID;
    private Date startDate;
```

```
    SampleSortedPhoneNumber(ConcernRolePhoneNumberDtls dtls) {
      this.phoneNumberID = dtls.phoneNumberID;
      this.startDate = dtls.startDate;
    }

    public long getPhoneNumberID() {
      return phoneNumberID;
    }

    public Date getStartDate() {
      return startDate;
    }

    public int compareTo(SampleSortedPhoneNumber o) {
      return o.getStartDate().compareTo(this.getStartDate());
    }
  }
}
```

**Step 2: Add a Binding to the New Primary Handler Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Register the primary handler implementation
    bind(PDCPrimaryPhoneNumberHandler.class).to(
      SamplePrimaryPhoneNumberHandlerImpl.class);
  }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

# Reciprocal Evidence

## What is Reciprocal Evidence?

Reciprocal evidence is a type of evidence which consists of two pieces of evidence that must be processed together. The relationship dynamic evidence type is an example of reciprocal evidence. When Person A is recorded as a spouse of Person B, the corresponding relationship evidence, Person B is a spouse of Person A is recorded. When evidence is inserted, modified or removed for Person A, the system inserts, modifies or removes the corresponding relationship evidence for Person B.

## Why Provide a Reciprocal Evidence Implementation?

If you develop a new reciprocal dynamic evidence type, then you must also provide an implementation of the `ReciprocalEvidenceConversion` interface.

## Reciprocal Evidence Implementations

When evidence is inserted, modified or removed a hook point is invoked, that by default triggers the reciprocal evidence handler functionality. This new evidence hook point is called the `GlobalEvidenceHook` and can be found in the curam.core.sl.infrastructure.impl package. The `GlobalEvidenceHook` Interface allows custom processing to occur after evidence operations have completed.

GlobalEvidenceHook Interface:

The GlobalEvidenceHook interface contains the following methods:

postInsertEvidence is invoked after evidence is inserted and accepts two parameters:

- caseKey - the identifier of the case that the evidence belongs to
- evKey - the identifier and type of the evidence

postModifyEvidence is invoked after evidence is modified and accepts two parameters:

- caseKey - the identifier of the case that the evidence belongs to
- evKey - the identifier and type of the evidence

postRemoveEvidence is invoked after evidence is removed and accepts two parameters:

- caseKey - the identifier of the case that the evidence belongs to
- evKey - the identifier and type of the evidence

postDiscardPendingUpdate is invoked after a pending update of evidence is discarded and accepts two parameters:

- caseKey - the identifier of the case that the evidence belongs to
- evKey - the identifier and type of the evidence

postDiscardPendingRemove is invoked after a pending remove of evidence is discarded and accepts two parameters:

- caseKey - the identifier of the case that the evidence belongs to
- evKey - the identifier and type of the evidence

Reciprocal Evidence Handler:

The default implementation for the GlobalEvidenceHook invokes the reciprocal evidence handler functionality. The reciprocal evidence handler is responsible for all common reciprocal evidence processing. It locates reciprocal evidence and if found performs the same changes on it that were performed on the original evidence. If the reciprocal evidence is not found, and the original evidence was inserted, then it will insert the corresponding reciprocal evidence. As the reciprocal evidence handler is core to the reciprocal evidence processing it cannot be customized directly, but can be customized by way of the GlobalEvidenceHook, if necessary.

Reciprocal Evidence Conversion Interface:

The ReciprocalEvidenceConversion interface is responsible for reciprocal and original evidence comparison, participant retrieval and for creating new and modified reciprocal evidence from original evidence. In order to make custom evidence reciprocal, a ReciprocalEvidenceConversion interface implementation must be provided. While the handler is not aware of the internal evidence structure, the conversion interface implementation is, as a result this is where the main customization point lies. The ReciprocalEvidenceConversion interface can be found in the curam.core.sl.infrastructure.impl package and contains the following methods:

- Object getReciprocal(final Object original, final long targetCaseID) - Creates reciprocal evidence details from the original evidence details

- Object getUpdatedReciprocal(final Object original, final Object unmodifiedReciprocal) - Creates modified reciprocal evidence details from the original evidence details and from un-modified reciprocal evidence details
- long getPrimaryParticipant(final Object originalEvidence) - Retrieves the primary participant (concern role ID) from the original evidence. Note that the primary participant on the original evidence is the related participant on the reciprocal evidence
- long getRelatedParticipant(final Object originalEvidence) - Retrieves related participant (concern role ID) from the original evidence. Note that the related participant on the original evidence is the primary participant on the reciprocal evidence
- boolean matchEvidenceDetails(final Object evidenceDetails1, final Object evidenceDetails2) - Checks evidence details for a match. Implementation of this method determines if two evidence details passed in are considered as a match.

The following section demonstrates how to implement reciprocal evidence.

## Reciprocal Evidence Implementation Example

The following example demonstrates a reciprocal evidence implementation. In this scenario, Working Relationship has been configured as a new dynamic evidence type. For more information on how to configure a new evidence type, please see the Cúram Dynamic Evidence Configuration Guide. The new Working Relationship evidence type has been identified as reciprocal and has the following attributes,

- participant - the case participant role id of the person/prospect person that the evidence is being entered for
- relatedParticipant - the case participant role id of the related person/prospect person
- workingRelationship - the working relationship between the two participants

It has been assumed that this dynamic evidence type has been activated and is configured for use with person/prospect person. In order for this reciprocal evidence to be handled correctly by the infrastructure, an implementation of the `ReciprocalEvidenceConversion` interface must be provided.

The steps involved are:

- Provide a reciprocal evidence conversion implementation
- Add a binding to the new reciprocal evidence conversion implementation

**Step 1: Provide a Reciprocal Evidence Conversion Implementation:**

```
public class SampleWorkingRelationshipReciprocalConversion
  implements ReciprocalEvidenceConversion {

  @Inject
  private EvidenceTypeDefDAO etDefDAO;

  @Inject
  private EvidenceTypeVersionDefDAO etVerDefDAO;

  public SampleWorkingRelationshipReciprocalConversion() {

  }

  public Object getReciprocal(
    final Object original, final long targetCaseID)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
```

```
 (DynamicEvidenceDataDetails) original;

    String workingRelationshipOriginal =
      originalDetails.getAttribute
("workingRelationship").getValue();

    String workingRelationshipRec = "";

    if (workingRelationshipOriginal.equals("ISMANAGEROF")) {
      workingRelationshipRec = "ISMANAGEDBY";
    }

    EvidenceTypeKey evdType = new EvidenceTypeKey();
    evdType.evidenceType = "WORKINGRELATIONSHIP";

    EvidenceTypeDef evdTypeDef =
      etDefDAO.readActiveEvidenceTypeDefByTypeCode
(evdType.evidenceType);

    EvidenceTypeVersionDef evTypeVersion =
      etVerDefDAO.getActiveEvidenceTypeVersionAtDate
(evdTypeDef, Date.getCurrentDate());

    DynamicEvidenceDataDetails reciprocalDetails =
      DynamicEvidenceDataDetailsFactory.newInstance
(evTypeVersion);

    DynamicEvidenceDataAttributeDetails workingRelationshipAttr =
      reciprocalDetails.getAttribute("workingRelationship");

    DynamicEvidenceTypeConverter.setAttribute
(workingRelationshipAttr, workingRelationshipRec);

    DynamicEvidenceDataAttributeDetails participantAttr =
      reciprocalDetails.getAttribute("participant");

    DynamicEvidenceTypeConverter.setAttribute(participantAttr,
        originalDetails.getAttribute
("relatedParticipant").getValue());

    DynamicEvidenceDataAttributeDetails relatedParticipantAttr =
      reciprocalDetails.getAttribute("relatedParticipant");

    DynamicEvidenceTypeConverter.setAttribute
(relatedParticipantAttr,
        originalDetails.getAttribute("participant").getValue());

    return reciprocalDetails;
  }

  public Object getUpdatedReciprocal(
    final Object original, final Object unmodifiedReciprocal)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) original;
    DynamicEvidenceDataDetails reciprocalDetails =
      (DynamicEvidenceDataDetails) unmodifiedReciprocal;

    long caseParticipantRoleIDRec = Long.parseLong(
      reciprocalDetails.getAttribute("participant").getValue());
    long relCaseParticipantRoleIDRec = Long.parseLong(
      reciprocalDetails.getAttribute
("relatedParticipant").getValue());
    String workingRelationshipRec =
reciprocalDetails.getAttribute("workingRelationship").getValue();
```

```
    for (final DynamicEvidenceDataAttributeDetails
listDetails: originalDetails.getAttributes()) {
      reciprocalDetails.getAttribute(listDetails.getName()).setValue(
        listDetails.getValue());
    }

    DynamicEvidenceDataAttributeDetails workingRelationshipAttr =
      reciprocalDetails.getAttribute("workingRelationship");

    DynamicEvidenceTypeConverter.setAttribute
(workingRelationshipAttr, workingRelationshipRec);

    DynamicEvidenceDataAttributeDetails participantAttr =
      reciprocalDetails.getAttribute("participant");

    DynamicEvidenceTypeConverter.setAttribute(participantAttr,
      caseParticipantRoleIDRec);

    DynamicEvidenceDataAttributeDetails relatedParticipantAttr =
      reciprocalDetails.getAttribute("relatedParticipant");

    DynamicEvidenceTypeConverter.setAttribute(relatedParticipantAttr,
      relCaseParticipantRoleIDRec);

    return reciprocalDetails;
  }

  public boolean matchEvidenceDetails(
    final Object evidenceDetails1,
 final Object evidenceDetails2)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails dynamicEvidenceDataDetails1 =
      (DynamicEvidenceDataDetails) evidenceDetails1;
    DynamicEvidenceDataDetails dynamicEvidenceDataDetails2 =
      (DynamicEvidenceDataDetails) evidenceDetails2;

    curam.core.sl.intf.CaseParticipantRole caseParticipantRoleObj =
      curam.core.sl.fact.CaseParticipantRoleFactory.newInstance();

    CaseParticipantRoleKey caseParticipantRoleKey =
 new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID =
 (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails1.getAttribute("participant"));

    Long concernRoleID1 =
      caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID =
 (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails1.getAttribute("relatedParticipant"));

    Long relConcernRoleID1 =
      caseParticipantRoleObj.readCaseIDandParticipantID
(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID =
 (Long) DynamicEvidenceTypeConverter.convert(
      dynamicEvidenceDataDetails2.getAttribute("participant"));

    Long concernRoleID2 = caseParticipantRoleObj.
readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID =
 (Long) DynamicEvidenceTypeConverter.convert(
```

```
      dynamicEvidenceDataDetails2.getAttribute("relatedParticipant"));

    Long relConcernRoleID2 = caseParticipantRoleObj
.readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    return dynamicEvidenceDataDetails1.getAttribute
("workingRelationship").getValue().equals(
      dynamicEvidenceDataDetails2.getAttribute
("workingRelationship").getValue())
      && (concernRoleID1.longValue() ==
concernRoleID2.longValue())
      && (relConcernRoleID1.longValue() ==
 relConcernRoleID2.longValue());
  }

  public boolean matchOriginalAndReciprocal(
    final Object originalEvidence, final Object reciprocalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) originalEvidence;

    DynamicEvidenceDataDetails reciprocalDetails =
      (DynamicEvidenceDataDetails) reciprocalEvidence;

    curam.core.sl.intf.CaseParticipantRole caseParticipantRoleObj =
      curam.core.sl.fact.CaseParticipantRoleFactory.newInstance();
    CaseParticipantRoleKey caseParticipantRoleKey =
new CaseParticipantRoleKey();

    caseParticipantRoleKey.caseParticipantRoleID =
(Long) DynamicEvidenceTypeConverter.convert(
      originalDetails.getAttribute("participant"));

    Long concernRoleID1 = caseParticipantRoleObj.
readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID =
(Long) DynamicEvidenceTypeConverter.convert(
      originalDetails.getAttribute("relatedParticipant"));

    Long relConcernRoleID1 = caseParticipantRoleObj.
readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID =
(Long) DynamicEvidenceTypeConverter.convert(
      reciprocalDetails.getAttribute("participant"));

    Long concernRoleID2 = caseParticipantRoleObj.
readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    caseParticipantRoleKey.caseParticipantRoleID =
(Long) DynamicEvidenceTypeConverter.convert(
      reciprocalDetails.getAttribute("relatedParticipant"));

    Long relConcernRoleID2 = caseParticipantRoleObj.
readCaseIDandParticipantID(caseParticipantRoleKey).participantRoleID;

    String workingRelationshipOriginal =
      originalDetails.getAttribute("workingRelationship").getValue();

    String workingRelationshipRec = "";
    if (workingRelationshipOriginal.equals("ISMANAGEROF")) {
      workingRelationshipRec = "ISMANAGEDBY";
    }

    return reciprocalDetails.getAttribute("workingRelationship")
```

```
.getValue().equals(
      workingRelationshipRec)
      && (concernRoleID1.longValue() ==
relConcernRoleID2.longValue())
      && (relConcernRoleID1.longValue() ==
concernRoleID2.longValue());
  }

  public long getPrimaryParticipant(final Object originalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
(DynamicEvidenceDataDetails) originalEvidence;

    long caseParticipantRoleID = Long.parseLong(
      originalDetails.getAttribute("participant").getValue());

    CaseParticipantRoleKey caseParticipantRoleKey =
new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID =
caseParticipantRoleID;

    return CaseParticipantRoleFactory.newInstance()
.read(caseParticipantRoleKey).participantRoleID;
  }

  public long getRelatedParticipant
(final Object originalEvidence)
    throws AppException, InformationalException {

    DynamicEvidenceDataDetails originalDetails =
      (DynamicEvidenceDataDetails) originalEvidence;

    long caseParticipantRoleID = Long.parseLong(
      originalDetails.getAttribute("relatedParticipant").getValue());

    CaseParticipantRoleKey caseParticipantRoleKey =
 new CaseParticipantRoleKey();
    caseParticipantRoleKey.caseParticipantRoleID =
 caseParticipantRoleID;

    return CaseParticipantRoleFactory.newInstance().
read(caseParticipantRoleKey).participantRoleID;
  }
}
```

**Step 2: Add a Binding to the New Reciprocal Evidence Conversion Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    MapBinder<CASEEVIDENCEEntry,
ReciprocalEvidenceConversion> recEvidenceConversionMapBinder =
      MapBinder.newMapBinder(binder(),
CASEEVIDENCEEntry.class, ReciprocalEvidenceConversion.class);

    reciprocalEvidenceConversionMapBinder.addBinding(
   CASEEVIDENCEEntry.get("WORKINGRELATIONSHIP")).to(
      SampleWorkingRelationshipReciprocalConversion.class);
  }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

### Reciprocal Evidence Limitations

The reciprocal evidence handling infrastructure has the following limitations:

- The evidence must be temporal evidence. It can be static, dynamic or generated evidence.
- The evidence must have a participant and related participant, alternatively, the `ReciprocalEvidenceConversion` implementation code must be able to determine the participant and related participant using the evidence details.
- When reciprocal evidence and its related original evidence are both on the same case then their changes must be always applied together, as otherwise original and reciprocal evidence data will be out of sync.
- Reciprocal evidence can be processed automatically only if both related participants are registered as MEMBER or PRIMARY participants on the same case, or evidence is recorded as person/prospect person evidence.

## Participant Data Case Owner

### Why Change the Participant Data Case Owner?

It may be necessary to change the participant data case owner if tighter control is required around the ownership of participants.

### Changing the Participant Data Case Owner

When a person/prospect person is registered on the system, a case is created in the background to help manage this data, this is also known as a 'Participant Data Case'. By default, this case has a case owner, the logged in user. It is possible to change this to a different case owner by way of the `PDCCaseOwnerAssignmentStrategy` Interface. The `PDCCaseOwnerAssignmentStrategy` Interface can be found in the curam.pdc.impl package and has one method `createOwner`. It accepts two parameters:

- key - the identifier of the Participant Data Case
- ownerDtls - the details of the Participant Data Case owner

### Changing the Participant Data Case Owner Example

The following example outlines how to change the Participant Data Case Owner, in this scenario the owner is set to the system user.

The steps involved are:

- Provide a case owner assignment strategy implementation that will set the case owner
- Add a binding to the case owner assignment strategy implementation

**Step 1: Provide a Case Owner Assignment Strategy Implementation:** The code snippet below demonstrates a sample implementation for `PDCCaseOwnerAssignmentStrategy`, it simply sets the owner to be the system user.

```
@Singleton
public class SampleCaseOwnerAssignmentStrategyImpl
  implements PDCCaseOwnerAssignmentStrategy {

  public void createOwner(CaseHeaderKey key,
OrgObjectLinkDtls ownerDtls)
    throws AppException, InformationalException {
```

```
    ownerDtls.orgObjectType = ORGOBJECTTYPE.USER;
    ownerDtls.userName = UserAccessFactory.newInstance().
getSystemUserDetails().userName;

    OrgObjectLinkFactory.newInstance().insert(ownerDtls);

    OrgObjectLinkKey orgObjectLinkKey = new OrgObjectLinkKey();
    orgObjectLinkKey.orgObjectLinkID = ownerDtls.orgObjectLinkID;

    CaseUserRoleDtls caseUserRoleDtls =
 new CaseUserRoleDtls();
    caseUserRoleDtls.caseID = key.caseID;
    caseUserRoleDtls.orgObjectLinkID =
 orgObjectLinkKey.orgObjectLinkID;
    caseUserRoleDtls.typeCode = CASEUSERROLETYPE.OWNER;
    caseUserRoleDtls.recordStatus = RECORDSTATUS.NORMAL;

    curam.core.sl.entity.fact.CaseUserRoleFactory.newInstance()
.insert(caseUserRoleDtls);

    CaseHeader caseHeaderObj = CaseHeaderFactory.newInstance();
    CaseHeaderDtls caseHeaderDtls = caseHeaderObj.read(key);
    caseHeaderDtls.ownerOrgObjectLinkID =
 orgObjectLinkKey.orgObjectLinkID;
    caseHeaderObj.modify(key, caseHeaderDtls);
  }
}
```

**Step 2: Add a Binding to the Case Owner Assignment Strategy Implementation:** Guice bindings are used to register the implementation.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Register the implementation
    bind(PDCCaseOwnerAssignmentStrategy.class)
   .to(SampleCaseOwnerAssignmentStrategyImpl.class);
  }
}
```

**Note:** New Guice modules must be registered by adding a row to the ModuleClassName database table. Please see the Persistence Cookbook for more information.

# Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

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

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

IBM may use or distribute any of the information you supply 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 Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All 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 the names and addresses used by an actual business enterprise 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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Privacy Policy considerations

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

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

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

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and

IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

## Trademarks

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/us/en/copytrade.shtml.

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

Printed in USA