IBM Cúram Social Program Management

Handbuch 'Cúram Person and Prospect Person Evidence Developers Guide'

Version 6.0.5

IBM Cúram Social Program Management

Handbuch 'Cúram Person and Prospect Person Evidence Developers Guide'

Version 6.0.5

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen in „Bemerkungen" auf Seite 39 gelesen werden.

Ausgabe: Mai 2013

Diese Ausgabe bezieht sich auf IBM Cúram Social Program Management v6.0 5 und alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellen

Kapitel 1. Einführung

1.1 Zweck

Dieses Handbuch vermittelt ein allgemeines technisches Verständnis von Angaben zu Personen/Anwärtern und ihren Komponenten. Das Handbuch zeigt außerdem die verfügbaren Anpassungsoptionen und Erweiterungspunkte auf und enthält Anweisungen zur Implementierung dieser Anpassungen.

1.2 Zielgruppe

Zielgruppe dieses Handbuchs sind Entwickler und Architekten, die versuchen, eine Lösung für Angaben zu Personen oder Anwärtern zu implementieren.

Wichtig: Dieses Handbuch ist nur für diejenigen Leser maßgeblich, die die Anwendung für Beteiligte mit dynamischen Angaben zu Personen oder Anwärtern benutzen.

1.3 Voraussetzungen

In diesem Dokument wird vorausgesetzt, dass der Leser mit Folgendem vertraut ist:

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

1.4 Kapitel in diesem Handbuch

In der folgenden Liste werden die Kapitel dieses Handbuchs beschrieben:

Übersicht über Angaben zu Personen/Anwärtern
Dieses Kapitel liefert eine allgemeine Übersicht über die wichtigsten technischen Aspekte bei Angaben zu Personen/Anwärtern.

Konzipieren von Lösungen für Personen-/Anwärterangaben
In diesem Kapitel werden einige Entwurfsaspekte erläutert, die beim Konzipieren von Lösungen für Personen-/Anwärterangaben berücksichtigt werden sollten.

Datenzuordnungen
Dieses Kapitel enthält die Zuordnung von Daten aus den mit der Anwendung bereitgestellten dynamischen Angabentypen zu älteren Datentabellen.

Anpassen von Personen-/Anwärterangaben
In diesem Kapitel werden die Anpassungsoptionen und Erweiterungspunkte beschrieben, die für Angaben zu Personen/Anwärtern zur Verfügung stehen.

Kapitel 2. Übersicht über Angaben zu Personen/Anwärtern

2.1 Daten für Personen/Anwärter als Angaben

Ein Teil der zu Personen und Anwärtern gespeicherten Informationen werden als Angaben geführt, die von den Fällen im System gemeinsam verwendet werden können. Das Speichern von Angaben zu Personen und Anwärtern und der Fluss dieser Angaben im System wird durch das Ineinandergreifen einer Reihe von Cúram-Komponenten und -Technologien ermöglicht:

- Cúram Dynamic Evidence wird zum Speichern der erfassten Personen-/Anwärterdaten und Ausführen von Basisvalidierungen verwendet.
- Mit Cúram Express Rules werden komplexe Validierungen für die erfassten Daten ausgeführt.
- Cúram Evidence Broker kann optional für die Bereitstellung der Daten zwischen den einzelnen Fällen verwendet werden.
- Mit Cúram Verification können wahlweise Verifizierungen für die erfassten Daten ausgeführt werden, wenn diese vom Broker aus anderen Fällen bereitgestellt gestellt wurden.

Abhängig von den Geschäftsanforderungen ist gegebenenfalls ein gewisses Maß an Konfigurationsmaßnahmen, ein gewisser Grad der Anpassung oder eine Kombination aus beidem erforderlich. In diesem Kapitel wird auf allgemeiner Ebene beschrieben, wie das System Personen-/Anwärterdaten verwaltet, und weist auf diejenigen Stellen hin, an denen gegebenenfalls eine Konfiguration und/oder Anpassung erforderlich ist.

Anmerkung: Besondere Beachtung sollte dem erforderlichen Geschäftsverhalten des Systems während der Entwurfsphase einer Cúram-Implementierung geschenkt werden. Als Ausgangslektüre sollten das Handbuch 'Cúram Participant Guide' und das Handbuch 'Cúram Evidence Guide' dienen, um ein Verständnis dafür zu entwickeln, wie ein System konfiguriert werden sollte, um die entsprechenden Geschäftsanforderungen zu unterstützen.

2.2 Wie werden Angaben zu Personen/Anwärtern verwaltet?

Die Verwaltung (Pflege) von Personen- und Anwärterdaten als Angaben wird von den folgenden wesentlichen Grundlagen gestützt:

- Jede Person und jedem Anwärter verfügt über einen zugeordneten Personen- oder Anwärterfall (Beteiligtendatenfall), der im Anschluss an die Registrierung verdeckt erstellt wird.
- Personen- und Anwärterdaten werden als Angaben in Tabellen für dynamische Angaben gespeichert und von den dynamischen Angabentypen beschrieben, die der Person und/oder dem Anwärter zugeordnet sind.
- Die als Angaben erfassten Daten werden zurück in die älteren Datenbanktabellen repliziert. Daher sollten die älteren Datenbanktabellen synchron mit den dynamischen Angaben sein.
- Beim Bearbeiten eines Personen- oder Anwärterdatensatzes werden die Daten aus den Tabellen für dynamische Angaben abgerufen, in diese Tabellen geschrieben (und per Replikation wieder zurück in die älteren Datenbanktabellen übernommen).
- In einigen Fällen lesen Anwendungsanzeigen und Vorgänge immer noch Daten aus den älteren Datenbanktabellen aus.
- Personen- und Anwärterfalltypen können so konfiguriert werden, dass mit dem Cúram Evidence Broker Beteiligtendaten aus und in anderen Fällen bereitgestellt werden.

2.2.1 Typen von Angaben zu Personen/Anwärtern

Es wurde eine Reihe dynamischer Angabentypen bereitgestellt und zu den Beteiligten vom Typ 'Person' und 'Anwärter' zu zugeordnet. Diese Angabentypen und ihre Attribute bzw. ihre Zuordnung zu der Person/dem Anwärter dürfen nicht entfernt werden.

Dort, wo eine Notwendigkeit besteht, zusätzliche Daten als Angaben zu Personen und Anwärtern zu pflegen, können wie im Handbuch 'Cúram Dynamic Evidence Configuration Guide' beschrieben neue Angabentypen erstellt und in der Administrationskomponente zu der Person/dem Anwärter zugeordnet werden. Gleichermaßen können neue Attribute zu den bereits vorhandenen Typen dynamischer Angaben hinzugefügt werden. Dort, wo die Daten, die zu neuen oder bereits vorhandenen Angabentypen hinzugefügt werden, bereits in einer älteren Datenbanktabelle vorhanden ist, müssen weitere Schritte zur Anpassung ausgeführt werden:

- Der Code, der die Daten aus den Tabellen für dynamische Angaben zur Replikation in den älteren Datenbanktabellen abruft ('Replikator'), muss so erweitert werden, dass er die Daten, die als Angaben gespeichert werden, repliziert.
- Dort, wo bereits Daten in den älteren Datenbanktabellen vorhanden sind, müssen diese Daten in die entsprechenden Tabellen für dynamische Angaben kopiert werden. Der Code, der für die Ausführung dieser Operation zuständig ist ('Konverter') muss so erweitert werden, dass die zusätzlichen Daten in Angaben konvertiert werden.

Weitere Details und Beispielimplementierungen für beide Anpassungen durch Erweiterung enthält das Kapitel 'Anpassen von Personen-/Anwärterangaben'.

2.2.2 Angabenvalidierungen

Die Angaben zu Personen und Anwärtern werden bei ihrer Erstellung und Bearbeitung validiert. Die Implementierung dieser Validierungen erfolgt auf eine der beiden folgenden Weisen:

- Durch Verwendung der Validierungsfunktionen des Editors für dynamische Angaben. Beispiel: Validierungen von Pflichteingabefeldern.
- Durch Verwendung von Cúram Express Rule-Regelwerken. Beispiel: Angabenübergreifende Validierungen.

Dort, wo neue dynamische Angabentypen oder Attribute hinzugefügt werden, sollten Kunden anhand eines dieser Mechanismen alle erforderlichen Validierungen hinzufügen. Dies wird ausführlicher im Handbuch 'Cúram Dynamic Evidence Configuration Guide' beschrieben.

Wenn Angaben für eine Person oder einen Anwärter über den Broker bereitgestellt werden, so werden diese Validierungen nicht überprüft. Die vom Broker bereitgestellten Angaben werden stets akzeptiert (angenommen), um zu vermeiden, dass sie verloren gehen, denn für Personen bzw. Anwärter existiert das Konzept der 'eingehenden Angaben' nicht. Werden Angaben zu Personen/Anwärtern eingegeben, so werden diese unverzüglich validiert. Angaben, die über den Broker aus einem anderen Fall bereitgestellt werden, werden jedoch automatisch akzeptiert und aktiviert, selbst wenn die Validierungsprüfungen fehlschlagen. Wenn bei anderen Falltypen Angaben zu Personen/Anwärtern über den Broker in dem Fall bereitgestellt werden, so verhindert eine fehlgeschlagene Validierungsprüfung hingegen die Aktivierung der Angaben.

2.2.3 Gemeinsames Verwenden von Angaben

Über das Framework für Angaben wird die Funktionalität zur gemeinsamen Verwendung von Angaben durch eine Person/einen Anwärter, Anwendungsfälle und laufende Fälle bereitgestellt. Der Evidence Broker ermöglicht und vermittelt diese gemeinsame Verwendung von Angaben. Die gemeinsame Verwendung von Angaben ist unidirektional und erfolgt pro Angabentyp. Dies bedeutet, dass unterschiedliche Ziele einen Angabentyp auf unterschiedliche Weise erhalten und gemeinsam verwenden können. Bei Be-

darf kann ein Falltyp zwar gemeinsam verwendete Angaben erhalten, ohne dass seine eigenen Angaben gemeinsam verwendet werden können. Die folgenden drei wichtigen Geschäftsfunktionen veranlassen den Evidence Broker zum Übertragen von Angaben:

- Wenn eine neue Person zu einem Zielfall hinzugefügt wird. Wenn zum Beispiel Personen-/Anwärterangaben wie etwa Identifikationsangaben zur gemeinsamen Verwendung mit einem integrierten Fall konfiguriert wurden, so überprüft der Evidence Broker zuerst, ob für diese Person bereits Personen-/Anwärterangaben vorhanden sind. Werden Angaben gefunden, so sucht der Evidence Broker dann nach aktiven Identifikationsangaben und übernimmt diese zur gemeinsamen Verwendung in den integrierten Fall.
- Wenn Änderungen an Angaben in einem Quellenfall vorgenommen werden. Wenn zum Beispiel Änderungen an den Identifikationsangaben für eine Person vorgenommen werden, sorgt der Evidence Broker dafür, dass diese Änderungen zur gemeinsamen Verwendung in den integrierten Fall übernommen werden.
- Wenn ein neuer Zielfall erstellt wird. Bei jeder Erstellung eines neuen integrierten Falls sucht der Evidence Broker nach Identifikationsangaben für Personen/Anwärter, die gemeinsam verwendet werden. Werden solche Angaben gefunden, übernimmt der Evidence Broker diese Identifikationsangaben zur gemeinsamen Verwendung in den integrierten Fall.

Ausführlichere Informationen zum Evidence Broker enthält das Handbuch 'Cúram Evidence Broker Guide'.

Kapitel 3. Konzipieren von Lösungen für Personen-/Anwärterangaben

Beim Konzipieren einer Lösung für Angaben-/Anwärterangaben sollten die eigentlichen Daten, ihre Struktur, eventuelle Einschränkungen sowie der Fluss dieser Daten im System berücksichtigt werden.

3.1 Daten: Dynamische Angabentypen

3.1.1 Struktur

Angaben zu Personen/Anwärtern werden überwiegend als dynamische Angaben gespeichert. Sie werden von Datenstrukturen repräsentiert, die als dynamische Angabentypen bezeichnet werden. Dynamische Angabentypen definieren die Daten, ihren Typ und ihr Verhalten wie etwa Volatilität, berechnete Attribute usw. Nachdem dynamische Angabentypen definiert worden sind, müssen sie aktiviert und zu den relevanten Falltypen, Personen und Anwärtern zugeordnet werden. Weitere Informationen dazu, wie dynamische Angabentypen definiert werden, enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'.

Zu berücksichtigende Aspekte

- Ändern sich die Angaben im Laufe der Zeit?
- Ist der Angabentyp reziprok? Falls dies zutrifft, so sollte der Angabentyp über Attribute für Beteiligte und zugehörige Beteiligte verfügen.
- Für welche Falltypen sollen die Angaben verfügbar sein?
- Wenn der Angabentyp allgemein verwendet werden soll, überlegen Sie, ob sie ihn nicht als 'Bevorzugt' definieren. Hierdurch würden Fallbearbeiter die Möglichkeit erhalten, zügig Angaben für häufig erfasste Angabentypen zu erstellen.

3.1.2 Einschränkungen

3.1.2.1 Validierungen

Der Editor für dynamische Angaben enthält eine Reihe von Standardvalidierungen, die häufig bei der Verarbeitung dynamischer Angaben verwendet werden. Validierungen mit höherer Komplexität wie etwa angabenübergreifende Validierungen können mittels Cúram Express Rules eingebunden werden. Weitere Informationen zu Validierungen enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'.

Zu berücksichtigende Aspekte

- Überlegen Sie, welche Validierungen zur Sicherstellung der Datenintegrität erforderlich sind.
- Werden Angaben zu Personen/Anwärtern eingegeben, so werden diese unverzüglich validiert. Angaben, die über den Broker aus einem anderen Fall bereitgestellt werden, werden jedoch automatisch akzeptiert und aktiviert, selbst wenn die Validierungsprüfungen fehlschlagen. Wenn bei anderen Falltypen Angaben zu Personen/Anwärtern über den Broker in dem Fall bereitgestellt werden, so verhindert eine fehlgeschlagene Validierungsprüfung die Aktivierung der Angaben.
- Schließen Sie alle Validierungen ein, die für die Durchsetzung von Einschränkungen bei der Reihenfolge erforderlich sind.
- Versuchen Sie, nach Möglichkeit stets Standardvalidierungsmuster zu verwenden. Validierungsregelwerke sollten in solchen Fällen entwickelt werden, wenn ihre Implementierung anhand von Standardvalidierungen nicht möglich ist.
- Berücksichtigen Sie bei der Entwicklung von Prüfregelsätzen mit komplexeren Validierungen , wie der Datenabruf erfolgt. Bei korrekter Ausführung kann sich dies maßgeblich auf die Leistung auswirken.

Wichtig: Systemprozesse stützen sich mit den zusammen mit der Anwendung ausgelieferten Validierungen und es ist nicht regelkonform, diese Validierungen zu entfernen oder Änderungen an ihnen vorzunehmen.

3.1.2.2 Verifizierungen

Dieser Abschnitt betrifft nur diejenigen Benutzer, die über eine Lizenz für die Benutzung der Verifizierungskomponente verfügen.

Als Verifizierung bezeichnet man den Vorgang der Überprüfung von Angaben auf Richtigkeit. Angaben können auf verschiedene Arten verifiziert werden, beispielsweise durch Dokumente wie Geburtsurkunden oder Kontoauszüge, oder in mündlicher Form, beispielsweise durch Telefongespräche. Bei der Erfassung von Angaben wird die Verifizierungsengine aufgerufen, um zu ermitteln, ob für die Angaben eine Verifizierung ausgeführt werden muss.

Anmerkung: Mit Ausnahme von Angaben, die über den Broker in einem Datensatz für eine Person bzw. einen Anwärter bereitgestellt werden, können Angaben erst dann aktiviert werden, wenn alle obligatorischen Verifizierungsvoraussetzungen erfüllt sind.

Weitere Informationen zu Verifizierungen und ihrer Konfiguration enthält das Handbuch 'Cúram Verification Guide'.

Zu berücksichtigende Aspekte

- Ist für diese Angaben eine Verifizierung erforderlich?
- Welche Regeln gelten für die Verifizierung?
- Welche Informationen muss der Kunde bereitstellten?

3.2 Ablauf: Evidence Broker

Der Evidence Broker ist der Mechanismus, mit dem die gemeinsame Nutzung von Angaben im gesamten System ermöglicht wird. Wenn der Evidence Broker Angaben zu einem Personen- bzw. Anwärterdatensatz überträgt, werden diese Angaben automatisch akzeptiert und auf dem Personen- bzw. Anwärterdatensatz aktiviert, sodass der Benutzer Angaben nicht manuell akzeptieren und aktivieren muss. Weitere Informationen zum Evidence Broker und den Optionen für seine Konfiguration enthält das Handbuch 'Cúram Evidence Broker Guide'. Empfohlene Ansätze für die Bereitstellung von Angaben über den Broker werden im Handbuch 'Cúram Evidence Guide' beschrieben.

Zu berücksichtigende Aspekte

- Wird ein und derselbe Angabentyp in mehreren Falltypen verwendet? Wenn dies zutrifft, sollen Änderungen an diesen Angaben dann auch an andere Fälle übertragen werden?
- Soll der Zielfall so eingerichtet sein, dass Änderungen automatisch akzeptiert werden, oder ist das manuelle Eingreifen seitens des Fallbearbeiters erwünscht, bei dem dieser entscheidet, ob er die eingehenden Angaben annimmt?
- Damit die systemseitige Verarbeitung ordnungsgemäß funktioniert, ist es unbedingt erforderlich, dass Daten für Personen/Anwärter, die außerhalb des Beteiligtenmanagers erfasst wurden, so konfiguriert sind, dass sie an den Beteiligtenmanager rückübertragen und von diesem gemeinsam verwendet werden.

3.3 Cúram Express Rules: Berechnungen der Anspruchsberechtigung/ Leistungshöhe bei Fällen

Für Bereiche, in den Beteiligtendaten mit Cúram Express Rules aus älteren Datenbanktabellen gelesen werden, um Berechnungen für die Anspruchsberechtigung und Leistungshöhe für einen Fall auszuführen, sollte anhand einer vorherigen Analyse entschieden werden, woher diese Daten bezogen werden sollen. Hierfür gibt es die folgenden drei Optionen mit jeweils eigenen Vorzügen und Einschränkungen:

- Beteiligtendaten aus den vom Beteiligtenmanager gespeicherten dynamischen Angaben lesen
- Beteiligtendaten lesen, die über den Broker in Fällen bereitgestellt wurden
- Weiterhin aus den älteren Tabellen lesen

3.3.1 Beteiligtendaten aus den vom Beteiligtenmanager gespeicherten dynamischen Angaben lesen

Zu berücksichtigende Aspekte

- Es wird mit der primären Datenquelle gearbeitet
- Änderungen an Angaben bewirken sofortige Neuberechnungen
- Fallbearbeiter haben nicht die Möglichkeit einer Überprüfung

3.3.2 Lesen von Beteiligtendaten, die über den Broker in Fällen bereitgestellt wurden

Zu berücksichtigende Aspekte

- Diese Option wird für alle neuen Entwicklungen empfohlen
- Änderungen treten erst nach der Aktivierung der Angaben in Kraft
- Der Angabentyp muss so konfiguriert sein, dass die Angaben über den Broker in dem Fall bereitgestellt werden

3.3.3 Weiterhin aus den älteren Tabellen lesen

Zu berücksichtigende Aspekte

- Diese Option sollte sorgfältig eingeschätzt werden und wird nur für Kunden empfohlen, die ein Upgrade ausführen

Kapitel 4. Zuordnungen von Daten aus den dynamischen Angabentypen

In den nachfolgenden Tabellen sind die Zuordnungen von Daten aus den dynamischen Angabentypen zu den älteren Datenbanktabellen dargestellt.

Anmerkung: Diese Zuordnungen werden von Replikatoren vorgenommen. Die regressive (umgekehrte) Zuordnung erfolgt anhand von Konvertern.

4.1 Adresse

Tabelle 1. Zuordnung von Adressen

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRoleAddress.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
address	Address.addressData
fromDate	ConcernRoleAddress.startDate
toDate	ConcernRoleAddress.endDate
addressType	ConcernRoleAddress.typeCode
comments	ConcernRoleAddress.comments

4.2 Bankkonto

Tabelle 2. Zuordnung von Bankkonten

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRoleBankAccount.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
accountName	BankAccount.name
accountNumber	BankAccount.accountNumber
accountType	BankAccount.typeCode
sortCode	BankAccount.bankSortCode
fromDate	BankAccount.startDate
toDate	BankAccount.endDate
accountStatus	BankAccount.bankAccountStatus
jointAccountInd	BankAccount.jointAccountInd
comments	BankAccount.comments

4.3 Geburt und Tod

Tabelle 3. Zuordnung von Geburt und Tod

Attribut für dynamische Angaben	Datenbankspalte
person	Person/ProspectPerson.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)

Tabelle 3. Zuordnung von Geburt und Tod (Forts.)

Attribut für dynamische Angaben	Datenbankspalte
birthLastName	Person/ProspectPerson.personBirthName
mothersBirthLastName	Person/ProspectPerson.motherBirthSurname
dateOfBirth	Person/ProspectPerson.dateOfBirth
dateOfDeath	Person/ProspectPerson.dateOfDeath
comments	N/V

4.4 Bevorzugte Kontaktdaten

Tabelle 4. Zuordnung von bevorzugten Kontaktdaten

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRole.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
preferredLanguage	ConcernRole.preferredLanguage
preferredCommunication	ConcernRole.prefCommMethod
comments	N/V

4.5 E-Mail-Adresse

Tabelle 5. Zuordnung von E-Mail-Adressen

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRoleEmailAddress.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
emailAddress	EmailAddress.emailAddress
fromDate	ConcernRoleEmailAddress.startDate
toDate	ConcernRoleEmailAddress.endDate
emailAddressType	ConcernRoleEmailAddress.typeCode
comments	EmailAddress.comments

4.6 Geschlecht

Tabelle 6. Zuordnung des Geschlechts

Attribut für dynamische Angaben	Datenbankspalte
person	Person/ProspectPerson.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
gender	Person/ProspectPerson.gender
comments	N/V

4.7 Identifikation

Tabelle 7. Zuordnung der Identifikation

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRoleAlternateID.concernRoleID (berechnet anhand von 'caseParticipantRoleID')
alternateID	ConcernRoleAlternateID.alternateID
altIDType	ConcernRoleAlternateID.typeCode
fromDate	ConcernRoleAlternateID.startDate
toDate	ConcernRoleAlternateID.endDate
comments	ConcernRoleAlternateID.comments

4.8 Name

Tabelle 8. Zuordnung von Namen

Attribut für dynamische Angaben	Datenbankspalte
participant	AlternateName.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
title	AlternateName.title
firstName	AlternateName.firstForename
middleName	AlternateName.otherForename
lastName	AlternateName.surname
suffix	AlternateName.nameSuffix
initials	AlternateName.initials
nameType	AlternateName.nameType
comments	AlternateName.comments

4.9 Telefonnummer

Tabelle 9. Zuordnung von Telefonnummern

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRolePhoneNumber.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
phoneCountryCode	PhoneNumber.phoneCountryCode
phoneAreaCode	PhoneNumber.phoneAreaCode
phoneNumber	PhoneNumber.phoneNumber
phoneExtension	PhoneNumber.phoneExtension
fromDate	ConcernRolePhoneNumber.startDate
toDate	ConcernRolePhoneNumber.endDate
phoneType	ConcernRolePhoneNumber.typeCode
comments	PhoneNumber.comments

4.10 Beziehung

Tabelle 10. Zuordnung von Beziehungen

Attribut für dynamische Angaben	Datenbankspalte
participant	ConcernRoleRelationship.concernRoleID (anhand von 'caseParticipantRoleID' berechnet)
relatedParticipant	ConcernRoleRelationship.relConcernRoleID
fromDate	ConcernRoleRelationship.startDate
toDate	ConcernRoleRelationship.endDate
relationshipType	ConcernRoleRelationship.relationshipType
endReason	ConcernRoleRelationship.relEndReasonCode
comments	ConcernRoleRelationship.comments

Kapitel 5. Anpassen von Personen-/Anwärterangaben

5.1 Einführung

In diesem Kapitel werden die Anpassungsoptionen und Erweiterungspunkte beschrieben, die für Angaben zu Personen/Anwärtern zur Verfügung stehen. Abhängig von Ihren vorhandenen Anpassungen und Konfigurationen dürften einige oder sogar alle für Sie zutreffen. Hierbei müssen die fünf nachfolgend aufgelisteten Hauptbereiche berücksichtigt werden:

- Replikatoren
- Konverter
- Auswahl der Primärinformationen
- Reziproke Angaben
- Eigentümer des Beteiligtendatenfalls

Jeder dieser Bereiche wird ausführlich beschrieben und mit Beispielen ergänzt. **Beachten Sie, dass es sich lediglich um Beispiele handelt.**

5.2 Replikatoren

5.2.1 Was ist ein Replikator?

Ein Replikator sorgt dafür, dass sich Änderungen von Angaben zum Zweck der Abwärtskompatibilität auch in den relevanten älteren Tabellen widerspiegeln. Der Replikator konvertiert dazu die Details der dynamischen Angaben in ein Struct, das die Details enthält, die in den älteren Tabellen gespeichert werden sollen. Diese Details werden dann in die relevanten Datenbanktabellen geschrieben, wodurch sichergestellt wird, dass die Informationen in den älteren Tabellen synchron mit denen in der primären Datenquelle (nämlich den dynamischen Angaben) ist. Für jeden Angabentyp für Personen/Anwärter werden Standardreplikatorimplementierungen zur Verfügung gestellt. Diese Standardimplementierungen enthalten Erweiterungspunkte, die eine Replikation zu angepassten Feldern ermöglichen. Dieser Aspekt wird im nachfolgenden Abschnitt behandelt.

Anmerkung: In einem Folgesatz, d. h. einer Gruppe sukzessiv gültigen Änderungen, wird zum Replizieren von Daten in die älteren Tabellen nur die jeweils letzte Version verwendet.

5.2.2 Warum sollte ein Replikator erweitert werden?

In Fällen, in denen ältere Datenbanktabellen erweitert wurden, kann die Erweiterung eines Replikators notwendig sein.

5.2.3 Replikatorerweiterung

Es ist möglich, die mit der Anwendung bereitgestellten Replikatoren so zu erweitern, dass Angaben zu Personen/Anwärtern in angepassten Datenbankspalten repliziert werden können. Für jeden bereitgestellten Angabentyp stehen die unten aufgelisteten Schnittstellen zur Verfügung, die sich im Paket 'curam.pdc.impl' befinden. Es können angepasste Implementierungen geschrieben werden, die je nach Angabentyp Gebrauch von diesen Schnittstellen machen.

Replikatorerweiterungsschnittstellen

- PDCAddressReplicatorExtender
- PDCAlternateIDReplicatorExtender
- PDCAlternateNameReplicatorExtender
- PDCBankAccountReplicatorExtender

- PDCBirthAndDeathReplicatorExtender
- PDCContactPreferencesReplicatorExtender
- PDCEmailAddressReplicatorExtender
- PDCGenderReplicatorExtender
- PDCPhoneNumberReplicatorExtender
- PDCRelationshipsReplicatorExtender

Der Großteil der Schnittstellen verfügt über eine einzige Methode namens `assignDynamicEvidenceToExtendedDetails`. Für sie können die folgenden zwei Parameter angegeben werden:

- dynamicEvidenceDataDetails - Die Details der dynamischen Angaben
- details - Das Struct, das die erweiterten Details für die ältere Tabelle enthält

`PDCBirthAndDeathReplicatorExtender` und `PDCGenderReplicatorExtender` verfügen über die beiden Methoden `assignDynamicEvidenceToExtendedPersonDetails` und `assignDynamicEvidenceToExtendedProspectPersonDetails`. Für `assignDynamicEvidenceToExtendedPersonDetails` können die folgenden zwei Parameter angegeben werden:

- dynamicEvidenceDataDetails - Die Details der dynamischen Angaben
- details - Das Struct, das die erweiterten Personendetails für die ältere Tabelle enthält

Für `assignDynamicEvidenceToExtendedProspectPersonDetails` können ebenfalls zwei Parameter angegeben werden:

- dynamicEvidenceDataDetails - Die Details der dynamischen Angaben
- details - Das Struct, das die erweiterten Anwärterdetails für die ältere Tabelle enthält

5.2.4 Beispiel: Implementieren einer Replikatorerweiterung für Personen-/Anwärterangaben

Das folgende Beispiel stellt dar, wie ein Replikator so erweitert wird, dass er Personen-/Anwärterangaben zu angepassten Feldern zuordnet. Das vorliegende Beispiel liefert eine sehr rudimentäre Implementierung einer Erweiterung von `PDCPhoneNumberReplicatorExtender`. In diesem Szenario wurde die Tabelle namens 'PhoneNumber' erweitert und enthält ein angepasstes Feld 'phoneProvider'. Die dynamische Angabenkonfiguration für die Telefonnummer enthält auch ein Attribut 'phoneProvider'. Bei diesem Beispiel wird vorausgesetzt, dass das Struct `ParticipantPhoneDetails` bereits um das angepasste Feld erweitert wurde. Weitere Informationen zur Konfiguration von dynamischen Angaben enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'. Die angepasste Replikatorerweiterungsimplementierung ist dafür zuständig, die dynamischen Angabendaten zu dem Struct-Attribut zuzuordnen, das das Attribut 'phoneProvider' in der erweiterten Tabelle 'PhoneNumber' darstellt.

Anmerkung: Es ist lediglich die Zuordnung von Daten erforderlich; die eigentliche Replikation von Daten wird von der Standardimplementierung ausgeführt.

Zum Erweitern eines Replikators müssen die folgenden Schritte ausgeführt werden:

- Bereitstellen einer Replikatorerweiterungsschnittstelle, mit der die angepassten Daten zurück zur älteren Tabelle zugeordnet werden
- Hinzufügen einer Bindung zu der neuen Replikatorerweiterungsimplementierung

5.2.4.1 Schritt 1: Bereitstellen einer Replikatorerweiterungsimplementierung

Der erste Schritt besteht aus der Bereitstellung einer neuen Implementierung, die die relevante Replikationserweiterungsschnittstelle für den Angabentyp implementiert und das angepasste Feld wieder zurück zu der älteren Tabelle zuordnet. Das nachfolgende Code-Snippet veranschaulicht eine angepasste Implementierung für `PDCPhoneNumberReplicatorExtender`. Es weist den Wert des Attributs für dynamische An-

gaben einfach dem Struct-Attribut 'phoneProvider' zu. Diese Informationen werden dann gemeinsam mit den übrigen Attributen für dynamische Angaben über die Standardkonverterimplementierung für PDCPhoneNumberReplicatorExtender gespeichert.

```
public class SampleReplicatorExtenderImpl
  implements PDCPhoneNumberReplicatorExtender {

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

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

5.2.4.2 Schritt 2: Hinzufügen einer Bindung zu der neuen Replikatorerweiterungsimplementierung

Die Implementierung wird mit Guice-Bindungen registriert.

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Registrieren der Replikatorerweiterungsimplementierung
  Multibinder<PDCPhoneNumberReplicatorExtender> sampleReplicatorExtender =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberReplicatorExtender.class);

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

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'ModuleClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

5.2.5 Warum sollte ein Replikator implementiert werden?

In Fällen, in denen neue dynamische Angabentypen eingeführt wurden, kann die Implementierung eines neuen Replikators notwendig sein.

5.2.6 Implementieren eines Replikators

Replikatoren können einfach entwickelt werden, um Szenarien wie neuen dynamischen Angabentypen Rechnung zu tragen. Im nächsten Abschnitt wird anhand eines detaillierten Beispiels dargestellt, welche Schritte und Artefakte erforderlich sind, um einen neuen Replikator in Betriebsbereitschaft zu versetzen. Replikatorimplementierungen werden über einen ereignisbasierten Mechanismus aufgerufen. Wenn dynamische Angaben im Anschluss an einen Einfüge-, Änderungs- oder Löschvorgang aktiviert werden, wird ein Ereignis ausgelöst. Für neue Angabentypen muss ein Ereignislistener entwickelt werden, der dieses Ereignis überwacht und den Replikationsprozess aufruft. Dieser Vorgang wird an späterer Stelle in diesem Kapitel ausführlicher erörtert. Im nächsten Abschnitt wird veranschaulicht, wie die Implementierung eines Replikators erfolgt.

5.2.7 Beispiel: Implementieren eines Replikators für Personen-/Anwärterangaben

Das folgende Beispiel stellt dar, wie die Implementierung eines Replikators erfolgt. In diesem Szenario wurde 'Beispiel für Wohnsitz im Ausland' als neuer dynamischer Angabentyp konfiguriert. Weitere Informationen dazu, wie ein neuer Angabentyp konfiguriert wird, enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'. Der Angabentyp 'Beispiel für Wohnsitz im Ausland' weist die folgenden Attribute auf:

- participant - Die ID der Fallbeteiligtenrolle der Person/des Anwärters, für die/den die Angaben eingegeben werden
- country - Das Land, in dem sich der Wohnsitz befindet
- fromDate - Das Datum, an dem der Aufenthalt an diesem Wohnsitz begann
- toDate - Das Datum, an dem der Aufenthalt an diesem Wohnsitz endete
- reason - Der Grund für den Wohnsitz in diesem Land

Es wurde vorausgesetzt, dass dieser dynamische Angabentyp aktiviert und zur Verwendung mit Personen/Anwärtern konfiguriert wurde. Bisher wurden die Informationen für das 'Beispiel für Wohnsitz im Ausland' als statische Angaben in der Datenbanktabelle 'SampleForeignResidency' gespeichert. Nun ist es erforderlich, diese Informationen als dynamische Angaben zu speichern. Gegebenenfalls ist ein neuer Replikator erforderlich, um die in einer älteren Datenbanktabelle vorgenommenen Änderungen an Angaben zu replizieren und auf diese Weise sicherzustellen, dass diese Tabelle mit den dynamischen Angaben synchron ist.

Zum Implementieren eines Replikators müssen die folgenden Schritte ausgeführt werden:
- Bereitstellen einer Replikatorschnittstelle für den dynamischen Angabentyp
- Bereitstellen einer Replikatorimplementierung, durch die dynamische Angaben in der älteren Datenbanktabelle repliziert werden
- Implementieren eines Ereignislisteners, der die Replikation auslöst
- Hinzufügen einer Bindung zu der neuen Ereignislistenerimplementierung

5.2.7.1 Schritt 1: Bereitstellen einer Replikatorimplementierung

Die neue Replikatorschnittstelle sollte die folgenden drei Methoden enthalten:

`replicateInsertEvidence` - Repliziert aktivierten, eingefügten Angaben des 'Beispiels für Wohnsitz im Ausland' in der älteren Datenbanktabelle 'Beispiel für Wohnsitz im Ausland'. Für sie kann der folgende Parameter angegeben werden:
- evidenceDescriptorDtls - Die aktivierten Angabendeskriptordetails

`replicateModifyEvidence` - Repliziert aktivierten, geänderten Angaben des 'Beispiels für Wohnsitz im Ausland' in der älteren Datenbanktabelle 'Beispiel für Wohnsitz im Ausland'. Für sie können die folgenden zwei Parameter angegeben werden:
- evidenceDescriptorDtls - Die aktivierten Angabendeskriptordetails
- previousActiveEvidDescriptorDtls - Die Angabendeskriptordetails für die Angaben, die vor dem Änderungsvorgang aktiv waren

`replicateRemoveEvidence` - Repliziert aktivierten, entfernten Angaben des 'Beispiels für Wohnsitz im Ausland' in der älteren Datenbanktabelle 'Beispiel für Wohnsitz im Ausland'. Für sie kann der folgende Parameter angegeben werden:
- evidenceDescriptorDtls - Die aktivierten Angabendeskriptordetails

```
@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;
}
```

5.2.7.2 Schritt 2: Bereitstellen einer Replikatorimplementierung

Die Replikatorimplementierung sollte Implementierungen für die drei im vorherigen Abschnitt beschriebenen Methoden bereitstellen. Diese Methoden sollten sicherstellen, dass die Daten von dynamischen Angaben in Daten konvertiert werden, die in die älteren Datenbanktabellen geschrieben werden können, und dass die älteren Tabellen für diesen Angabentyp aktualisiert werden.

```
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);
    }
  }
}
```

5.2.7.3 Schritt 3: Implementieren eines Ereignislisteners

Es muss ein neuer Ereignislistener implementiert werden, um das Auftreten von Ereignissen des Typs 'Beispiel für Wohnsitz im Ausland' zu überwachen. Diese Ereignisse treten als Folge einer Angabenaktivierung auf. Der Listener sollte die Schnittstelle 'curam.pdc.impl.PDCEvents' implementieren und Implementierungen für die drei Methoden bereitstellen. Dieser Punkt bietet sich zum Starten des Replikationsprozesses sowie aller sonstigen angepassten Verarbeitungsschritte an, die gegebenenfalls ausgeführt werden müssen.

```
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);
    }
  }
}
```

5.2.7.4 Schritt 4: Hinzufügen einer Bindung zu der neuen Ereignislistenerimplementierung

Die Implementierung wird mit Guice-Bindungen registriert.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Registrieren des Ereignislisteners
    Multibinder<PDCEvents> sampleEventListeners =
      Multibinder.newSetBinder(binder(), PDCEvents.class);

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

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'ModuleClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

5.3 Konverter

5.3.1 Was ist ein Konverter?

Ein Konverter ist ein Mechanismus, mit dem ältere Personen-/Anwärterdaten in dynamische Angaben konvertiert werden können. Wenn ältere Datenbanktabellen außerhalb der Anwendung mit Daten aufgefüllt wurden, beispielsweise anhand von Tools wie Cúram Data Manager (DMX-Dateien), können diese Daten mit Hilfe von Konvertern in dynamische Angaben konvertiert (umgewandelt) werden. Für jeden Angabentyp für Personen/Anwärter werden Standardkonverterimplementierungen zur Verfügung gestellt. Diese Standardimplementierungen enthalten Erweiterungspunkte, die eine Konvertierung angepasster Felder ermöglichen. Dieser Aspekt wird im nachfolgenden Abschnitt behandelt.

5.3.2 Warum sollte ein Konverter erweitert werden?

In Fällen, in denen ältere Datenbanktabellen erweitert wurden, kann die Erweiterung eines Konverters notwendig sein. Konverter werden im Allgemeinen nur in Entwicklungsumgebungen oder als Tools für Upgrades bereitgestellt und sollten nicht im Rahmen der ganz regulären täglichen Verarbeitung eingesetzt werden.

5.3.3 Konvertererweiterung

Die gemeinsam mit der Anwendung bereitgestellten Konverter können so erweitert werden, dass die Konvertierung von angepassten Datenbankspalten in dynamische Angaben für Personen/Anwärter möglich ist. Für jeden Angabentyp stehen die unten aufgelisteten Schnittstellen zur Verfügung, die sich im Paket 'curam.pdc.impl' befinden. Es können angepasste Implementierungen geschrieben werden, die je nach Angabentyp Gebrauch von diesen Schnittstellen machen.

Konvertererweiterungsschnittstellen (Populator-Schnittstellen)

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

Der Großteil der Schnittstellen verfügt über eine einzige Methode namens `populate`. Für diese Methode können abhängig von den jeweiligen Angabentypen unterschiedliche Parameter angegeben werden.

Die Schnittstellen `PDCBirthAndDeathEvidencePopulator` und `PDCGenderEvidencePopulator` verfügen über die beiden Methoden `populatePerson` und `populateProspectPerson`.

Für `populatePerson` können die folgenden vier Parameter angegeben werden:
- concernRoleKey - Die eindeutige Kennung für die Rolle des Betroffenen, auf die sich diese Angaben beziehen
- caseIDKey - Die eindeutige Kennung für den Fall für Beteiligtendaten
- personDtls - Das Struct, das die erweiterten Details zur Person aus der älteren Tabelle enthält
- dynamicEvidenceDataDetails - Die Details der dynamischen Angaben

Für `populateProspectPerson` können ebenfalls vier Parameter angegeben werden:
- concernRoleKey - Die eindeutige Kennung für die Rolle des Betroffenen, auf die sich diese Angaben beziehen
- caseIDKey - Die eindeutige Kennung für den Fall für Beteiligtendaten
- prospectPersonDtls - Das Struct, das die erweiterten Details zum Anwärter aus der älteren Tabelle enthält
- dynamicEvidenceDataDetails - Die Details der dynamischen Angaben

5.3.4 Beispiel: Implementieren eines Populators für Personen-/Anwärterangaben

Das folgende Beispiel stellt dar, wie ein Konverter so erweitert wird, dass er angepasste Datenbankspalten zu Angaben für Personen/Anwärter zuordnet. Das vorliegende Beispiel liefert eine sehr rudimentäre Implementierung einer Erweiterung von `PDCPhoneNumberEvidencePopulator`. In diesem Szenario wurde die Tabelle namens 'PhoneNumber' erweitert und enthält eine angepasste Spalte 'phoneProvider'. Die dynamische Angabenkonfiguration für die Telefonnummer enthält auch ein Attribut 'phoneProvider'. Die angepasste Populatorimplementierung ist dafür zuständig, das Struct-Attribut, das das Attribut 'phoneProvider' in der erweiterten Tabelle 'PhoneNumber' darstellt, in dynamische Angaben zu konvertieren. Weitere Informationen zur Konfiguration von dynamischen Angaben enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'.

Anmerkung: Es ist lediglich die Konvertierung von Daten erforderlich; das eigentliche Speichern der dynamischen Angaben übernehmen die Standardkonverter.

Zum Erweitern eines Konverters müssen die folgenden Schritte ausgeführt werden:
- Bereitstellen einer Populatorimplementierung, die das angepasste Feld aus der älteren Tabelle in dynamische Angabendaten konvertiert

- Hinzufügen einer Bindung zu der neuen Populatorimplementierung

5.3.4.1 Schritt 1: Bereitstellen einer Populatorimplementierung

Der erste Schritt besteht aus der Bereitstellung einer neuen Implementierung, die die relevante Populatorschnittstelle für den Angabentyp implementiert und das angepasste Feld aus der älteren Tabelle in dynamische Angaben konvertiert. Das nachfolgende Code-Snippet veranschaulicht die angepasste Implementierung für `PDCPhoneNumberEvidencePopulator`: Es konvertiert einfach das Struct-Attribut von 'phoneProvider' in das Attribut, das dynamischen Angaben entspricht. Diese dynamischen Angaben werden dann gemeinsam mit den übrigen Attributen für dynamische Angaben über die Standardkonverterimplementierung gespeichert.

```
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);
  }
}
```

5.3.4.2 Hinzufügen einer Bindung zu der neuen Populatorimplementierung

Die Implementierung wird mit Guice-Bindungen registriert.

```
public class SampleModule extends AbstractModule {

public void configure() {

  // Registrieren der Populatorimplementierung
  Multibinder<PDCPhoneNumberEvidencePopulator> samplePopulator =
  Multibinder.newSetBinder(binder(), PDCPhoneNumberEvidencePopulator.class);

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

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'ModuleClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

5.3.5 Warum sollte ein Konverter implementiert werden?

In Fällen, in denen neue dynamische Angabentypen eingeführt wurden, kann die Implementierung eines neuen Konverters notwendig sein. Konverter werden im Allgemeinen nur in Entwicklungsumgebungen oder als Tools für Upgrades bereitgestellt und sollten nicht im Rahmen der ganz regulären täglichen Verarbeitung eingesetzt werden.

5.3.6 Implementieren eines Konverters

Implementierungen von Konvertern können mit der Schnittstelle `PDCConverter` entwickelt werden. Die Schnittstelle `PDCConverter` befindet sich im Paket 'curam.pdc.impl'. Diese Schnittstelle besitzt eine einzige Methode namens `storeEvidence`. Für sie können die folgenden zwei Parameter angegeben werden:

- concernRoleKey - Die eindeutige Kennung für die Rolle des Betroffenen
- caseIDKey - Die eindeutige Kennung für den Fall für Beteiligtendaten

Im nächsten Abschnitt wird veranschaulicht, wie die Implementierung eines Konverters erfolgt.

5.3.7 Beispiel: Implementieren eines Konverters für Personen-/ Anwärterangaben

Das folgende Beispiel stellt dar, wie die Implementierung eines Konverters erfolgt. In diesem Szenario wurde 'Beispiel für Wohnsitz im Ausland' als neuer dynamischer Angabentyp konfiguriert. Weitere Informationen dazu, wie ein neuer Angabentyp konfiguriert wird, enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'. Der Angabentyp 'Beispiel für Wohnsitz im Ausland' weist die folgenden Attribute auf:

- participant - Die ID der Fallbeteiligtenrolle der Person/des Anwärters, für die/den die Angaben eingegeben werden
- country - Das Land, in dem sich der Wohnsitz befindet
- fromDate - Das Datum, an dem der Aufenthalt an diesem Wohnsitz begann
- toDate - Das Datum, an dem der Aufenthalt an diesem Wohnsitz endete
- reason - Der Grund für den Wohnsitz in diesem Land

Es wurde vorausgesetzt, dass dieser dynamische Angabentyp aktiviert und zur Verwendung mit Personen/Anwärtern konfiguriert wurde. Bislang wurden die Informationen für das 'Beispiel für Wohnsitz im Ausland' als statische Angaben in der Datenbanktabelle 'SampleForeignResidency' gespeichert. Nun ist es erforderlich, diese Informationen als dynamische Angaben zu speichern. Hierzu ist ein neuer Konverter erforderlich, der diese Informationen aus der älteren Tabelle ausliest und als dynamische Angaben speichert.

Zum Implementieren eines Konverters müssen die folgenden Schritte ausgeführt werden:

- Bereitstellen einer Konverterimplementierung, mit der die älteren Daten in dynamische Angaben umgewandelt werden
- Hinzufügen einer Bindung zu der neuen Konverterimplementierung

5.3.7.1 Schritt 1: Bereitstellen einer Konverterimplementierung

Das nachfolgende Code-Snippet veranschaulicht die Implementierung für `PDCConverter`. Es ruft sämtliche Informationen für das 'Beispiel für Wohnsitz im Ausland' für eine Person/einen Anwärter aus der älteren Tabelle 'SampleForeignResidency' ab, konvertiert diese Informationen in die Struktur für dynamische Angabendaten und speichert dann die daraus hervorgehenden Informationen.

```
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);
    }
  }
}
```

5.3.7.2 Schritt 2: Hinzufügen einer Bindung zu der neuen Konverterimplementierung

Die Implementierung wird mit Guice-Bindungen registriert.

```
public class SampleModule extends AbstractModule {

  public void configure() {

  // Registrieren der Konverterimplementierung
  Multibinder<PDCConverter> sampleForeignResidencyConverter =
    Multibinder.newSetBinder(binder(), PDCConverter.class);

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

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'ModuleClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

5.4 Auswahl der Primärinformationen

Ältere Beteiligtenmanager-Entitäten folgen dem Konzept der primären Indikatoren. Hierbei konnten Benutzer bei der Erstellung von Angaben festlegen, welche Kontonummer/Telefonnummer usw. als primäre Daten verwendet werden sollten. Dies gilt nicht für dynamische Angabentypen. Der Benutzer gibt nun nicht mehr an, welche Informationen den primären Datensatz darstellen, vielmehr wird anhand eines Algorithmus im Hintergrund berechnet, welche Informationen als primärer Datensatz geführt werden sollen. Diese Algorithmen basieren auf einer definierten Geschäftsstrategie und können geändert werden. Nähere Details hierzu werden im folgenden Abschnitt dargelegt.

5.4.1 Warum sollte die Auswahl der Primärinformationen geändert werden?

Da die Bestimmung des Primärdatensatzes nicht benutzergesteuert erfolgt, kann es erforderlich sein, diesen Auswahlvorgang zu ändern, wenn die standardmäßige Geschäftsstrategie nicht verwendet werden kann oder soll.

5.4.2 Ändern der Auswahl der Primärinformationen

Die Strategien, mit denen bestimmt wird, welche Daten als Primärinformationen ausgewählt werden sollen, können anhand der standardmäßigen Primary-Handler-Schnittstellen geändert werden. Für jeden angegebenen dynamischen Angabentyp, der in der älteren (Legacy-)Tabelle über eine Primärkennung verfügt, ist eine der unten aufgelisteten Schnittstellen definiert, die im Paket 'curam.pdc.impl' enthalten sind.

Primary-Handler-Implementierungen werden über einen ereignisbasierten Mechanismus aufgerufen. Wenn dynamische Angaben im Anschluss an eine Einfüge-, Änderungs- oder Löschoperation aktiviert werden, wird ein Ereignis ausgelöst. Für neue Angabentypen muss ein Ereignislistener entwickelt werden, der dieses Ereignis überwacht und den entsprechenden Algorithmus aufruft, der die Primärdaten bestimmt. Dieser Vorgang wird an späterer Stelle in diesem Kapitel erörtert. Im nächsten Abschnitt wird veranschaulicht, wie die Implementierung eines Primary Handlers erfolgt.

Primary-Handler-Schnittstellen

- PDCPrimaryAddressHandler
- PDCPrimaryAlternateIDHandler
- PDCPrimaryAlternateNameHandler

- PDCPrimaryBankAccountHandler
- PDCPrimaryEmailAddressHandler
- PDCPrimaryPhoneNumberHandler

5.4.3 Beispiel: Ändern der Auswahl der Primärinformationen

Das folgende Beispiel stellt dar, wie die Implementierung eines Primary Handlers erfolgt. In diesem Szenario gibt die definierte Geschäftsstrategie zum Auswählen einer primären Telefonnummer vor, dass die Telefonnummer mit dem spätesten Anfangszeitpunkt (Gültigkeitsbeginn) ausgewählt wird.

Zum Implementieren eines Primary Handlers müssen die folgenden Schritte ausgeführt werden:

- Bereitstellen einer Primary-Handler-Implementierung, die den Primärdatensatz identifiziert
- Hinzufügen einer Bindung zu der neuen Primary-Handler-Implementierung

5.4.3.1 Schritt 1: Bereitstellen einer Primary-Handler-Implementierung

Der erste Schritt besteht aus der Bereitstellung einer neuen Implementierung, die die relevante Primary-Handler-Schnittstelle für den Angabentyp implementiert und den Primärdatensatz identifiziert. Das nachfolgende Code-Snippet veranschaulicht eine Implementierung für `PDCPrimaryPhoneNumberHandler`: Es sucht einfach die Telefonnummer mit dem spätesten Anfangszeitpunkt (Gültigkeitsbeginn) und legt diese als Primärdatensatz fest.

```
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());
    }
  }
}
```

5.4.3.2 Schritt 2: Hinzufügen einer Bindung zu der neuen Primary-Handler-Implementierung

Die Implementierung wird mit Guice-Bindungen registriert.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Registrieren der Primary-Handler-Implementierung
    bind(PDCPrimaryPhoneNumberHandler.class).to(
      SamplePrimaryPhoneNumberHandlerImpl.class);
  }
}
```

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'ModuleClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

5.5 Reziproke Angaben

5.5.1 Was versteht man unter reziproken Angaben?

Als reziproke Angaben wird ein Typ von Angaben bezeichnet, der sich aus zwei einzelnen Angabenelementen zusammensetzt, die gemeinsam verarbeitet werden müssen. Der dynamische Angabentyp 'Beziehung' ist ein Beispiel für reziproke (wechselseitige) Angaben. Wenn Person A als Ehepartner von Person B erfasst ist, so werden die entsprechenden Beziehungsangaben (Person B ist Ehepartner von Person A) erfasst. Wenn für Person A Angaben eingefügt, geändert oder entfernt werden, so sorgt das System dafür, dass die entsprechenden Beziehungsangaben für Person B ebenfalls eingefügt, geändert oder entfernt werden.

5.5.2 Warum sollte eine Implementierung für reziproke Angaben bereitgestellt werden?

Wenn Sie einen neuen reziproken dynamischen Angabentyp entwickeln, müssen Sie auch eine Implementierung der Schnittstelle `ReciprocalEvidenceConversion` bereitstellen.

5.5.3 Implementierungen für reziproke Angaben

Wenn Angaben eingefügt, geändert oder entfernt werden, so wird ein Ankerpunkt aufgerufen, der standardmäßig die Funktion des Handlers für reziproke Angaben aufruft. Dieser Ankerpunkt für neue Angaben heißt `GlobalEvidenceHook` und ist im Paket 'curam.core.sl.infrastructure.impl' enthalten. Die Schnittstelle `GlobalEvidenceHook` ermöglicht die Durchführung einer angepassten Verarbeitung, nachdem die Angabenoperationen abgeschlossen worden sind.

Schnittstelle 'GlobalEvidenceHook'

Die Schnittstelle `GlobalEvidenceHook` enthält die folgenden Methoden:

Die Methode `postInsertEvidence` wird aufgerufen, nachdem Angaben eingefügt worden sind. Für sie können die folgenden zwei Parameter angegeben werden:

- caseKey - Die Kennung des Falls, zu dem die Angaben gehören
- evKey - Die Angabenkennung und der Angabentyp

Die Methode `postModifyEvidence` wird aufgerufen, nachdem Angaben geändert worden sind. Für sie können die folgenden zwei Parameter angegeben werden:

- caseKey - Die Kennung des Falls, zu dem die Angaben gehören
- evKey - Die Angabenkennung und der Angabentyp

Die Methode `postRemoveEvidence` wird aufgerufen, nachdem Angaben entfernt worden sind. Für sie können die folgenden zwei Parameter angegeben werden:

- caseKey - Die Kennung des Falls, zu dem die Angaben gehören
- evKey - Die Angabenkennung und der Angabentyp

Die Methode `postDiscardPendingUpdate` wird aufgerufen, nachdem eine anstehende Aktualisierung von Angaben verworfen wurde. Für sie können die folgenden zwei Parameter angegeben werden:

- caseKey - Die Kennung des Falls, zu dem die Angaben gehören
- evKey - Die Angabenkennung und der Angabentyp

Die Methode `postDiscardPendingRemove` wird aufgerufen, nachdem eine anstehende Entfernung von Angaben verworfen wurde. Für sie können die folgenden zwei Parameter angegeben werden:

- caseKey - Die Kennung des Falls, zu dem die Angaben gehören
- evKey - Die Angabenkennung und der Angabentyp

Handler für reziproke Angaben

Die Standardimplementierung für `GlobalEvidenceHook` ruft die Funktion des Handlers für reziproke Angaben auf. Der Handler für reziproke Angaben ist für die Verarbeitung aller gemeinsamen reziproken Angaben zuständig. Er spürt reziproke Angaben auf und führt an den gefundenen reziproken Angaben dieselben Änderungen aus, die auch an den ursprünglichen Angaben vorgenommen wurden. Wurden die ursprünglichen Angaben eingefügt, ohne dass hierfür reziproke Angaben gefunden werden können, so fügt der Handler die entsprechenden reziproken Angaben ein. Da der Handler für reziproke Angaben das Kernstück in der Verarbeitung reziproker Angaben darstellt, kann er nicht unmittelbar (direkt) angepasst werden, sondern seine Anpassung müsste bei Bedarf über `GlobalEvidenceHook` erfolgen.

Schnittstelle für die Konvertierung reziproker Angaben

Die Schnittstelle `ReciprocalEvidenceConversion` ist für den Vergleich der reziproken mit den ursprünglichen Angaben zuständig und habe darüber hinaus die Funktion, Beteiligte abzurufen und neue bzw. geänderte reziproke Angaben aus ursprünglichen Angaben zu erstellen. Um angepasste Angaben als reziprok zu definieren, muss eine Implementierung der Schnittstelle `ReciprocalEvidenceConversion` bereitge-

stellt werden. Während dem Handler die interne Struktur der Angaben nicht bekannt ist, ist die Implementierung der Konvertierungsschnittstelle mit dieser sehr wohl vertraut. Dies stellt somit den wichtigsten Ansatzpunkt für die Anpassung dar. Die Schnittstelle `ReciprocalEvidenceConversion` befindet sich im Paket 'curam.core.sl.infrastructure.impl' und umfasst die folgenden Methoden:

- Object getReciprocal(final Object original, final long targetCaseID) - Erstellt reziproke Angabendetails aus den ursprünglichen Angabendetails
- Object getUpdatedReciprocal(final Object original, final Object unmodifiedReciprocal) - Erstellt geänderte reziproke Angabendetails aus den ursprünglichen Angabendetails und aus nicht geänderten reziproken Angabendetails
- long getPrimaryParticipant(final Object originalEvidence) - Ruft den primären Beteiligten (ID der Rolle des Betroffenen) aus den ursprünglichen Angaben ab (Beachten Sie hierbei, dass der primäre Beteiligte in den ursprünglichen Angaben dem zugehörigen Beteiligten in den reziproken Angaben entspricht)
- long getRelatedParticipant(final Object originalEvidence) - Ruft den zugehörigen Beteiligten (ID der Rolle des Betroffenen) aus den ursprünglichen Angaben ab (Beachten Sie hierbei, dass der zugehörige Beteiligte in den ursprünglichen Angaben dem primären Beteiligten in den reziproken Angaben entspricht)
- boolean matchEvidenceDetails(final Object evidenceDetails1, final Object evidenceDetails2) - Überprüft die Angabendetails auf Übereinstimmungen (Die Implementierung dieser Methode bestimmt, ob zwei übergebene Angabendetails als Übereinstimmung gelten)

Im folgenden Abschnitt wird veranschaulicht, wie die Implementierung reziproker Angaben erfolgt.

5.5.4 Beispiel: Implementierung für reziproke Angaben

Das folgende Beispiel veranschaulicht die Implementierung für reziproke Angaben. In diesem Szenario wurde 'Arbeitsverhältnis' als neuer dynamischer Angabentyp konfiguriert. Weitere Informationen dazu, wie ein neuer Angabentyp konfiguriert wird, enthält das Handbuch 'Cúram Dynamic Evidence Configuration Guide'. Der neue Angabentyp für Arbeitsverhältnisse wurde als reziprok definiert und weist die folgenden Attribute auf:

- participant - Die ID der Fallbeteiligtenrolle der Person/des Anwärters, für die/den die Angaben eingegeben werden
- relatedParticipant - Die ID der Fallbeteiligtenrolle der angehörigen Person/des angehörigen Anwärters
- workingRelationship - Das Arbeitsverhältnis zwischen den beiden Beteiligten

Es wurde vorausgesetzt, dass dieser dynamische Angabentyp aktiviert und zur Verwendung mit Personen/Anwärtern konfiguriert wurde. Damit die Infrastruktur diese reziproken Angaben ordnungsgemäß handhaben kann, muss eine Implementierung der Schnittstelle `ReciprocalEvidenceConversion` bereitgestellt werden.

Die folgenden Schritte müssen ausgeführt werden:

- Bereitstellen einer Implementierung für die Konvertierung reziproker Angaben
- Hinzufügen einer Bindung zu der neuen Implementierung für die Konvertierung reziproker Angaben

5.5.4.1 Schritt 1: Bereitstellen einer Implementierung für die Konvertierung reziproker Angaben

```
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;
  }
}
```

5.5.4.2 Schritt 2: Hinzufügen einer Bindung zu der neuen Implementierung für die Konvertierung reziproker Angaben

Die Implementierung wird mit Guice-Bindungen registriert.

```
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);
  }
}
```

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'ModuleClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

5.5.5 Einschränkungen für reziproke Angaben

Für die Infrastruktur zur Handhabung reziproker Angaben gelten die folgenden Einschränkungen:

- Bei den Angaben muss es sich um zeitbezogene Angaben handeln, die statische, dynamische oder generierte Angaben sein können.
- Die Angaben müssen über einen Beteiligten und einen angehörigen Beteiligten verfügen. Alternativ hierzu muss der Code der Implementierung von `ReciprocalEvidenceConversion` in der Lage sein, anhand von Angabendetails den Beteiligten und den angehörigen Beteiligten zu ermitteln.
- Wenn sich reziproke Angaben und die zugehörigen ursprünglichen Angaben in demselben Fall befinden, müssen Änderungen an ihnen stets gemeinsam angewendet werden, denn andernfalls ist die Synchronität der ursprünglichen Angaben und der reziproken Angabendaten nicht mehr gegeben.
- Reziproke Angaben können nur dann automatisch verarbeitet werden, wenn beide verknüpfte (angehörige) Beteiligte in demselben Fall als Mitglieder (MEMBER) oder primäre Beteiligte (PRIMARY) registriert oder wenn Angaben als Personen-/Beteiligtenangaben erfasst sind.

5.6 Eigentümer des Beteiligtendatenfalls

5.6.1 Warum sollte der Eigentümer des Beteiligtendatenfalls geändert werden?

Wenn ein höheres Maß an Kontrolle bezüglich der Eigentümerschaft von Beteiligten erforderlich ist, kann es erforderlich sein, den Eigentümer des Beteiligtendatenfalls zu ändern (wechseln).

5.6.2 Ändern des Eigentümers des Beteiligtendatenfalls

Wenn eine Person/ein Anwärter im System registriert wird, so wird zur leichteren Pflege (Verwaltung) dieser Daten im Hintergrund ein Fall erstellt, der auch als 'Beteiligtendatenfall' bezeichnet wird. Dieser Fall besitzt standardmäßig einen Falleigentümer, und zwar den angemeldeten Benutzer. Über die Schnittstelle `PDCCaseOwnerAssignmentStrategy` ist es möglich, einen anderen Falleigentümer anzugeben. Die Schnittstelle `PDCCaseOwnerAssignmentStrategy` befindet sich im Paket 'curam.pdc.impl' und verfügt über eine einzige Methode namens `createOwner`. Für sie können die folgenden zwei Parameter angegeben werden:

- key - Die Kennung des Beteiligtendatenfalls
- ownerDtls - Die Details zu dem Eigentümer des Beteiligtendatenfalls

5.6.3 Beispiel: Ändern des Eigentümers des Beteiligtendatenfalls

Das folgende Beispiel stellt dar, wie die Änderung des Eigentümers des Beteiligtendatenfalls erfolgt. In diesem Szenario wird der Systembenutzer als Eigentümer definiert.

Die folgenden Schritte müssen ausgeführt werden:

- Bereitstellen einer Implementierung für die Zuweisungsstrategie von Falleigentümern, mit der die Festlegung des Falleigentümers erfolgt
- Hinzufügen einer Bindung zu der Implementierung für die Zuweisungsstrategie von Falleigentümern

5.6.3.1 Schritt 1: Bereitstellen einer Implementierung für die Zuweisungsstrategie von Falleigentümern

Das nachfolgende Code-Snippet veranschaulicht eine Beispielimplementierung für `PDCCaseOwnerAssignmentStrategy`: Es legt als Eigentümer einfach den Systembenutzer fest.

```
@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);
  }
}
```

5.6.3.2 Schritt 2: Hinzufügen einer Bindung zu der Implementierung für die Zuweisungsstrategie von Falleigentümern

Die Implementierung wird mit Guice-Bindungen registriert.

```
public class SampleModule extends AbstractModule {

  public void configure() {

    // Registrieren der Implementierung
    bind(PDCCaseOwnerAssignmentStrategy.class)
   .to(SampleCaseOwnerAssignmentStrategyImpl.class);
  }
}
```

Anmerkung: Neue Guice-Module müssen durch Hinzufügen einer Zeile zur Datenbanktabelle 'Module-ClassName' registriert werden. Weitere Informationen hierzu enthält das Cookbook zu Persistenz.

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden. Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM-Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden. Für die in diesem Handbuch beschriebenen Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing

IBM Europe, Middle East & Africa

Tour Descartes

2, avenue Gambetta

92066 Paris La Defense Cedex

France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden.

Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht. Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Bereitstellung des in diesem Dokument beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen.

IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Alle von IBM angegebenen Preise sind empfohlene Richtpreise und können jederzeit ohne weitere Mitteilung geändert werden. Händlerpreise können u. U. von den hier genannten Preisen abweichen.

Diese Veröffentlichung dient nur zu Planungszwecken. Die in dieser Veröffentlichung enthaltenen Informationen können geändert werden, bevor die beschriebenen Produkte verfügbar sind.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufs. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren und können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten. Die Musterprogramme werden "WIE BESEHEN", ohne Gewährleistung jeglicher Art bereitgestellt. IBM übernimmt keine Haftung für Schäden, die durch Ihre Verwendung der Musterprogramme entstehen.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

© (Name Ihres Unternehmens) (Jahr). Teile des vorliegenden Codes wurden aus Musterprogrammen der IBM Corp. abgeleitet.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farbabbildungen.

Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der International Business Machines Corporation. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Website "Copyright and trademark information" unter http://www.ibm.com/legal/us/en/copytrade.shtml.

Andere Namen können Marken der jeweiligen Rechtsinhaber sein. Weitere Firmen-, Produkt- und Servicenamen können Marken oder Servicemarken anderer Unternehmen sein.

Gedruckt in Deutschland