IBM Cúram Social Program Management
Version 6.0.5

Cúram Express Rules Referenzhandbuch

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen in „Bemerkungen" auf Seite 275 gelesen werden.

Überarbeitung: März 2014

Diese Ausgabe bezieht sich auf IBM Cúram Social Program Management v6.0.5 und alle nachfolgenden Releases, sofern nicht anderweitig in neuen Ausgaben angegeben.

Licensed Materials - Property of IBM.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellen

Referenzinformationen zu Cúram Express Rules

Cúram Express Rules werden verwendet, um Berechnungen durchzuführen. Dazu steht eine Entwicklungsumgebung für das Authoring und Testen der Regelwerke an Cúram Express Rules zur Verfügung. Die Regeln können während der Ausführung ausgeführt werden. Der CER-Editor ist ein Tool für Geschäftsbenutzer und technische Benutzer, um CER-Regelwerke anzuzeigen, zu erstellen und zu verwalten.

Einführung

In diesem Dokument werden die Cúram Express Rules (CER)-Regelsprache, die -Entwicklungsumgebung und die -Laufzeitfeatures beschrieben.

Zielgruppe

Dieses Referenzhandbuch ist für alle Benutzer gedacht, die an der Implementierung von Geschäftsberechnungsregeln mit CER beteiligt sind. Hierzu gehören auch die folgenden Personen:

- Geschäftsanalysten, von denen Anforderungen zusammengestellt werden, die Geschäftsberechnungen einbeziehen. Wenn Sie das Leistungsspektrum und die Konzepte von CER kennen, können Sie Ihre Anforderungen so strukturieren, dass sie einfacher in CER implementiert werden können.
- Regelwerkentwickler, die für die Codierung der Geschäftslogik in einem Regelwerk zuständig sind. Sie müssen die CER-Sprache verstehen, um die Geschäftslogik codieren zu können.
- Tester, von denen die Erfüllung der Anforderungen durch die Implementierung sichergestellt werden muss. Sie müssen die Testunterstützung von CER verstehen, um eine Entscheidung über den Testansatz treffen zu können.

Häufig werden mehrere (wenn nicht sogar alle) dieser Rollen von ein und derselben Person übernommen. Je nach Ihrer Rolle und/oder Ihrem Hintergrund sollten Sie in diesem Handbuch die Kapitel in derjenigen Reihenfolge lesen, die Ihren Anforderungen am besten entspricht.

Zugehörige Literatur

Tabelle 1. Beschreibung der zugehörigen Dokumentation

Dokument	Dokumenttyp	Inhalt
Mit Cúram Express Rules arbeiten	Entwicklerhandbuch	In diesem Handbuch finden Sie Schritt-für-Schritt-Anleitungen für die Erstellung von CER-Regelwerken im CER-Editor, die auch Beispiele mit unterschiedlicher Regelkomplexität beinhalten.

Tabelle 1. Beschreibung der zugehörigen Dokumentation (Forts.)

Dokument	Dokumenttyp	Inhalt
Inside Cúram Eligibility and Entitlement Using Cúram Express Rules	Entwicklerhandbuch	In diesem Handbuch ist beschrieben, wie die Cúram-Engine für Anspruchsberechtigung und Leistungshöhe mit CER interagiert, um Feststellungen für Produktbereitstellungsfälle zu berechnen.
Build eines Produkts	Entwicklerhandbuch	In diesem Handbuch sind alle Tasks beschrieben, die zum Erstellen eines Produkts ausgeführt werden müssen. Dies schließt auch die Zuweisung von CER-Regeln zu einem Produkt ein.
Universal Access Customization Guide	Entwicklerhandbuch	In diesem Handbuch ist beschrieben, wie CER-Regeln im Cúram-Modul "Universal Access" (Universalzugriff) verwendet werden, um Screening-Ergebnisse für Bürger bereitzustellen.

Aufbau dieses Referenzhandbuchs

Dieses Handbuch besteht zum einen aus einer Übersicht und zum anderen aus Referenzmaterial. Sie müssen es *nicht* unbedingt in der Reihenfolge des Inhaltsverzeichnisses lesen.

„Übersicht über CER" auf Seite 3
In diesem Kapitel werden CER, seine Vorzüge und seine Verwendungsmöglichkeiten vorgestellt.

„Entwicklungs- und Testtools von CER" auf Seite 9
In diesem Kapitel sind die Tools beschrieben, mit denen Sie Ihre CER-Regelwerke entwickeln und testen können.

„Verarbeitung von Daten" auf Seite 31
In diesem Kapitel ist beschrieben, wie CER Daten verarbeitet und speichert sowie auf Änderungen an Daten reagiert.

„Abhängigkeitsmanager" auf Seite 81
In diesem Kapitel erfahren Sie, wie die Anwendung erfasst, dass ein berechneter Wert von Eingabewerten abhängig ist, und wie diese Abhängigkeitsdatensätze zur Unterstützung automatischer Berechnungen eingesetzt werden,

„Informationen zum CER-Editor" auf Seite 105
In diesem Kapitel sind die verschiedenen Bestandteile des CER-Editors beschrieben.

„Referenzinformationen zu Regelelementen in Paletten des CER-Editors" auf Seite 119
In diesem Kapitel ist detailliert beschrieben, wie die Elemente eines Regelwerks im CER-Editor erstellt werden.

Übersicht über CER

Eine kurze Übersicht über CER, einschließlich Konzepte, Vorteile und Richtlinien.

CER - Beschreibung

CER ist

- eine Sprache, in der die Regeln für Geschäftsberechnungen (in so genannten "Regelwerken") ausgedrückt werden können,
- eine Entwicklungsumgebung für das Authoring und den Test dieser Regelwerke und
- eine Laufzeitumgebung für die Ausführung von Regeln.

Regelsprache

CER ist eine Sprache, in der Sie mögliche Fragen sowie die Regeln zur Ermittlung der Antworten auf diese Fragen definieren können.

Für jede Frage wird Folgendes angegeben:

- Name
- Typ der Daten, die die Antwort auf die Frage bereitstellen
- Regeln für die Bereitstellung der Antwort (auf die gestellte Frage)

Die Antwort auf eine Frage (z. B. "Ist diese Person leistungsberechtigt?") kann ein einfaches "Ja" oder "Nein" sein. Sie können jedoch die Antworttypen so komplex wie benötigt definieren. So könnte beispielsweise die Frage "Welche Personengruppen im Haushalt haben einen vordringlichen Bedarf?" durch die Bereitstellung einer Liste mit Haushaltsgruppen beantwortet werden, bei der jede Haushaltsgruppe eine Liste von Personen enthält.

Auch die Regeln, mit denen die Antwort auf eine Frage ermittelt wird, können so einfach oder so komplex wie nötig sein. Beispielsweise ist die Regel für die Beantwortung der Frage "Wie lautet das Geburtsdatum des Anspruchstellers?" wahrscheinlich (ganz einfach) "Datum, das der Anspruchsteller als Geburtsdatum angegeben hat", während die Regel für die Beantwortung der Frage "Ist diese Person leistungsberechtigt?" wahrscheinlich weitere Fragen nach sich zieht (z. B. "Wie hoch ist das Einkommen dieser Person?" oder "Wie viele Kinder hat diese Person?").

CER besitzt für diese Konzepte eine eigene Terminologie:

- **Regelklasse**

  Eine Regelklasse ist ein Typ für eine "Sache", über die es Daten gibt, z. B. eine Person (Klasse "Person"), ein Einkommen (Klasse "Income") oder eine Forderung bzw. ein Anspruch (Klasse "Claim"). Neue Regelklassen können im CER-Editor erstellt werden. Weitere Informationen finden Sie im Abschnitt „Technische Ansicht" auf Seite 108.

- **Regelobjekt**

  Ein Regelobjekt ist eine Instanz einer Regelklasse, beispielsweise "Thomas Schmidt" (Klasse "Person"), "Einkommen von Thomas Schmidt aus einer Teilzeittätigkeit" (Klasse "Income") oder "Antrag von Thomas Schmidt auf Kindesunterhalt" (Klasse "Claim").

- **Regelattribut**

  Ein Regelattribut ist eine Frage, die gestellt werden kann. Eine solche Frage wird für eine Regelklasse definiert und kann für jedes Regelobjekt dieser Klasse gestellt werden. Beispiel: Die Regelklasse "Person" kann das Regelattribut "dateOfBirth" (= Geburtsdatum) definieren und für das Regelobjekt "Thomas Schmidt" kann folglich die Frage nach dem Geburtsdatum gestellt werden (z. B. 3. Oktober 1970). Neue Attribute können für die ausgewählte Regelklasse im CER-Editor erstellt werden. Weitere Informationen finden Sie im Abschnitt „Technische Ansicht" auf Seite 108.

- **Ausdruck**

  Ein Ausdruck ist ein Berechnungsschritt, der zum Beantworten einer Frage verwendet werden kann. Beispiel: Ist die Anspruchsberechtigung einer Person davon abhängig, dass das Gesamteinkommen der Person einen bestimmten Grenzwert nicht überschreitet, kann mithilfe eines Summenausdrucks ("sum") das Gesamteinkommen berechnet und diese Summe anschließend mit einem Vergleichsausdruck ("compare") mit dem Grenzwert verglichen werden. Zur Erstellung eines Ausdrucks kann ein Regelelement "Summe" im CER-Editor auf das Regelattribut gezogen werden. Weitere Informationen finden Sie im Abschnitt „Geschäftsansicht" auf Seite 107.

- **Regelwerk**

  Ein Regelwerk ist eine Sammlung von Regelklassen, die normalerweise auf einen bestimmten Zweck konzentriert sind (z. B. könnte ein Regelwerk zur Ermittlung von Leistungsbezügen für Kinder die Regelklassen "Claim", "Person" und "Income" enthalten). Neue Regelwerke können im Abschnitt "Regeln und Angabe" der Schnittstelle "Administration" erstellt werden.

  Anmerkung: Seit Cúram Version 6 sind Regelwerke nicht mehr eigenständig. Eine Klasse in einem Regelwerk kann eine Regelklasse aus einem anderen Regelwerk erweitern. Der Datentyp eines Regelattributs in einem Regelwerk kann eine Regelklasse aus einem anderen Regelwerk sein. Ausdrücke für das Lesen oder Erstellen von Regelobjekten können Regelklassen aus anderen Regelwerken verwenden.

- **Regelsitzung**

  Eine Regelsitzung steuert die Verarbeitung von Regeln. Ihre Anwendung kann beispielsweise eine Regelsitzung erstellen, um die Anspruchsberechtigung von Thomas Schmidt hinsichtlich von Leistungsbezügen für Kinder zu ermitteln, indem das entsprechende Regelwerk aufgerufen wird und die Fragen zur Anspruchsberechtigung gestellt werden, die sich auf die persönliche Situation von Thomas Schmidt beziehen.

Umgebung für Authoring und Test

CER-Regelwerke werden im CER-Editor erstellt und verwaltet. Sie werden als XML-Daten in der Anwendungsdatenbank gespeichert. Die XML-Daten für ein CER-Regelwerk richten sich nach dem von CER bereitgestellten Regelschema.

CER enthält darüber hinaus einen umfassenden Regelwerkvalidierer, der Fehler in einem Regelwerk erkennen kann, bevor die Regeln ausgeführt werden. Sie können ein Regelwerk im CER-Editor validieren. Weitere Informationen finden Sie im Abschnitt „Globales Menü des CER-Editors" auf Seite 106.

CER unterstützt die Ausführung von Regelsitzungen in den folgenden Umgebungen:

- Produktionsumgebungen, in denen CER bei Ihrer Anwendung integriert ist, um bei Bedarf Fragen zu beantworten.
- Eigenständige Testumgebungen, in denen Sie reproduzierbare automatisierte Tests für Ihre Regelwerke erstellen.

CER-Regelwerke sind vollständig dynamisch. In Produktionsumgebungen unterstützt CER den Upload von Änderungen an Regelwerken, die bei ihrer Veröffentlichung wirksam werden und einen erneuten Build oder eine erneute Implementierung der Anwendung überflüssig machen.

Der Test von CER-Regelwerken kann auf jeder von Ihnen gewünschten Stufe stattfinden. Sie können detaillierte Testdaten für ein vollständiges Geschäftsszenario bereitstellen, aber auch isolierte Tests für Teile Ihres Regelwerks erstellen, *ohne* hierzu ein umfangreiches Volumen von Eingabedaten sorgfältig zusammenstellen zu müssen.

Beispiel: Um festzustellen, ob eine Person einen Anspruch auf Leistungsbezug für Kinder hat, muss möglicherweise eine komplexe Berechnung durchgeführt werden, die (unter anderem) auch das Gesamteinkommen der Person mit einem bestimmten Grenzwert vergleichen muss. Die Berechnung des Gesamteinkommens der Person ist darüber hinaus *selbst* ebenfalls eine komplexe Berechnung, die Entscheidungen darüber beinhaltet, ob bestimmte Einkommensarten im Zusammenhang mit dem Anspruch auf Leistungsbezug für Kinder anrechenbar sind.

Wenn die Berechnung der Anspruchsberechtigung getestet wird, müssen Sie bei der herkömmlichen Entwicklung unter Umständen sorgfältig Einkommensdaten zusammenstellen, damit ein Gesamteinkommen berechnet werden kann, das Sie anschließend verwenden können, um die Berechnung der Anspruchsberechtigung zu testen. Je nach der Komplexität der Berechnungen kann die Zusammenstellung solcher Daten in der Ausführung sehr mühsam und im Ergebnis umständlich zu ändern sein.

Bei CER können Sie hingegen einfach eine Berechnung auslösen, ohne untergeordnete Detaildaten bereitstellen zu müssen. In CER ist es außerordentlich unkompliziert, einen Test zu erstellen, der effektiv besagt "Das zum Zweck dieses Tests verwendete Gesamteinkommen beträgt 10 Euro - versuchen Sie *nicht*, das Gesamteinkommen während dieses Tests zu berechnen".

Diese Funktionalität von CER vereinfacht das Testen aller Funktionen in einem Regelwerk auf einer für Sie sinnvollen Ebene.

Die Authoring-Umgebung bietet ebenfalls Tools, die Sie beim Entwickeln und Testen von Regeln unterstützen:

- **RuleDoc**

  Hierbei handelt es sich um einen HTML-Auszug der Struktur in den Regelwerken.
- **SessionDoc**

  Hierbei handelt es sich um eine HTML-Darstellung der Daten in Ihren Regelobjekten.
- **Tool für Abdeckung**

  Dieses Tool meldet, in welchem Umfang ein Regelwerk durch die Tests untersucht wurde.

Laufzeitumgebung

CER führt Regeln während der Laufzeit bedarfsgerecht aus.

Seit Cúram Version 6 bietet CER darüber hinaus Funktionen für die folgenden Aufgaben:

- Speicherung von Regelobjekten in der Datenbank, damit Regelobjekte für die künftige Verarbeitung verfügbar sind.
- Integration beim Abhängigkeitsmanager, damit festgestellt werden kann, wenn sich Eingabedaten geändert haben, und damit Berechnungsergebnisse, die auf diesen Eingabedatenelementen basieren (ähnlich wie bei der herkömmlichen Tabellenkalkulationsverarbeitung), automatisch neu berechnet werden.

Vorteile von CER

CER bietet die folgenden Hauptvorteile:

- **Einfachheit**

  CER-Regelwerke sind nie komplexer als Ihre Geschäftsanforderungen. Geschäftsbenutzer und technische Benutzer können gleichermaßen CER-Regeln lesen und die ausgeführten Aktionen verstehen. Das Schreiben und Testen von Regelwerken ist denkbar einfach.
- **Flexibilität**

  Regelwerke können ohne großen Aufwand geändert werden. Sie können jederzeit neue Fragen hinzufügen. CER stellt sicher, dass das bestehende Verhalten hierdurch nicht beeinträchtigt wird. Sie können auch das Verfahren für die Beantwortung vorhandener Fragen ändern. CER zeigt Ihnen in einem solchen Fall, welche Berechnungen von dieser Frage abhängig sind, damit Sie die Auswirkungen Ihrer Änderungen vollständig erfassen können.
- **Lokalisierungsunterstützung**

  CER kann lokalisierbare Ausgabe erzeugen. Antworten auf Fragen können daher für die Endbenutzer gemäß ihren Vorgaben für Sprache und Ländereinstellung angezeigt werden.
- **Gültigkeit**

  CER sucht intensiv nach Fehlern im Regelwerk, bevor es ausgeführt wird. Der CER-Regelwerkvalidierer meldet so viele Fehler wie möglich, damit Sie sie in einem Arbeitsgang beheben können. CER stellt technische Probleme in einem Regelwerk fest, damit Sie sich ganz auf die Funktionalität Ihres Regelwerkes konzentrieren können.
- **Testfähigkeit**

  Sie können Ihre CER-Regelwerke mit dem von Ihnen gewünschten Detaillierungsgrad testen. Bei CER behalten Sie auch über umfangreiche Regelwerke immer die Kontrolle, da Sie für separate Abschnitte der Regeln eigene Tests erstellen können.

- **Dynamische Unterstützung**

  Sie können Änderungen an einem CER-Regelwerk auf einem aktiven System vornehmen. Ihre Änderungen werden bei ihrer Veröffentlichung unverzüglich wirksam.
- **Verhaltensähnlichkeit mit Tabellenkalkulation**

  Die Erstellung von CER-Regeln ähnelt der Schichtung von Formeln in den Zellen eines Tabellenkalkulationsprogramms (die sicher vielen Benutzern vertraut ist). Wenn Eingabedaten (z. B. Angaben, persönliche Daten oder Zahlungssätze) geändert werden, stellt CER eine Integration beim Abhängigkeitsmanager her, damit die abgeleiteten Werte, die von der Änderung betroffen sind, automatisch neu berechnet werden.

Einsatzmöglichkeiten

Die CER-Entwicklungsumgebung ist nahtlos in die übergeordnete Anwendungsentwicklungsumgebung integriert.

CER wird von einer Reihe von Anwendungsbereichen verwendet, zu denen unter anderem Folgende gehören:

- **Cúram-Modul "Universal Access"**

  Das Cúram-Modul "Universal Access" (Universalzugriff) ermittelt gestützt auf CER eine potenzielle Anspruchsberechtigung für Sozialeinrichtungsprogramme, wenn Bürger mit dem Self-Service-Modul ein Self-Screening durchführen. Anhand der erfassten Daten bestimmt CER, für welche Programme ein Bürger möglicherweise infrage kommt und liefert (in der Sprache des Bürgers) einen Text, der erläutert, *warum* der betreffende Bürger potenziell leistungsberechtigt ist oder nicht.
- **Berater**

  Der Berater berechnet anhand von CER-Regeln eine Beratung, die für Benutzer angezeigt wird. Diese Beratung wird bei geänderten Bedingungen automatisch aktualisiert.
- **Engine für Anspruchsberechtigung und Leistungshöhe**

  Die Cúram-Engine für Anspruchsberechtigung und Leistungshöhe stellt dank der nahtlosen Integration bei CER für einen Fall die Anspruchsberechtigung und Leistungshöhe (sowie eine Erläuterung über die Ableitung dieser Fakten) für die gesamte Lebensdauer des Falls fest. Die Engine für Anspruchsberechtigung und Leistungshöhe stützt sich auf die Integration zwischen CER und dem Abhängigkeitsmanager und erkennt mit ihrer Hilfe, wann die Feststellung eines Falls (entweder aufgrund von fallspezifischen Änderungen wie Angaben oder wegen umfangreicherer Datenänderungen wie beispielsweise Änderungen an persönlichen Daten oder produktweiten Satzdaten) neu berechnet werden muss.

Mit CER können Sie auch Berechnungen für eigene benutzerdefinierte Geschäftsbereiche ausführen.

Die CER-Laufzeit besitzt keine integrierten Geschäftskonzepte. Jeder Geschäftsbereich kommuniziert vielmehr über die folgenden Mittel mit CER:

- Schnittstellenregelklassen, in die die Geschäftskonzepte eingeschlossen sind, und/oder
- Erweiterungen der CER-Sprache, die Geschäftskonzepte verwenden können.

Details über die Nutzung von CER durch Anwendungskomponenten können Sie den Geschäfts- und Technikhandbüchern für die jeweiligen Komponenten entnehmen.

Richtlinien

Im Kern stützt sich CER auf bestimmte Schlüsselprinzipien. Basiskenntnisse über diese Prinzipien erleichtern Ihnen das Verständnis der Verfahrensweise von CER:

- **Fragen werden nur bei Bedarf beantwortet**

  Die Arbeit, die zur Beantwortung einer Frage erforderlich ist, wird nur dann ausgeführt, wenn die Frage gestellt wird.

- **Sämtliche Daten sind unveränderlich**

  Die Antwort auf eine Frage ist ein Wert, der nicht außerhalb von CER versehentlich geändert werden kann. Falls die Antwort auf eine Frage neu berechnet wird, wird ein neuer Antwortwert erzeugt.

- **Mehrere Fragen sind möglich**

  Ein Regelwerk wird nicht ein Mal von Anfang bis Ende komplett ausgeführt, sondern ermöglicht vielmehr, dass so viele Fragen wie nötig gestellt werden können.

- **Regeln werden angegeben, nicht die Ausführungsreihenfolge**

  Sie geben die Regeln für die Beantwortung einer Frage an. Die effiziente Beantwortung dieser Fragen zur Laufzeit bleibt CER überlassen.

- **Keine Flüchtigkeit**

  Identische Eingabedaten, die von identischen Regeln verarbeitet werden, erzeugen immer dieselbe Ausgabe.

- **Kein "Arbeitsspeicher"**

  Es gibt keine Zähler oder laufenden Summen. Ein Zähler oder eine Summe ist eine selbstständige Frage - Sie geben Regeln für ihre Beantwortung an, CER sorgt dann für die erforderliche Ausführungsreihenfolge, wenn die Frage beantwortet wird.

- **Namen nur bei Bedarf**

  Sie müssen lediglich Namen für Geschäftskonzepte und Fragen angeben. Gedanken über beschreibende Namen für Zwischenergebnisse müssen Sie sich nicht machen (es sei denn, Sie möchten).

- **Entwicklung und Test gehen Hand in Hand**

  CER stellt eine leistungsfähige Unterstützung für die Verwaltung der Tests Ihrer Regeln bereit.

- **Keine integrierten Geschäftskonzepte**

  Die CER-Laufzeit enthält absichtlich keine Geschäftskonzepte. Sie definieren die benötigten Geschäftskonzepte, was die CER-Laufzeit als vielseitig einsetzbare Umgebung erhält.

- **Regelimplementierung ist eng auf Regelanforderungen abgestimmt**

  Die Implementierung Ihrer Regelanforderungen ist so komplex wie diese Anforderungen - *aber nie komplexer*. CER-Regelwerke sind für die Geschäftsanalysten, die die ursprünglichen Anforderungen zusammengestellt haben, sinnfällig.

- **Nutzung der anerkannten Java-Unterstützung**

  Für CER wurde nicht das Rad neu erfunden - die von der bestehenden Java™-Technologie bereitgestellte Funktionalität wird in CER-Regelwerken einfach wiederverwendet.

- **Verwaltung von Berechnungsabhängigkeiten**

  CER ist beim Abhängigkeitsmanager integriert, um Berechnungsabhängigkeiten für Sie automatisch zu verwalten. Wenn sich ein Eingabedatenelement ändert, wissen der Abhängigkeitsmanager und CER, was neu berechnet werden muss.

Sie müssen keine besondere Verarbeitung schreiben, die erst feststellen muss, welche berechneten Ausgaben betroffen sein *könnten*.

Entwicklungs- und Testtools von CER

In diesem Abschnitt sind die Tools beschrieben, mit denen Sie Ihre CER-Regelwerke entwickeln und testen können.

CER-Regeleditor

Der CER-Editor stellt eine benutzerfreundliche Umgebung bereit, in der Techniker und Geschäftsbenutzer gleichermaßen ein Regelwerk und seine Regelklassen erstellen, bearbeiten und validieren können.

Informationen zur Verwendung des CER-Editors finden Sie im Abschnitt „Informationen zum CER-Editor” auf Seite 105.

Lokalisierungsunterstützung

Eine Beschreibung der Lokalisierung in CER.

Die Lokalisierung in CER dient zwei verschiedenen Zwecken:

- Lokalisierung von berechneten Daten, die von einem CER-Regelattribut zurückgegeben werden, damit die Ausgabe für Benutzer mit unterschiedlichen Ländereinstellungen angezeigt werden kann.
- Lokalisierung der Beschreibungen von Artefakten in den CER-Regelwerken, damit Benutzer die Regelwerke im CER-Editor in ihrer eigenen Ländereinstellung anzeigen können.

Diese Aspekte werden in den nachfolgenden Abschnitten detaillierter behandelt.

Wichtig: Die *Namen* von Regelwerkelementen wie beispielsweise Regelklassen und -attributen können *nicht* lokalisiert werden, weil sie in der CER-Sprache intern als Kennungen verwendet werden (ein Beispiel hierfür ist der Name eines Attributs in einem Ausdruck `reference`).

Die *Beschreibungen* von Regelelementen können hingegen lokalisiert werden, wobei die Namen der Elemente nicht geändert werden.

Lokalisierung von berechneten Daten

CER unterstützt die konventionelle Java-Klasse `String`.

Elemente der Klasse "String" (also Zeichenfolgen) können in den Anfangsstadien bei der Entwicklung eines Regelwerks hilfreich sein. Falls Ihre Regeln jedoch Ausgabe enthalten, die für Benutzer in verschiedenen Ländereinstellungen angezeigt werden muss, müssen Sie unter Umständen die CER-Unterstützung für die Lokalisierung verwenden.

Der Zeichenfolgewert „Hello, world” im nachfolgenden Beispiel ist für Benutzer, die Englisch als Spracheinstellung verwenden, gut lesbar. Es gibt jedoch auch Benutzer, die eine andere Sprache verwenden.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="HelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">
```

```
    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

CER enthält eine Schnittstelle namens `curam.creole.value.Message`, die während der Laufzeit die Konvertierung eines Werts in eine ländereinstellungsspezifische Zeichenfolge ermöglicht.

Eine Liste der CER-Ausdrücke, die eine Instanz der Schnittstelle `curam.creole.value.Message` erstellen können, finden Sie im Abschnitt „Lokalisierbare Nachrichten" auf Seite 170.

Das Beispiel `HelloWorld` wird nun umgeschrieben, damit es lokalisierbar ist:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="LocalizableHelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <!-- Use Message, not String -->
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Look up the value from a localizable
             property, instead of hard-coding a
             single-language String -->
        <ResourceMessage key="greeting"
          resourceBundle="curam.creole.example.HelloWorld"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Lokalisierung von „Hello, world!" in Englisch.

```
# file curam/creole/example/HelloWorld_en.properties

greeting=Hello, world!
```

Lokalisierung von „Hello, world!" in Französisch.

```
# file curam/creole/example/HelloWorld_fr.properties

greeting=Bonjour, monde!
```

Diese Nachricht verhält sich während der Laufzeit nun folgendermaßen: Jeder Code, der mit Ihrem Regelwerk interagiert, muss die Methode `toLocale` für alle Nachrichten aufrufen, um diese in die erforderliche Ländereinstellung zu konvertieren.

Das nachfolgende Beispiel zeigt die Interaktion mit dem lokalisierten Regelwerk.

```
package curam.creole.example;

import java.util.Locale;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.LocalizableHelloWorldRuleSet.impl.HelloWorld;
import
 curam.creole.ruleclass.LocalizableHelloWorldRuleSet.impl.HelloWorld_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Message;

public class TestLocalizableHelloWorld extends TestCase {

  /**
   * Runs the class as a stand-alone Java application.
   */
  public static void main(final String[] args) {

    final TestLocalizableHelloWorld testLocalizableHelloWorld =
        new TestLocalizableHelloWorld();
    testLocalizableHelloWorld.testLocalizedRuleOutput();

  }

  /**
   * A simple test case, displaying output localized into different
   * locales.
   */
  public void testLocalizedRuleOutput() {

    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final HelloWorld helloWorld =
        HelloWorld_Factory.getFactory().newInstance(session);

    // returns a Message, not a String
    final Message greeting = helloWorld.greeting().getValue();

    // to decode the message, we need to use the user's locale
    final String greetingEnglish =
        greeting.toLocale(Locale.ENGLISH);
    final String greetingFrench = greeting.toLocale(Locale.FRENCH);

    System.out.println(greetingEnglish);
    System.out.println(greetingFrench);

    assertEquals("Hello, world!", greetingEnglish);
    assertEquals("Bonjour, monde!", greetingFrench);
  }

}
```

Falls die folgenden Datentypen in einer lokalisierbaren Nachricht eingesetzt werden, werden sie während der Laufzeit so formatiert, dass sie für die Ländereinstellung verwendet werden können:

- Regelobjekte (unter Verwendung des Wertes für das Attribut `description` des Regelobjekts)
- Datumsangaben (unter Verwendung von `curam.util.type.Date`)
- Codetabellenelemente
- Verschachtelte lokalisierbare Nachrichten

Alle anderen Objekte werden mittels ihrer eigenen Methode `toString` angezeigt.

Lokalisierung der Beschreibungen von CER-Regelartefakten

Im CER-Editor können Sie für die folgenden Regelwerkartefakte (über die Anmerkung "Label" bzw. "Beschriftung") eine Beschreibung bereitstellen:

- Regelwerk
- Regelklasse
- Regelattribut
- Ausdruck

Wenn ein CER-Regelwerk unter Verwendung der Cúram-Verwaltungsanwendung veröffentlicht wird, werden die Beschreibungen dieser Regelwerkartefakte im Ressourcenspeicher der Anwendung als Eigenschaftendateien (namens `RULESET-(regelwerkname)(regelwerkversionsnummer)` gespeichert. Sie können diese Eigenschaftendateien wie bei allen anderen Ressourcen im Ressourcenspeicher lokalisieren.

Die Unterstützung für die Lokalisierung von CER-Regelartefakten über den CER-Editor wird in einem künftigen Release Bestandteil des CER-Editors sein.

Regelwerkvalidierer

CER enthält einen Validierer, der die Struktur Ihrer Regelwerke überprüft. Für gewöhnlich validieren Sie Ihre Regelwerke in der Cúram-Verwaltungsanwendung oder im CER-Editor.

Mit dem nachstehenden Befehl können Sie die Struktur von Regelwerken im Dateisystem validieren.

```
build creole.validate.rulesets
```

Auf dem Ziel wird der CER-Regelwerkvalidierer für Ihre Regelwerke ausgeführt. Er meldet alle Fehler und/oder Warnungen für Ihre CER-Regelwerke.

Tipp: Der CER-Regelwerkvalidierer meldet außerdem alle Warnungen über nicht kritische Probleme in Ihren Regelwerken. Diese Warnungen verhindern weder Ausführung noch Test der Regeln, sollten jedoch berücksichtigt werden, um ein optimales Regelwerk zu gewährleisten.

Regelwerkinterpreter

CER enthält einen Interpreter, der dynamisch definierte Regelwerke ausführen kann.

Im folgenden Beispielcode wird der CER-Regelwerkinterpreter verwendet, um Regeln aus dem Regelwerk namens `HelloWorldRuleSet` auszuführen.

```
package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.execution.RuleObject;
```

```
import curam.creole.execution.session.InterpretedRuleObjectFactory;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import curam.creole.parser.RuleSetXmlReader;
import curam.creole.ruleitem.RuleSet;
import curam.creole.storage.inmemory.InMemoryDataStorage;

public class TestHelloWorldInterpreted extends TestCase {

  /**
   * Runs the class as a stand-alone Java application.
   */
  public static void main(final String[] args) {

    final TestHelloWorldInterpreted testHelloWorld =
        new TestHelloWorldInterpreted();
    testHelloWorld.testUsingInterpreter();
  }

  /**
   * Reads the HelloWorldRuleSet from its XML source file.
   */
  private RuleSet getRuleSet() {

    /* The relative path to the rule set source file */
    final String ruleSetRelativePath = "./rules/HelloWorld.xml";

    /* read in the rule set source */
    final RuleSetXmlReader ruleSetXmlReader =
        new RuleSetXmlReader(ruleSetRelativePath);

    /* dump out any problems */
    ruleSetXmlReader.validationProblemCollection().printProblems(
        System.err);

    /* fail if there are errors in the rule set */
    assertTrue(!ruleSetXmlReader.validationProblemCollection()
        .containsErrors());

    /* return the rule set from the reader */
    return ruleSetXmlReader.ruleSet();
  }

  /**
   * A simple test case, using the fully-dynamic CER rule set
   * interpreter.
   */
  public void testUsingInterpreter() {

    /* read in the rule set */
    final RuleSet ruleSet = getRuleSet();

    /* start a session which creates interpreted rule objects */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new InterpretedRuleObjectFactory()));

    /* create a rule object instance of the required rule class */
    final RuleObject helloWorld =
        session.createRuleObject(ruleSet.findClass("HelloWorld"));

    /*
     * Access the "greeting" rule attribute on the rule object -
     * the result must be cast to the expected type (String)
```

```
     */
    final String greeting =
        (String) helloWorld.getAttributeValue("greeting")
            .getValue();

    System.out.println(greeting);
    assertEquals("Hello, world!", greeting);
  }

}
```

Sie können diese Beispielklasse entweder als eigenständige Java-Anwendung (d. h. über ihre Methode `main`) oder als JUnit-Test ausführen. Diese beiden Möglichkeiten zur Ausführung dieser Klasse werden ausschließlich aus Gründen des Benutzungskomforts bereitgestellt. Verwenden Sie beim Schreiben von eigenem Code für die Ausführung von Regelwerken Ihr bevorzugtes Verfahren.

Dieser Code soll nachfolgend detaillierter betrachtet werden. Die Testmethode `testUsingInterpreter` führt die folgenden Schlüsselfunktionen aus:

- Das Regelwerk wird eingelesen.
- Es wird eine CER-Sitzung gestartet.
- In der Sitzung wird eine neue Regelobjektinstanz erstellt.
- Die Regeln werden ausgeführt, indem aus dem Regelobjekt der Wert eines Attributs abgerufen wird.

Regelwerk einlesen

```
/* read in the rule set */
              final RuleItem_RuleSet ruleSet = getRuleSet();
```

Diese Zeile ruft eine Dienstprogrammmethode auf, um das Regelwerk aus einer XML-Quellendatei einzulesen.

Das Regelwerk wird explizit validiert, um sicherzustellen, dass es fehlerfrei ist:

```
/* dump out any problems */
              ruleSetXmlReader.validationProblemCollection().printProblems(
              System.err);

              /* fail if there are errors in the rule set */
              assertTrue(!ruleSetXmlReader.validationProblemCollection()
              .containsErrors());
```

CER-Sitzung starten

```
/* start a session which creates interpreted rule objects */
              final Session session =
              Session_Factory.getFactory().newInstance(
              new RecalculationsProhibited(),
              new InMemoryDataStorage(
              new InterpretedRuleObjectFactory()));
```

Diese Zeilen erstellen eine neue CER-Sitzung für das Regelwerk.

Eine Sitzung verwaltet die Regelobjekte, die für die Klassen im Regelwerk erstellt werden. Im vorliegenden Beispiel wird eine Sitzung verwendet, die vollständig dynamische Regelobjekte erstellt (mittels `InterpretedRuleObjectFactory`). Wie nachfolgend gezeigt, erfolgt in einer Sitzung mit Interpreter jeder Verweis auf eine Regelklasse oder einen Attributnamen über einen API-Aufruf, der diese Namen als Zeichenfolgeparameter verwendet.

Neues Regelobjekt erstellen

```
/* create a rule object instance of the required rule class */
             final RuleObject helloWorld =
             session.createRuleObject("HelloWorld");
```

Diese Zeile erstellt ein neues Regelobjekt (eine Instanz der Regelklasse "HelloWorld") und speichert das Regelobjekt im Hauptspeicher der CER-Sitzung.

Regeln ausführen

```
/*
   * Access the "greeting" rule attribute on the rule object -
   * the result must be cast to the expected type (String)
   */
   final String greeting =
   (String) helloWorld.getAttributeValue("greeting")
   .getValue();
```

Diese Zeile ruft den Wert des Attributs "greeting" aus dem zuvor erstellten Regelobjekt ab.

Wenn der Wert des Attributs angefordert wird, führt CER die Regeln für die Ableitung des Attributwerts aus (und gibt in diesem Fall die konstante Zeichenfolge "Hello, world!" zurück).

Anmerkung: Bei der Ausführung einer Sitzung mit Interpreter müssen Sie die Ausgabe von `getValue` in den erwarteten Datentyp umsetzen.

Im obigen Beispiel wird lediglich der Wert eines einzigen Attributs angefordert. Solange die Sitzung aktiv ist, kann mit entsprechendem Code jedoch der Wert jedes beliebigen Attributs für jedes beliebige Regelobjekt in der Sitzung angefordert werden. CER merkt sich bereits berechnete Werte und führt eine Berechnung nur dann aus, wenn sie zum ersten Mal angefordert wird.

CER-Testcodegenerator

CER enthält einen Codegenerator, der für Ihre Regelklassen Java-Wrapperklassen generieren kann. Diese generierten Klasssen können das Schreiben des Testcodes vereinfachen und dem Comiler die Erkennung von Problemen ermöglichen, die andernfalls erst zur Laufzeit auftreten würden.

Der CER-Regelwerkinterpreter lässt über Zeichenfolgen Verweise auf Regelklassen- und Attributnamen zu. Dies ermöglicht zwar eine vollständig dynamische Konfiguration von Regelwerken, aber es kann im Rahmen von Tests umständlich sein, Zeichenfolgen zu verwenden und Attributwerte umzusetzen. Falls Sie eine Regelklasse oder einen Attributnamen falsch eingeben bzw. den falschen Umsetzungstyp verwenden, wird der Code zwar möglicherweise problemlos kompiliert, führt jedoch während der Laufzeit zu Fehlern.

Der Code für die Ausführung des Regelwerks "HelloWorldRuleSet" wird nun wie folgt umgeschrieben, um zu zeigen, wie Regeln mit CER-generierten Testregelklassen ausgeführt werden:

```
package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
```

```
import curam.creole.ruleclass.HelloWorldRuleSet.impl.HelloWorld;
import
 curam.creole.ruleclass.HelloWorldRuleSet.impl.HelloWorld_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;

public class TestHelloWorldCodeGen extends TestCase {

  /**
   * Runs the class as a stand-alone Java application.
   */
  public static void main(final String[] args) {

    final TestHelloWorldCodeGen testHelloWorld =
        new TestHelloWorldCodeGen();
    testHelloWorld.testUsingGeneratedTestClasses();

  }

  /**
   * A simple test case, using the CER-generated test classes for
   * strong typing and ease of coding tests.
   */
  public void testUsingGeneratedTestClasses() {

    /* start a strongly-typed session */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    /*
     * create a rule object instance of the required rule class, by
     * using its generated factory
     */
    final HelloWorld helloWorld =
        HelloWorld_Factory.getFactory().newInstance(session);

    /*
     * use the generated accessor to get at the "greeting" rule
     * attribute - no cast necessary, and any error in the
     * attribute name would lead to a compile error
     */
    final String greeting = helloWorld.greeting().getValue();

    System.out.println(greeting);
    assertEquals("Hello, world!", greeting);
  }

}
```

Im Vergleich mit dem Code für `TestHelloWorldInterpreted` lässt sich Folgendes feststellen:

- Die mit dem Codegenerator verwendete Sitzung verwendet `StronglyTypedRuleObjectFactory`. Der Name leitet sich daher ab, dass anstelle der dynamischen Instanzen von `RuleObject` generierte Java-Klassen genutzt werden.
- Das Laden des Regelwerks ist nicht erforderlich (daher wird keine Dienstprogrammmethode verwendet).
- Der Verweis auf die Regelklasse `HelloWorld` erfolgt über eine gleichnamige Java-Schnittstelle. Jeder Schreibfehler im Namen wird vom Java-Compiler erkannt.
- Der Verweis auf das Regelattribut `greeting` erfolgt analog über eine gleichnamige Java-Methode für die Schnittstelle.

- Eine Umsetzung des Rückgabetyps ist nicht erforderlich, da die generierte Methode `greeting` den korrekten Typ (`String`) zurückgibt.

Warnung: Der generierte Code ist nur für die Verwendung in Testumgebungen gedacht, bei denen es vor allem darauf ankommt, Änderungen am Code erneut zu kompilieren.

Der generierte Code ist *nicht* maschinenübergreifend portierbar, da er absolute Pfade für die Regelwerke auf der lokalen Maschine enthält.

Insbesondere darf der generierte Code *nicht* in einer Produktionsumgebung eingesetzt werden, in der sich Regelwerke dynamisch ändern können.

Code generieren

Führen Sie zur Ausführung des Codegenerators den folgenden Befehl aus:

```
build creole.generate.test.classes
```

Auf dem Ziel wird auch der CER-Regelwerkvalidierer für Ihre Regelwerke ausgeführt. Falls Fehler vorliegen, meldet der CER-Regelwerkvalidierer die Fehler und stoppt die Verarbeitung. Sind keine Fehler vorhanden, gibt der CER-Generator generierte Java-Klassen und -Schnittstellen für Ihre CER-Regelwerke und -Regelklassen aus.

Tipp: Der CER-Regelwerkvalidierer meldet außerdem alle Warnungen über nicht kritische Probleme in Ihren Regelwerken. Diese Warnungen verhindern weder Ausführung noch Test der Regeln, sollten jedoch berücksichtigt werden, um ein optimales Regelwerk zu gewährleisten.

Der CER-Codegenerator legt seine Ausgabe im Verzeichnis `EJBServer/build/svr/creole.gen/source` ab.

Nachfolgend finden Sie ein Beispiel der generierten Java-Schnittstelle für die Regelklasse `HelloWorld`:

```
/*
 * Generated by Curam CREOLE Code Generator
 * Generator Copyright 2008-2010 Curam Software Ltd.
 */
package curam.creole.ruleclass.HelloWorldRuleSet.impl;
/**
 * Code-generated interface for tests.
 * <p/>
 * Clients must not implement this interface.
 */
public interface HelloWorld extends
 curam.creole.execution.RuleObject {
  /**
   * Code-generated accessor for tests.
   * @return container for the greeting attribute value
   */
  public curam.creole.execution.AttributeValue<? extends
 java.lang.String> greeting();
}
```

Tipp: Sie sollten Ihre Testklassen neu generieren, wenn Sie Strukturänderungen an den Regelwerken im Dateisystem vornehmen, also beispielsweise Folgendes ausführen:

- Neues Regelwerk erstellen oder vorhandenes Regelwerk entfernen

- Neue Regelklasse zu einem Regelwerk hinzufügen oder vorhandene Regelklasse aus einem Regelwerk entfernen
- Neues Regelattribut zu einer Regelklasse hinzufügen oder vorhandenes Regelattribut aus einer Regelklasse entfernen
- Wert von "extends" für eine vorhandene Regelklasse ändern
- Datentyp eines Attributs ändern

Sie müsssen die Testklassen *nicht* neu generieren, falls Ihre Änderungen auf die *Implementierung* eines Regelattributs (also auf seine Ableitungsausdrücke) beschränkt sind. Die Ableitungen werden aus dem Regelwerk zur Laufzeit stets dynamisch verarbeitet und sind in den generierten Testklassen nicht vorhanden.

Tool für Regelwerkabdeckung

CER enthält ein Tool, mit dem diejenigen Teile eines Regelwerks gemeldet werden, die während der Laufzeit "abgedeckt" werden.

Die Abdeckungsstatistik kann für jede Verarbeitung gemeldet werden, die Werte aus CER anfordert. Beispiele:

- Aktive Onlineanwendung
- Ausführungen von JUnit-Tests

Zur Erfassung der Abdeckungsdaten legen Sie für die Umgebungseigenschaft `curam.creole.coverage.logfile` (in der Datei `Bootstrap.properties`) die Position einer Datei fest. Während der Ausführung von Regeln werden Zeilen mit Informationen zur Abdeckung an die Datei angehängt, sobald CER-Ausdrücke ausgewertet werden.

Tipp: Um die Abdeckungsdaten zu löschen, müssen Sie lediglich die in der Einstellung `curam.creole.coverage.logfile` angegebene Datei löschen.

Mit der Zeit kann die Datei für die Abdeckungsdaten relativ groß werden. Sie sollten daher die Erfassung der Abdeckungsdaten inaktivieren, wenn sie nicht erforderlich ist. Hierzu entfernen Sie die Einstellung für `curam.creole.coverage.logfile` oder setzen Sie auf Kommentar.

Führen Sie das folgende Ziel aus, um einen Abdeckungsbericht zu erstellen:

`build creole.report.coverage -Dfile.coverage.log=` *dateiposition*

In die Datei `.../EJBServer/build/svr/creole.gen/coverage/index.html` wird daraufhin ein einfacher und zur Detailanalyse geeigneter Bericht geschrieben, für den die folgende Farbcodierung gilt:

- Grün = Abgedeckt
- Gelb = Teilweise abgedeckt
- Rot = Nicht abgedeckt

Regelattribute mit einer Ableitung von <specified> sind absichtlich aus dem Bericht ausgeschlossen. Nachfolgend ein Beispiel für den Bericht:

CREOLE Coverage Report

Generated: 23-Mar-2011 16:33:07

Coverage for Rule Set: SimpleTestProductEligibilityEntitlementRuleSet

Rule Class Name	Rule Attribute Name	Rule Expressions	Fully Covered		Partially Covered		Not Covered	
Rule Set Summary		**623**	**231**	**37.08%**	**1**	**0.16%**	**391**	**62.76%**
AgeRangeCalculator		*47*	*45*	*95.74%*	*0*	*0.00%*	*2*	*4.26%*
	description	2	0	0.00%	0	0.00%	2	100.00%
	homeHelpAgeTimeline	5	5	100.00%	0	0.00%	0	0.00%
	isAliveTimeline	10	10	100.00%	0	0.00%	0	0.00%
	isHomeHelpAgeTimeline	8	8	100.00%	0	0.00%	0	0.00%
	isInAgeRangeTimeline	10	10	100.00%	0	0.00%	0	0.00%
	maximumAge	1	1	100.00%	0	0.00%	0	0.00%
	maximumAgeTimeline	5	5	100.00%	0	0.00%	0	0.00%
	minimumAge	1	1	100.00%	0	0.00%	0	0.00%
	minimumAgeTimeline	5	5	100.00%	0	0.00%	0	0.00%
	personCalculator	0	0		0		0	
CREOLEBonus		*0*	*0*		*0*		*0*	
	amount	0	0		0		0	
	evidenceID	0	0		0		0	
	type	0	0		0		0	
CREOLEBonusCalculator		*5*	*3*	*60.00%*	*0*	*0.00%*	*2*	*40.00%*

Abbildung 1. Beispiel für Abdeckungsbericht

Bitte beachten Sie, dass Regelwerke und -klassen, die in andere Regelwerke (mit dem Mechanismus <Include>) eingeschlossen sind, grundsätzlich ein Teil der Quelle für das äußere einschließende Regelwerk werden. Dies sollte bei der Analyse von Abdeckungsberichten stets berücksichtigt werden.

CER-Veröffentlichungsbereich

Die Cúram-Verwaltungsanwendung enthält Anzeigen, in denen CER-Regelwerke aufgelistet sind.

Ausgehend von diesen Anzeigen können Sie Folgendes ausführen:

- Vorhandenes Regelwerk anzeigen (im CER-Editor) und Versionsverläufe für jedes beliebige CER-Regelwerk anzeigen
- Vorhandenes Regelwerk öffnen und Änderungen vornehmen (im CER-Editor)
- Neues Regelwerk erstellen (und zum Hinzufügen von Regeln im CER-Editor öffnen)
- Vorhandenes Regelwerk entfernen

Seit Cúram Version 6 werden Änderungen an CER-Regelwerken *nicht* sofort wirksam, sondern stattdessen bis zu ihrer Veröffentlichung im Veröffentlichungsbereich gespeichert.

Sie können in diesem Bereich viele Änderungen an CER-Regelwerken kumulieren. Es kann sogar sein, dass Sie Änderungen an vielen Regelwerken kumulieren *müssen*, falls die vorgenommene Änderung mehr als ein Regelwerk betrifft.

Die anstehenden Änderungen können Sie jederzeit validieren lassen. Sobald Sie mit den Änderungen zufrieden sind, können Sie sie veröffentlichen. Das System stellt mit einer erneuten Validierung fest, ob die Regelwerke gültig sind, und lässt bei Gültigkeit die Fortsetzung der Veröffentlichung zu.

Die Veröffentlichung der Änderungen an CER-Regelwerken erfolgt durch eine zurückgestellte Verarbeitung, da vorhandene CER-Regelobjekte gemäß den Änderungen an CER-Regelklassen aktualisiert und/oder Neuberechnungen für Attribute, deren Ableitungen geändert wurden, in die Warteschlange gestellt werden müssen.

Schema- und Kataloggenerierung

Das Schema für CER-Regelwerke wird dynamisch assembliert, damit das festgelegte Schema für CER-Regelwerke, -Regelklassen und -Regelattribute und Beiträge zu CER-Ausdrücken und -Anmerkungen nach Anwendungskomponente berücksichtigt werden.

Das dynamisch assemblierte Schema befindet sich normalerweise im Hauptspeicher, wenn CER die Validierung verarbeitet. Manchmal kann es jedoch hilfreich sein, wenn dieses Schema (und ein auf das Schema verweisender Katalog) im Dateisystem vorhanden ist.

Sie können die CER-Schemadatei (`EJBServer/build/svr/creole.gen/schema/RuleSet.xsd`) generieren, indem Sie den folgenden Befehl ausführen:

```
build creole.generate.schema
```

Mit dem folgenden Befehl können Sie einen Katalog generieren (`EJBServer/build/svr/creole.gen/catalog/CREOLECatalog.xml`), der auf die CER-Schemadatei verweist:

```
build creole.generate.catalog
```

RuleDoc

Das RuleDoc ist eine Regeldokumentation, die Sie automatisch aus Ihren Cúram Express Rules-Regelwerken und -Regelklassen (CER) generieren können. CER stellt ein Tool für die Generierung des RuleDoc bereit.

Das RuleDoc kann Sie bei den folgenden Aufgaben unterstützen:

- Diskussion über das Verhalten des CER-Regelwerks mit einer nicht technischen Zielgruppe
- Darstellung der Abhängigkeiten zwischen den Regelattributen, insbesondere bei zunehmender Komplexität der Regelwerke
- Ermittlung des Einflusses von vorgenommenen Änderungen auf die Ableitung eines Regelattributs

Einfaches Beispiel

Die folgenden XML-Angaben stellen ein einfaches Regelwerk für "Hello, world" dar:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="HelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">
```

```
    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Das generierte RuleDoc für das obige Regelwerk, in dem die einzige Regelklasse aufgelistet ist, sieht wie folgt aus:

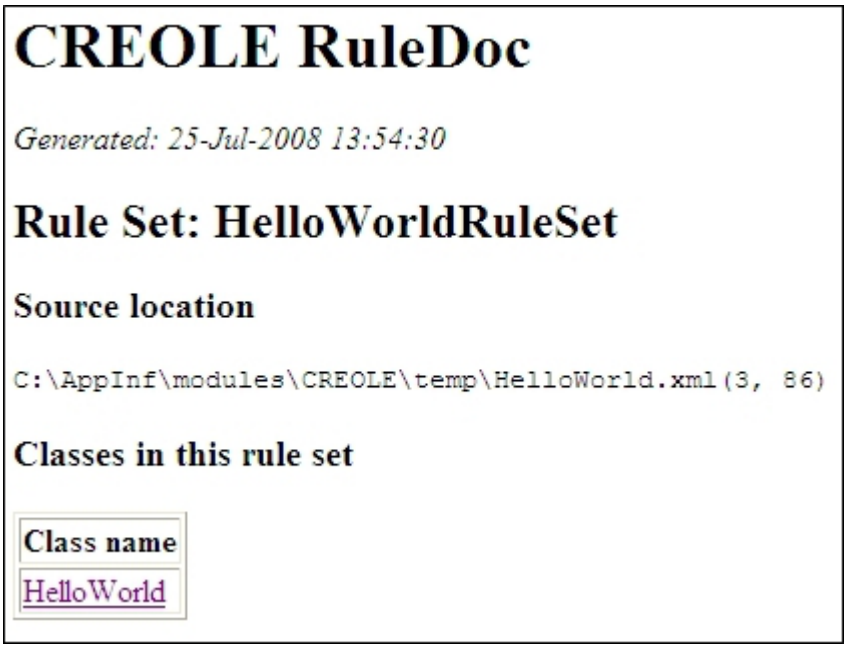

Abbildung 2. RuleDoc für `HelloWorldRuleSet`

Wenn Sie auf die Regelklasse `HelloWorld` klicken, wird das zugehörige RuleDoc angezeigt:

Abbildung 3. RuleDoc für Regelklasse `HelloWorld`

Nach dem Klicken auf das Attribut `greeting` wird dessen Ableitung angezeigt:

Abbildung 4. RuleDoc für Regelattribut `greeting`

Komplexeres Beispiel

Bei komplexeren Regelwerken unterstützt Sie das RuleDoc bei den folgenden Aktionen:

- In den Abhängigkeiten zwischen den Regelklassen des Regelwerks navigieren
- Ableitungsberechnung der einzelnen Regelattribute nachvollziehen
- Abhängigkeit weiterer Regelattribute von einem bestimmten Regelattribut ermitteln

Die folgenden XML-Angaben stellen ein komplexeres Regelwerk für eine Berechnungsfunktion für das Jahr des Renteneintritts dar:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="RetirementYearRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="RetirementYear">

    <Attribute name="yearOfBirth">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <Number value="1970"/>
      </derivation>
    </Attribute>

    <Attribute name="ageAtRetirement">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <Number value="65"/>
      </derivation>
    </Attribute>

    <Attribute name="yearOfRetirement">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <reference attribute="yearOfBirth"/>
          <reference attribute="ageAtRetirement"/>
        </arithmetic>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Das generierte RuleDoc für das obige Regelwerk sieht wie folgt aus:

Abbildung 5. RuleDoc mit Ableitung und Verwendung

Aus diesem Beispiel wird Folgendes ersichtlich:

- Ableitung des Regelattributs `yearOfRetirement` (mit Verknüpfungen zu den Attributen `yearOfBirth` und `ageAtRetirement`, von denen das Attribut abhängig ist)
- Regelattribut `yearOfBirth` mit einer Verknüpfung zum Regelattribut `yearOfRetirement`, das direkt von ihm abhängig ist

RuleDoc generieren

Führen Sie zur Ausführung des CER-RuleDoc-Generators den folgenden Befehl aus:

```
build creole.generate.ruledoc
```

Auf dem Ziel wird auch der CER-Regelwerkvalidierer für Ihre Regelwerke ausgeführt. Falls Fehler vorliegen, meldet der CER-Regelwerkvalidierer die Fehler und

stoppt die Verarbeitung. Sind keine Fehler vorhanden, gibt der CER-Generator das RuleDoc für die Regelwerke und Regelklassen aus.

Tipp: Der CER-Regelwerkvalidierer meldet außerdem alle Warnungen über nicht kritische Probleme in Ihren Regelwerken. Diese Warnungen verhindern weder Ausführung noch Test der Regeln, sollten jedoch berücksichtigt werden, um ein optimales Regelwerk zu gewährleisten.

Der CER-RuleDoc-Generator legt seine Ausgabe im Verzeichnis `EJBServer/build/svr/creole.gen/ruledoc` ab.

SessionDoc

Während einer Sitzung können Sie eine HTML-Dokumentation namens SessionDoc ausgeben. Das SessionDoc enthält einen Datensatz für alle Regelobjekte, die während der Sitzung erstellt wurden, und kann im Rahmen von Tests wertvolle Dienste für das Debugging leisten.

Es kann hilfreich sein, den JUnit-Ankerpunkt `tearDown` zu verwenden, um durchzusetzen, dass das SessionDoc für alle Testmethoden in den Testklassen ausgegeben wird:

```
@Override
          protected void tearDown() throws Exception {
          /*
          * Write out SessionDoc, to a directory named after the test
          * method.
          */
          final File sessionDocOutputDirectory =
          new File("./gen/sessiondoc/" + this.getName());
          sessionDoc.write(sessionDocOutputDirectory);

          super.tearDown();
          }
```

Das folgende Beispiel zeit die SessionDoc-Hauptseite für einen Test `testSelfMade-MillionaireScenario`:

CREOLE Session

Generated: 13-Jul-2012 12:08:08

Session Type

- Recalculation strategy: curam.creole.execution.session.RecalculationsProhibited
- Data storage: curam.creole.storage.inmemory.InMemoryDataStorage
- Rule object factory: curam.creole.execution.session.StronglyTypedRuleObjectFactory

Options

- "Used by" links included: true

Rule Objects (by Rule Set)

- FlexibleRetirementYearRuleSet

Abbildung 6. SessionDoc für Test `testSelfMadeMillionaireScenario`

Diese Seite enthält einige wichtige Details:

- Datum und Uhrzeit für die Erstellung des SessionDoc
- Strategien für die Erstellung der Sitzung
- Liste der Regelwerke, deren Regelobjekte im SessionDoc erfasst wurden
- Angabe, ob Links des Typs "Used by" enthalten sind

Wenn Sie auf den Link für das einzige Regelwerk `FlexibleRetirementYearRuleSet` klicken, werden dessen Regelobjekte angezeigt:

FlexibleRetirementYearRuleSet

Generated: 13-Jul-2012 12:08:08

External rule objects

Details	Type	Description	Action
details	FlexibleRetirementYearRuleSet.FlexibleRetirementYear	Undescribed instance of rule class 'FlexibleRetirementYear', id '1'	Created during this session

Internal rule objects

Details	Type	Description	Action

Abbildung 7. Regelobjekte für das Regelwerk `FlexibleRetirementYearRuleSet`

Auf dieser Seite ist Folgendes angegeben:

- "Externe" Regelobjekte (für Bootstrap), die während dieser Sitzung durch den Client-Code erstellt wurden (im Test wurde nur ein Objekt erstellt)

- "Interne" Regelobjekte, die während dieser Sitzung durch Regeln erstellt wurden (bei diesem Test nicht berechnet)

Wenn Sie auf den Link "details" für das einzige Regelobjekt `FlexibleRetirementYear` klicken, wird dessen SessionDoc angezeigt:

Created externally

Action during this session

Created during this session

Attributes

Name	Declared type	State	Value	Derivation	Depends on	Used by
ageAtRetirement	Number	CALCULATED	50	• If • retirementCause == / Then • "Lottery winner" / • 35 • "Self-made millionaire" / • 50 Otherwise / • 65	• retirementCause	• yearOfRetirement
description	Message	CALCULATED	Undescribed instance of rule class 'FlexibleRetirementYear', id '1'	• Default rule object description.	*None*	*None*
retirementCause	String	SPECIFIED	Self-made millionaire	• Specified externally.	*None*	• ageAtRetirement
yearOfBirth	Number	SPECIFIED	1980	• Specified externally.	*None*	• yearOfRetirement
yearOfRetirement	Number	CALCULATED	2030	• Arithmetic: ○ yearOfBirth ○ + ○ ageAtRetirement	• ageAtRetirement • yearOfBirth	*None*

Abbildung 8. SessionDoc für Regelobjekt `FlexibleRetirementYear`

Am (hier nicht dargestellten) Beginn des SessionDoc sind Details für das Regelobjekt zusammengefasst. Anschließend ist jedes Regelattribut für das Regelobjekt mit den folgenden Details aufgeführt:

- **Name**

  Der Name des Regelattributs.
- **Declared type**

  Der Typ des Regelattributs, der im Regelwerk deklariert ist. Der tatsächliche Laufzeitwert kann aus einem Subtyp dieses deklarierten Typs stammen.
- **State**

  Der Status des Wertes. Mögliche Angaben:
  - **CALCULATED**

    Der Wert wurde durch Regeln berechnet.
  - **SPECIFIED**

    Der Wert wurde als Ersatz für eine definierte Berechnung explizit durch Client-Code angegeben oder im Rahmen eines Ausdrucks `create` initialisiert.

    Anmerkung: Vor Cúram Version 6 wurde der Status INITIALIZED verwendet. Ab Cúram Version 6 wird stattdessen der Status SPECIFIED verwendet.

  - **NOT_YET_CALCULATED**

    Der Wert ist nicht explizit angegeben oder wird nicht während der Regelausführung berechnet (weil der Wert noch nie von anderen Berechnungen oder Tests angefordert wurde).
  - **ERROR**

    Während der Berechnung des Wertes trat ein Fehler auf (Details und den Berechnungsstack des Fehlers können Sie den Anwendungsprotokollen oder der Konsolenausgabe entnehmen).
- **Value**

  Eine Anzeigedarstellung des Wertes. Falls der Wert noch nie berechnet wurde (NOT_YET_CALCULATED) oder fehlerhaft ist (ERROR), wird "?" angezeigt. Handelt es sich bei dem Wert um ein Regelobjekt, wird der Wert als navigationsfähiger Hyperlink angezeigt, damit Sie die Details dieses Regelobjekts anzeigen können.
- **Derivation**

  Die RuleDoc-Ableitung des Attributs (ohne Links). Weitere Informationen zum RuleDoc finden Sie im Abschnitt „RuleDoc" auf Seite 20.
- **Depends on**

  Hier sind die Links zu den Attributen angegeben, die zur Berechnung dieses Wertes verwendet wurden.
- **Used by**

  (Optional) Hier sind die Links zu den Attributen angegeben, die diesen Wert bei der Berechnung ihres eigenen Wertes verwendet haben.

Tipp: Durch die Implementierung eines Regelattributs `description` für jede Klasse können Sie erreichen, dass Ihr SessionDoc verständlicher ist. Weitere Details enthält der Abschnitt „Regelattribut `description`" auf Seite 145.

Falls Sie ein SessionDoc für eine umfangreiche Datenbank ausführen, kann es sinnvoll sein, die Ausgabe der Links unter "Used by" zu unterdrücken, da der Einschluss solcher Links dazu führen kann, dass viele Regelobjekte im SessionDoc ausgegeben werden. Um die Ausgabe der Links unter "Used by" zu unterdrücken, verwenden Sie `curam.creole.execution.session.SessionDoc.write(File, boolean)` und übergeben hierbei *false* als zweiten Parameter.

Für Regelobjekte, die in den Datenbanktabellen von CER gespeichert sind, können Sie das SessionDoc erstellen, indem Sie die Klasse `curam.creole.util.DumpOutRuleObjects` mit einem einzigen Argument ausführen, für das Sie den Namen eines Verzeichnisses angeben, in dem das SessionDoc erstellt werden soll. Das Dienstprogramm "DumpOutRuleObjects" ruft alle Regelobjekte aus den Datenbanktabellen von CER ab. Die Aktion für jedes externe Regelobjekt lautet folglich "retrieved" (= abgerufen). Alle internen Regelobjekte werden erstellt (da sie nicht gespeichert sind). Für sie ist daher die Aktion "created" (= erstellt) angegeben.

Tipp: Das Dienstprogramm "DumpOutRuleObjects" kann eine nützliche Methode zum Anzeigen der Regelobjekte sein, die in den Datenbanktabellen von CER gespeichert sind. Außerdem kann es für das Debugging von Nutzen sein, sobald Sie den Punkt erreicht haben, an dem Sie CER-Regeln in Ihre Onlineanwendung integrieren wollen.

Durch das Anzeigen der Regelobjekte können Sie die Werte der berechneten Attribute für Regelobjekte einsehen und in einer technische Ansicht feststellen, wie die einzelnen Berechnungsergebnisse erzielt wurden.

Nicht verwendete Regelattribute

CER unterstützt das Melden von Regelattributen, auf die nicht von anderen Berechnungen im Regelwerk verwiesen sind und die daher potenziell entfernt werden können.

Führen Sie zur Erstellung des CER-Berichts über nicht verwendete Attribute den folgenden Befehl aus:

```
build creole.report.unused.attributes
```

CER validiert die Regelwerke und meldet alle nicht verwendeten Regelattribute an die Konsole.

Warnung: Es ist durchaus möglich, dass es sich bei einem Regelattribut um ein "übergeordnetes" Regelattribut handelt, auf das nur durch Client-Code verwiesen wird. Solche Attribute werden in diesem Bericht möglicherweise als nicht verwendet gemeldet. Sie sollten jedoch nur dann scheinbar nicht verwendete Regelattribute aus Ihrem Regelwerk entfernen, wenn Sie sicher sind, dass kein Client-Code oder Test von diesen Attributen abhängig ist.

CER-Konsolidierungskomponente für Regelwerke

In Cúram Version 5.2 ermöglichte CER das Aufteilen eines Regelwerks in kleinere Dateien, was die gleichzeitige Entwicklung von Regelwerken in verschiedenen Dokumenten unterstützte.

Details über das Aufteilen eines Regelwerks enthält der Abschnitt „Anweisung `Include`" auf Seite 166.

Vor dem Laden des CER-Regelwerks für die Daten wird es von den Erstellungsscripts der Anwendung (insbesondere dem Ziel **`build creole.consolidate.rulesets`**) automatisch in einer einzigen Regelwerkdatei konsolidiert.

Anmerkung: Die CER-Konsolidierungskomponente für Regelwerke blendet nur Anweisungen `Include` aus, die eine Position des Typs `RelativePath` enthalten.

Alle anderen Typen von Anweisungen `Include` bleiben in der konsolidierten Ausgabe absichtlich unverändert erhalten.

Beispiel

Nachfolgend finden Sie ein CER-Regelwerk, das ein weiteres Regelwerk enthält:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Include"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- This rule class is defined directly in this rule set -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
```

```
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Include a rule set defined in another file.

       When assembled into a single rule set, the
       names of all the rule classes must be unique. -->
  <Include>
    <RelativePath value="./HelloWorld.xml"/>
  </Include>

</RuleSet>
```

Im Folgenden ist dasselbe Regelwerk nach der Konsolidierung dargestellt:

```
<?xml version="1.0" encoding="UTF-8"?><RuleSet
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 name="Example_Include" xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- This rule class is defined directly in this rule set -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Include a rule set defined in another file.

       When assembled into a single rule set, the
       names of all the rule classes must be unique. -->

<!--Start inclusion of ./HelloWorld.xml-->
<Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>
<!--End inclusion of ./HelloWorld.xml-->


</RuleSet>
```

Verarbeitung von Daten

Eine Beschreibung der Verarbeitung von Daten in CER. CER führt Berechnungen für Daten aus. Die Ergebnisse dieser Berechnungen stellen selbst ebenfalls Daten dar. Seit Cúram Version 6 unterstützt CER die Speicherung von Regelobjektdaten in der Datenbank und in Momentaufnahmen.

CER-Sitzungen

Bei CER werden die folgenden Hauptdatenelemente verwendet:

- **Regelobjekte**

  Ein Regelobjekt ist eine Instanz einer Regelklasse aus einem CER-Regelwerk, beispielsweise das Regelobjekt für die Person "Thomas Schmidt".

- **Attributwerte**

  Ein Attributwert ist der Wert eines CER-Regelattributs für ein bestimmtes Regelobjekt, beispielsweise das Geburtsdatum von Thomas Schmidt.

Alle Interaktionen mit CER-Regelobjekten und -Attributwerten finden innerhalb einer CER-Sitzung statt. Zu diesen Interaktionen gehören das Erstellen, Abrufen und/oder Entfernen von CER-Regelobjekten sowie jede Berechnung oder Neuberechnung von Attributen für Regelobjekte.

Jede CER-Sitzung wird unter Verwendung der Klasse `curam.creole.execution.session.Session_Factory` erstellt.

Bei der Erstellung einer CER-Sitzung muss Folgendes angegeben werden:

- **Neuberechnungsstrategie**

  Die Strategie für die Verarbeitung einer Anforderung zur Neuberechnung eines CER-Attributwerts in einer CER-Sitzung.

  Anmerkung: Die CER-Funktionalität für die direkte Ausführung von Neuberechnungen wird nun durch den Abhängigkeitsmanager abgelöst (siehe „Abhängigkeitsmanager" auf Seite 81). Die Schnittstelle und die Implementierungen für die CER-Neuberechnungsstrategie werden nur aus Gründen der Abwärtskompatibilität bereitgestellt.

  .

  Die Strategie für die Verarbeitung einer Anforderung zur Neuberechnung eines CER-Attributwerts. Beispielsweise kann angegeben werden, ob die Neuberechnung sofort erfolgen, in eine andere Transaktion zurückgestellt oder gänzlich unzulässig sein soll.

- **Datenspeicher**

  Der Speichermechanismus, der für persistente Regelobjekte verwendet werden soll; beispielsweise lediglich hauptspeicherintern (was bedeutet, dass die Daten nach Ablauf der Sitzungsgültigkeit nicht mehr vorhanden sind), Datenbankspeicher oder Momentaufnahmespeicher.

  Anmerkung:

  Seit Cúram Version 6 bietet CER eine Auswahl von Datenspeicherimplementierungen. Diese Datenspeicherimplementierungen wirken sich darauf aus, ob Regelobjekte, die in einer Transaktion erstellt oder geändert wurden, in anderen Transaktionen abgerufen oder geändert werden können.

  Besonders wichtig ist in diesem Zusammenhang, dass die Auswahl des Datenspeichers *keinen* Einfluss auf die Semantik der Regelausdrücke hat. Dies bedeutet, dass Sie einen einfachen (hauptspeicherinternen) Datenspeicher für Ihre JUnit-Tests verwenden können (damit eine große Anzahl von JUnit-Tests schnell ausgeführt werden kann) und dass Sie einen persistenten Datenspeicher (Datenbank) für Ihre Produktionslogik einsetzen können (damit Regelobjekte transaktionsübergreifend persistent sind), *ohne* dass es Unterschiede in den zugrunde liegenden Berechnungen gibt.

Die Implementierungen von Datenspeicher müssen wiederum eine zu verwendende Regelobjektfactory angeben. Diese Factory steuert, ob Regelobjekte streng typisiert oder lediglich interpretiert erstellt werden.

Die Anwendung umfasst die folgenden Implementierungen:

- Neuberechnungsstrategie:
  - **curam.creole.execution.session.RecalculationsProhibited**

    Löst einen Fehler aus, wenn in der CER-Sitzung versucht wird, eine Neuberechnung auszuführen.
  - **curam.core.sl.infrastructure.propagator.impl.ImmediateRecalculationStrategy**

    Führt Neuberechnungen sofort (synchron in der derselben Datenbanktransaktion) aus.
  - **curam.core.sl.infrastructure.propagator.impl.DeferredRecalculationStrategy**

    Stellt Neuberechnungen (für gespeicherte Attributwerte) in eine andere Datenbanktransaktion zurück.

    Stellt Neuberechnungen (für gespeicherte Attributwerte) in eine andere Datenbanktransaktion zurück.
- Datenspeicher:
  - **curam.creole.storage.inmemory.InMemoryDataStorage**

    Behält Regelobjekte lediglich im Hauptspeicher bei. Regelobjekte in diesem Datenspeicher sind nur solange verfügbar, wie der Datenspeicher gültig ist (normalweise nur für eine einzige Datenbanktransaktion).
  - **curam.creole.storage.database.DatabaseDataStorage**

    Anmerkung: Als Optimierung werden nur externe Regelobjekte und deren Attributwerte aus Daten in der Cúram-Datenbank abgerufen. Interne Regelobjekte und deren Attribute können naturgemäß zu einem späteren Zeitpunkt zuverlässig berechnet werden, während externe Regelobjekte normalerweise Daten aus externen Quellen enthalten und folglich nicht neu berechnet werden können.

    Ruft externe Regelobjekte aus einer der folgenden Quellen ab:
  - **curam.creole.execution.session.RuleObjectsSnapshot.SnapshotDataStorage**

    Erstellt ein XML-Dokument mit Details einer Gruppe von Regelobjekten, die an den Abhängigkeiten für eine oder mehrere Attributberechnungen beteiligt sind. Stellt ein Überprüfungsprotokoll der Daten bereit, die letztlich in dieser Berechnung verwendet werden. Das XML-Dokument kann normalweise in einer Datenbanktabelle gespeichert werden, damit die Momentaufnahme von Regelobjekten durch eine nachfolgende Datenbanktransaktion abgerufen (jedoch nicht geändert) werden kann.
    - Entweder ein Regelobjektkonverter, der beim Datenbankdatenspeicher registriert ist. Jeder Regelobjektkonverter benennt die von ihm verarbeiteten Regelklassen. Bei seinem Aufruf liest der Regelobjektkonverter die zugrunde liegenden Geschäftstabellen, um die entsprechenden Daten abzurufen, die Regelobjekte im Hauptspeicher zu füllen und sie anschließend an den Datenbankdatenspeicher zurückzugeben.
    - Oder eigene Datenbanktabellen von CER für die Speicherung von Regelobjekten, falls kein Regelobjektkonverter für die Verarbeitung der angeforderten Regelklasse registriert ist.

    Anmerkung: Bitte beachten Sie, dass jeder Regelobjektkonverter Grenzwerte für seine Unterstützung von Ausdrücken "readall" (siehe „readall" auf Seite 224 und „readall" auf Seite 224) durchsetzen kann. Beispielsweise unterstüt-

zen einige Regelobjektkonverter möglicherweise nicht die Ausführung eines Ausdrucks des Typs "readall" (siehe „readall" auf Seite 224) ohne verschachtelte Angabe von `match` oder geben Einschränkungen für die Regelattribute vor, die im Wert für `retrievedattribute` des Ausdrucks `match` angegeben werden können. Verstöße gegen die Einschränkungen des Regelobjektkonverters führen dazu, dass zur Laufzeit eine Ausnahmebedingung ausgelöst wird. Sie sollten sicherstellen, dass Ihre Regelwerktests Logik enthalten, mit denen die Regelobjektkonverter (d. h. die für den Datenbankdatenspeicher ausgeführten Konverter) aufgerufen werden - im Gegensatz zur Mehrzahl der Regellogiktests, die den hauptspeicherinternen Datenspeicher nutzen und daher keine Regelobjektkonverter aufrufen. Wissenswertes über die Einschränkungen, die eine Implementierung eines Regelobjektkonverters für die Unterstützung von Ausdrücken "readall" (siehe „readall" auf Seite 224) vorgibt, finden Sie in der Dokumentation für die jeweilige Implementierung.

Externe Regelobjekte sind für den Abruf und die Bearbeitung durch nachfolgende Datenbanktransaktionen verfügbar.

– **curam.creole.storage.hybrid.HybridDataStorage**

Kombiniert Verhaltensaspekte der Implementierungen "InMemoryDataStorage" und "DatabaseDataStorage". *Ist für die interne Verwendung durch Cúram reserviert.*

- Regelobjektfactory:
  – **curam.creole.execution.session.StronglyTypedRuleObjectFactory**

  Erstellt Regelobjekte als Instanzen von Java-Klassen (bzw. ruft sie ab), die durch den CER-Testcodegenerator generiert wurden (siehe „CER-Testcodegenerator" auf Seite 15).

  Anmerkung: Darf nicht in Code für die Produktionverwendet werden.

  – **curam.creole.execution.session.InterpretedRuleObjectFactory**

  Verwendet ein vollständig interpretiertes (und somit dynamisches) Verfahren, um Regelobjekte zu erstellen und abzurufen (siehe „Regelwerkinterpreter" auf Seite 12).

Wichtig: Generell sollten nicht mehrere CER-Sitzungen in einer einzigen Datenbanktransaktion verwendet werden. Die im Hauptspeicher befindlichen Kopien der Regelobjekte in einer CER-Sitzung sind unabhängig von den im Hauptspeicher befindlichen Kopien von Regelobjekten in allen anderen CER-Sitzungen.

Das Verhalten kann nicht garantiert werden, wenn mehrere CER-Sitzungen (in derselben Transaktion) versuchen, dasselbe Regelobjekt aus der Datenbank abzurufen oder abzufragen.

- **Komponententests**

  Verwenden Sie (zur schnellen Ausführung) eine Implementierung des Typs "InMemoryDataStorage" mit "StronglyTypedRuleObjectFactory" (damit generierte Java-Klassen in Tests verwendet werden können) und mit "RecalculationsProhibited" (damit Daten nicht versehentlich auf halbem Weg durch Tests geändert werden).

- **Produktionslogik mit dynamischen Regelwerken**

  Verwenden Sie eine Implementierung des Typs "DatabaseDataStorage" (damit Regelobjekte transaktionsübergreifend verfügbar sind) mit "InterpretedRuleObjectFactory" (damit die Regelwerke vollständig dynamisch sind) und mit "RecalculationsProhibited" und verwenden Sie die vom Abhängigkeitsmanager bereit-

gestellten Funktionen (siehe „Abhängigkeitsmanager” auf Seite 81), um alle Anforderungen zur Neuberechnung von CER-Werten in einer neuen (und unabhängigen) CER-Sitzung auszuführen.

- **Momentaufnahmen**

  Verwenden Sie eine Implementierung des Typs "SnapshotDataStorage" (damit die Regelobjekte aus einem nicht änderbaren XML-Dokument gelesen werden) mit "InterpretedRuleObjectFactory" (damit Regelwerke vollständig dynamisch sind) und "RecalculationsProhibited" (Änderungen werden von Momentaufnahmen nicht unterstützt).

Externe und interne Regelobjekte

CER bietet verschiedene Verfahren für die Erstellung von Regelobjekten:

- **Extern**

  Dies sind Regelobjekte, die von CER-Clients außerhalb des Kontextes einer Berechnung erstellt werden. Externe Regelobjekte stellen tendenziell realistische oder behördliche Konzepte wie eine Person oder einen Fall dar.

- **Intern**

  Dies sind Regelobjekte, die durch CER-Regeln (oder innerhalb des von CER-Regeln aufgerufenen Java-Codes) erstellt wurden. Interne Regelobjekte dienen eher als "Berechnungsfunktionen" oder als abgeleitete Daten für einen Zwischenschritt in einer komplexen Berechnungskette.

Die Unterscheidung zwischen externen und internen Regelobjekte wird nachfolgend erläutert und ergibt sich aus den folgenden Aspekten:

- Kann ein Regelobjekt mit dem Ausdruck "readall" (siehe „`readall`” auf Seite 224) abgerufen werden?
- Kann ein Regelobjekt zum Abruf in nachfolgenden Transaktionen in der Datenbank gespeichert werden?

Externe Regelobjekte

CER lässt zu, dass Client-Code Fragen zu einem Regelobjekt stellt (CER führt dann Regeln aus, um die Antworten auf diese Fragen bereitzustellen).

Damit Client-Code eine Frage zu einem Regelobjekt stellen kann, muss dieses Regelobjekt sowohl dem Client-Code als auch CER bekannt sein. Insofern muss die CER-Sitzung mindestens über ein Ausgangsregelobjekt verfügen, das durch Client-Code erstellt oder abgerufen wurde. Dieser Client-Code kann Testcode oder aber Code sein, der CER bei einer Anwendung integriert.

Ein externes Regelobjekt ist der Ausgangspunkt für die Fragestellung durch den Client-Code. Die Antwort auf eine solche Frage kann jedoch durchaus ein Regelobjekt oder eine Liste von Regelobjekten bereitstellen, die entweder aus Regeln erstellt oder aus anderen externen Regelobjekten abgerufen wurden.

Wichtig: Sobald Berechnungen begonnen haben, verhindert die Strategie `RecalculationsProhibited` die Erstellung von weiteren Regelobjekten, die alle zuvor ausgeführten Berechnungen für "readall" (siehe „`readall`” auf Seite 224) ungültig machen würden.

Zur Vermeidung derartiger Fehler sollten Sie Ihren Client-Code bzw. Ihre Tests so strukturieren, dass die Erstellung aller Testregelobjekte *vor* allen etwaigen Berechnungen (also vor jeder Ausführung von `getValue`) stattfindet.

Beispiel: Das folgende Beispiel zeigt ein Regelwerk:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_externalRuleObjects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- These attributes must be specified at creation time -->
    <Initialization>
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

      <Attribute name="lastName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

    <Attribute name="incomes">
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <!-- Read all the rule objects of
              type "Income" -->
        <readall ruleclass="Income"/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Income">
    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Im obigen Regelwerk wird der Ausdruck "readall" (siehe „readall" auf Seite 224) verwendet, um alle Instanzen der Regelklasse `Income` abzurufen.

Zur Erstellung eines externen Regelobjekts müssen der Client-Code oder die Tests beim Erstellen des Regelobjekts die Sitzung angeben:

```
package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.RuleObject;
import curam.creole.execution.session.InterpretedRuleObjectFactory;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
```

```
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.parser.RuleSetXmlReader;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Income;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Income_Factory;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Person;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Person_Factory;
import curam.creole.ruleitem.RuleSet;
import curam.creole.storage.inmemory.InMemoryDataStorage;

/**
 * Tests external rule objects created directly by client code.
 */
public class TestCreateExternalRuleObjects extends TestCase {

  /**
   * Example showing the creation of external rule objects using
   * generated code.
   */
  public void testUsingGeneratedTestClasses() {

    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    /**
     * Note that the compiler enforces that the right type of
     * initialization arguments are provided.
     */
    final Person person =
        Person_Factory.getFactory().newInstance(session, "John",
            "Smith");
    CREOLETestHelper.assertEquals("John", person.firstName()
        .getValue());

    /**
     * These objects will be retrieved by the
     *
     * <readall ruleclass="Income"/>
     *
     * expression in the rule set.
     */
    final Income income1 =
        Income_Factory.getFactory().newInstance(session);
    income1.amount().specifyValue(123);

    final Income income2 =
        Income_Factory.getFactory().newInstance(session);
    income2.amount().specifyValue(345);

  }

  /**
   * Example showing the creation of external rule objects using
   * the CER rule set interpreter.
   */
  public void testUsingInterpreter() {

    /* read in the rule set */
    final RuleSet ruleSet = getRuleSet();

    /* start an interpreted session */
```

```
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new InterpretedRuleObjectFactory()));

    /**
     * Note that the compiler cannot enforce that the right type of
     * initialization arguments are provided - if these are wrong
     * CER will report a runtime error.
     */
    final RuleObject person =
        session.createRuleObject(ruleSet.findClass("Person"),
            "John", "Smith");
    CREOLETestHelper.assertEquals("John", person
        .getAttributeValue("firstName").getValue());

    /**
     * These objects will be retrieved by the
     *
     * <readall ruleclass="Income"/>
     *
     * expression in the rule set.
     */
    final RuleObject income1 =
        session.createRuleObject(ruleSet.findClass("Income"));
    income1.getAttributeValue("amount").specifyValue(123);

    final RuleObject income2 =
        session.createRuleObject(ruleSet.findClass("Income"));
    income2.getAttributeValue("amount").specifyValue(345);
  }

  /**
   * Reads the Example_externalRuleObjects from its XML source
   * file.
   */
  private RuleSet getRuleSet() {

    /* The relative path to the rule set source file */
    final String ruleSetRelativePath =
        "./rules/Example_externalRuleObjects.xml";

    /* read in the rule set source */
    final RuleSetXmlReader ruleSetXmlReader =
        new RuleSetXmlReader(ruleSetRelativePath);

    /* dump out any problems */
    ruleSetXmlReader.validationProblemCollection().printProblems(
        System.err);

    /* fail if there are errors in the rule set */
    assertTrue(!ruleSetXmlReader.validationProblemCollection()
        .containsErrors());

    /* return the rule set from the reader */
    return ruleSetXmlReader.ruleSet();
  }

}
```

Einsatzsituationen für externe Regelobjekte: Sie sollten externe Regelobjekte für Folgendes erstellen:

- Ausgangsregelobjekte oder übergeordnete Regelobjekte, die immer vorhanden sein müssen, damit der Client-Code sinnvolle Fragen stellen kann. Diese Regelobjekte sind normalerweise Singletons (also die einzige Instanz dieser speziellen Regelklasse während der Sitzung).
- Regelobjekte, die auf der Grundlage externer Daten erstellt werden (z. B. eine Person oder ein Fall).

Interne Regelobjekte

CER lässt zu, dass Regeln als Ergebnis oder als Nebenprodukt von Berechnungen neue Regelobjekte erstellen.

Zum Erstellen eines Regelobjekts verwenden Sie die Ausdruck "create" (siehe „create" auf Seite 191) und geben Sie die von der Regelklasse benötigten Initialisierungsargumentwerte und/oder zusätzliche Werte für das erstellte Regelobjekt an.

Wichtig: Regelobjekte, die mit dem Ausdruck "create" (siehe „create" auf Seite 191) erstellt wurden, können *nicht* mit Ausdrücken "readall" (siehe „readall" auf Seite 224) abgerufen werden, weil CER nicht garantieren kann, dass alle internen Regelobjekte erstellt worden sind oder erstellt werden (dies ist davon abhängig, ob ein Ausdruck "create" (siehe „create" auf Seite 191) im Ausführungspfad für eine Berechnung gefunden wird).

Beispiel: Das folgende Beispiel zeigt ein CER-Regelwerk, das den Ausdruck "create" (siehe „create" auf Seite 191) verwendet, um eine bedingte Erstellung von Regelobjekten aus Regeln vorzunehmen:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_internalRuleObjects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Uses <create> to get a new rule object.

         Other calculations (on this or other rule objects) can
         access this newly-created rule object by referring to
         this attribute, i.e.

         <reference attribute="minorAgeRangeTest"/> .

         The rule object created CANNOT be retrieved using a
         <readall> expression.
     -->
    <Attribute name="minorAgeRangeTest">
      <type>
        <ruleclass name="AgeRangeTest"/>
      </type>
      <derivation>
        <!-- Create an age-range test which checks whether this
             person is aged between 0-17 inclusive (i.e. is
             under 18 years).
          -->
```

```
      <create ruleclass="AgeRangeTest">
        <this/>
        <Number value="0"/>
        <Number value="17"/>
      </create>
    </derivation>

  </Attribute>

  <!-- Uses the age-range check to determine whether this person
       is a minor. -->
  <Attribute name="isMinor">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <reference attribute="isPersonInAgeRange">
        <reference attribute="minorAgeRangeTest"/>
      </reference>
    </derivation>
  </Attribute>

  <!-- Uses <create> to get a new rule object, within
       another expression.

       Because the new rule object is created "anonymously",
       it is not available to any other rule objects (but
       it will still show up as "created" in any SessionDoc.
  -->
  <Attribute name="isOfWorkingAge">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <!-- Create an age-range test which checks whether this
           person is legally permitted to work (i.e. is aged
           at least 16 and less than 65), and then check
           whether the test passes. -->
      <reference attribute="isPersonInAgeRange">
        <create ruleclass="AgeRangeTest">
          <this/>
          <Number value="16"/>
          <Number value="64"/>
        </create>
      </reference>
    </derivation>

  </Attribute>

</Class>

<!-- A generic test which checks whether the
     person's age lies within a specified (inclusive)
     range. -->
<Class name="AgeRangeTest">
  <Initialization>
    <Attribute name="person">
      <type>
        <ruleclass name="Person"/>
      </type>
    </Attribute>
    <Attribute name="minimumAge">
      <type>
        <javaclass name="Number"/>
      </type>
    </Attribute>
    <Attribute name="maximumAge">
```

```
        <type>
          <javaclass name="Number"/>
        </type>
      </Attribute>
    </Initialization>

    <Attribute name="isPersonInAgeRange">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <compare comparison=">=">
                <reference attribute="age">
                  <reference attribute="person"/>
                </reference>
                <reference attribute="minimumAge"/>
              </compare>
              <compare comparison="&lt;=">
                <reference attribute="age">
                  <reference attribute="person"/>
                </reference>
                <reference attribute="maximumAge"/>
              </compare>
            </members>
          </fixedlist>
        </all>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Anmerkung: Wie alle CER-Ausdrücke wird der Ausdruck "create" (siehe „create“ auf Seite 191) nur berechnet, wenn er angefordert wird. Beispielsweise wird das Regelobjekt *minorAgeRangeTest* nur dann erstellt, wenn sein Wert oder der Wert von *isMinor* durch den Client-Code oder durch eine andere Berechnung angefordert wird.

Im obigen Beispiel verwendet *isOfWorkingAge* das Verfahren für die "anonyme" Erstellung eines Regelobjekts. Hierzu wird die Erstellung des Regelobjekts in einen Ausdruck einschlossen, der auf ein Attribut im neu erstellten Regelobjekt verweist. Solche Regelobjekte sind nicht für andere Berechnungen verfügbar, werden jedoch in jedem generierten SessionDoc aufgeführt.

Ein anonymes Regelobjekt kann nützlich sein, wenn Sie in einem erstellten Regelobjekt auf Regelattribute zugreifen müssen, das erstellte Regelobjekt selbst jedoch nicht für andere Berechnungen zur Verfügung stehen muss.

Pool für interne Regelobjekte

Seit Cúram Version 6 behält CER einen "Pool" mit internen Regelobjekten bei, die während einer Sitzung erstellt wurden.

Der Pool der Sitzung wird immer dann abgefragt, wenn ein Ausdruck "create" (siehe „create“ auf Seite 191) ausgewertet wird. Falls bereits ein Regelobjekt mit der-

selben Initialisierung und/oder denselben angegebenen Parametern erstellt wurde, wird es aus dem Pool wiederverwendet, statt ein neues Regelobjekt zu erstellen.

Diese Strategie der Poolnutzung verbessert die Effizienz in Situationen, bei denen viele Anweisungen "create" (siehe „create" auf Seite 191) versuchen, identische Regelobjekte zu erstellen. Die Verwendung eines einzigen Regelobjekts bedeutet, dass alle berechneten Attribute für das einzelne Regelobjekt höchstens ein Mal berechnet werden und keine identischen Berechnungen für viele identische Regelobjekte ausgeführt werden müssen.

Die Wiederverwendung von Regelobjekten aus dem Pool ist garantiert unbedenklich, da die Kernprinzipien von CER sicherstellen, dass jede Berechnung ausschließlich von ihren Eingabewerten abhängig ist, was wiederum bedeutet, dass identische Eingaben identische Ausgaben gewährleisten.

Einsatzsituationen für interne Regelobjekte: Sie sollten diesen Modus für die bedingte Erstellung von Regelobjekten gemäß Regeln oder in Fällen verwenden, in denen die Initialisierungsattributwerte aus anderen Regeln berechnet werden. Ein weiterer Verwendungsfall liegt bei einem Regelobjekt vor, das nur einen Zwischenschritt in einer Berechnung darstellt.

Bei einer sehr komplexen Berechnung sollten Sie davon ausgehen, dass ein einziges externes Regelobjekt ein Attribut für das Berechnungsergebnis enthält, dass mehrere externe Regelobjekte für Eingabedaten verwendet werden, die nicht aus Regeln stammen, und dass möglicherweise eine große Anzahl von internen Regelobjekten für die Zwischenberechnungsschritte vorhanden sind.

Falls Sie Regelobjekte verwenden, die immer vorhanden sind und/oder für Ausdrücke "readall" (siehe „readall" auf Seite 224) zugänglich sein müssen, sollten Sie möglicherweise der direkten Erstellung von externen Regelobjekten im Code den Vorzug geben.

Verarbeitung von Datentypen

Jedes Attribut und jeder Ausdruck in einem CER-Regelwerk gibt (bei Anforderung) einen Datenteil zurück. CER unterstützt eine flexible Gruppe von Datentypen, die für jedes Attribut und bei einigen Ausdrücken im Regelwerk angegeben werden müssen.

Unterstützte Datentypen

CER unterstützt die folgenden Datentypen:

- Regelklassen
- Java-Klassen
- Anwendungscodetabellen

Regelklassen: Jede in Ihrem Regelwerk definierte CER-Regelklasse kann in demselben Regelwerk als Datentyp verwendet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ruleclassDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
```

```
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="favoritePet">
      <type>
        <!-- The type of this attribute is a rule class
             defined elsewhere in this rule set.        -->
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Übernahme

CER unterstützt die einfache Implementierungsübernahme für Regelklassen.

Eine Regelklasse kann optional eine Deklaration *extends* angeben, um sich für eine andere Regelklasse in demselben Regelwerk als Unterklasse zu definieren.

Eine Unterregelklasse übernimmt die berechneten Regelattribute aller ihr übergeordneten Klassen und kann optional jedes dieser Attribute überschreiben, um abweichende Ableitungsberechnungsregeln bereitzustellen.

Eine Unterregelklasse übernimmt auch die initialisierten Regelattribute aller ihr übergeordneten Klassen. Jeder Ausdruck "create" (siehe „create" auf Seite 191) für die Unterregelklasse muss den Wert der initialisierten Attribute für alle übergeordneten Regelklassen der Unterregelklasse *vor* allen Deklarationen für die Unterregelklasse selbst angeben.

CER lässt die Deklaration eines Attributs als abstrakt zu (siehe „abstract" auf Seite 172). Jede Regelklasse, die ein abstraktes Attribut definiert oder übernimmt (jedoch nicht überschreibt) muss selbst als abstrakt deklariert sein. Eine abstrakte Klasse kann nicht in einem Ausdruck "create" (siehe „create" auf Seite 191) verwendet werden.

CER lässt zu, dass immer dort eine Regelobjektinstanz einer Regelklasse zurückgegeben wird, wo eine der übergeordneten Regelklassen erwartet wird.

Der CER-Regelwerkvalidierer meldet einen Fehler, falls ein Ausdruck in Ihrem Regelwerk versucht, einen inkompatiblen Wert zurückzugeben:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ruleclassInheritance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">


  <!-- The CER rule set validator will insist that this
       rule class is marked abstract, because it contains
       an abstract rule attribute. -->
  <Class name="Resource" abstract="true">
    <Initialization>
      <!-- Whenever a Resource rule object is created,
           its owner must be initialized.

           Since Resource is abstract, it cannot itself be
           used in a <create> expression, only concrete
           subclasses can. -->
      <Attribute name="owner">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
    </Initialization>


    <!-- The monetary value of the resource. -->
    <Attribute name="value">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Every resource has an amount, but it's
             calculated in a sub-class-specific way. -->
        <abstract/>
      </derivation>
    </Attribute>
  </Class>

  <!-- A building is a type of resource. -->
  <Class name="Building" extends="Resource">
    <!-- The physical address of the building,
         e.g. 123 Main Street.

         The address value must be specified
         in addition to the inherited owner
         rule attribute, which is an
         initialized attribute on the super-rule
         class. -->
    <Initialization>
      <Attribute name="address">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>


    <!-- Building is a concrete class
         (no pun intended!), and so the CER
         rule set validator will insist that this
         class inherits or declares a calculation
         for all inherited abstract rule attributes. -->
    <Attribute name="value">
```

```
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <arithmetic operation="-">
        <reference attribute="purchasePrice"> </reference>
        <reference attribute="outstandingMortgageAmount"/>
      </arithmetic>

    </derivation>
  </Attribute>

  <!-- The price originally paid for the building. -->
  <Attribute name="purchasePrice">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- The amount of outstanding loans or mortgages against
       this building. -->
  <Attribute name="outstandingMortgageAmount">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Vehicle" extends="Resource">
  <Initialization>
    <Attribute name="registrationPlate">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
  </Initialization>

  <Attribute name="value">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <!-- The value of this type of Resource is
           directly specified, rather than calculated.-->
      <specified/>

    </derivation>
  </Attribute>

</Class>

<Class name="Person">

  <!-- A sample attribute showing how initialized attributes
       are inherited. -->
  <Attribute name="sampleBuilding">
    <type>
      <ruleclass name="Building"/>
    </type>
    <derivation>
```

```
        <create ruleclass="Building">
          <!-- The first initialized rule attribute
               is inherited from Resource.

               Set this person to be the owner -->
          <this/>
          <!-- The second initialized rule attribute
            is specified directly on Building.

            Set the address of the Building. -->
          <String value="123 Main Street"/>
        </create>
      </derivation>
    </Attribute>

    <!-- a sample attribute which shows how a Building can be
         returned as a Resource (because a Building *IS* a
         Resource -->
    <Attribute name="sampleResource">
      <type>
        <ruleclass name="Resource"/>
      </type>
      <derivation>
        <reference attribute="sampleBuilding"/>
      </derivation>
    </Attribute>


  </Class>

</RuleSet>
```

Stammregelklasse

Falls eine Regelklasse nicht mit "extends" angibt, dass eine andere Regelklasse erweitert wird, erweitert die Regelklasse automatisch die Stammregelklasse von CER, die ein einziges Regelattribut `description` enthält.

Das Regelattribut `description` stellt eine lokalisierbare Beschreibung der Regelobjektinstanz bereit. Regelklassen können für ihre Regelobjektinstanzen jederzeit die Ableitung der Regelberechnung von `description` überschreiben.

Jede Regelklasse übernimmt letztlich aus der Stammregelklasse (und enthält somit ein Regelattribut `description`). Dies entspricht in etwa der Tatsache, dass alle Java-Klassen aus `java.lang.Object` übernehmen.

Die Standardimplementierung des Regelattributs `description`, die von der Stammregelklasse bereitgestellt wird, verwendet den Ausdruck "defaultDescription" (siehe „`defaultDescription`” auf Seite 200).

Java-Klassen: Jede Java-Klasse im Klassenpfad Ihrer Anwendung kann als Datentyp in einem CER-Regelwerk verwendet werden.

Wichtig: Beim Speichern von Regelobjekten in der Datenbank können Sie nur Datentypen verwenden, für die bei CER ein Typhandler registriert ist.

CER enthält Typhandler für die am häufigsten verwendeten Datentypen.

Paketnamen

Der Name einer Java-Klasse muss mit ihrem Paketnamen vollständig qualifiziert sein. Dies gilt jedoch nicht für Klassen in den folgenden Paketen:

- `java.lang.*`
- `java.util.*`

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaclassDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="isMarried">
      <type>
        <!-- java.lang.Boolean does not need its
             package specified -->
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <!-- Fully qualified name to a Cúram class -->
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>


  </Class>

</RuleSet>
```

Tipp: Primitive Java-Typen wie beispielsweise `boolean` können in CER nicht verwendet werden. Verwenden Sie stattdessen die entsprechenden Klassenäquivalente (z. B. `Boolean`).

Unveränderliche Objekte

Eines der Hauptprinzipien von CER besteht darin, dass ein einmal berechneter Wert nicht geändert werden kann.

Zur Einhaltung dieses Prinzips müssen alle von Ihnen verwendeten Java-Klassen *unveränderlich* sein.

Wichtig: Falls Sie eine *veränderliche* Java-Klasse in Ihrem CER-Regelwerk als Datentyp verwenden, müssen Sie sicherstellen, dass kein Java-Code versucht, den Wert eines Objekts dieser Java-Klasse zu ändern. CER kann die Zuverlässigkeit von Berechnungen nicht gewährleisten, wenn zugrunde liegende Werte geändert werden.

Glücklicherweise gibt es eine breite Palette von unveränderlichen Klassen, die normalerweise die meisten Anforderungen an Datentypen erfüllen können. Im Allgemeinen lässt sich anhand des Javadoc einer Java-Klasse feststellen, ob diese Klasse unveränderlich ist.

Die folgenden aufgelisteten unveränderlichen Klassen werden aller Wahrscheinlichkeit nach Ihre Anforderungen erfüllen:

- `java.lang.String`
- `java.lang.Boolean`
- Implementierungen von `java.lang.Number`. CER konvertiert auf jeden Fall Instanzen von `Number` in ein eigenes numerisches Format (gestützt durch `java.math.BigDecimal`), bevor Rechenoperationen oder Vergleiche ausgeführt werden.
- Implementierungen von `java.util.List`, die die optionalen Operationen dieser Klasse *nicht* unterstützen (siehe JavaDoc für `List`).
- `curam.util.type.Date`
- Implementierungen von `curam.creole.value.Message`.
- `curam.creole.value.CodeTableItem`

Übernahme

CER erkennt die Übernahmehierarchie von Java-Klassen und -Schnittstellen.

CER lässt zu, dass immer dort der Wert einer Java-Klasse zurückgegeben wird, wo eine der übergeordneten Java-Klassen oder -Schnittstellen erwartet wird:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaclassInheritance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="isMarried">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarriedAsObject">
      <type>
        <!-- For the sake of example, returning this
             value as an java.lang.Object (which is
             unlikely to be useful in a "real" rule
             set. -->
        <javaclass name="Object"/>
      </type>
      <derivation>
        <!-- This is ok, as a Boolean *IS* an Object. -->
        <reference attribute="isMarried"/>
      </derivation>
    </Attribute>

    <Attribute name="isMarriedAsString">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- The CER rule set validator would report the error
             below (as a Boolean *IS NOT* an String):

             ERROR    Person.isMarriedAsString
             Example_javaclassInheritance.xml(28, 41)
             Child 'reference' returns 'java.lang.Boolean',
             but this item requires a 'java.lang.String'. -->
        <!-- <reference attribute="isMarried"/> -->
```

```
        <!-- (Declaring as specified so that this example
              builds cleanly) -->
        <specified/>

      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Parametrisierte Klassen

In Java 5 wurde die Unterstützung von parametrisierten Klassen eingeführt. CER ermöglicht die Verwendung von parametrisierten Java-Klassen in Ihrem Regelwerk.

Die Parameter für eine parametrisierte Klasse werden einfach innerhalb der Deklaration von <javaclass> aufgelistet:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaclassParameterized"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="favoriteWords">
      <type>
        <!-- A list of Strings -->
        <javaclass name="List">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="luckyNumbers">
      <type>
        <!-- A list of Numbers -->
        <javaclass name="List">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="children">
      <!-- A list of Person rule objects.

           Because java.util.List can be parameterized with
           any Object, we can use a rule class as a parameter.
      -->
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
  <!-- The dogs owned by this person. -->
  <Attribute name="dogs">
    <type>
      <javaclass name="List">
        <ruleclass name="Dog"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- The cats owned by this person. -->
  <Attribute name="cats">
    <type>
      <javaclass name="List">
        <ruleclass name="Cat"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>


  <!-- All the pets owned by this person. -->
  <Attribute name="pets">
    <type>
      <javaclass name="List">
        <ruleclass name="Pet"/>
      </javaclass>
    </type>
    <derivation>
      <joinlists>
        <fixedlist>
          <listof>
            <javaclass name="List">
              <ruleclass name="Pet"/>
            </javaclass>
          </listof>
          <members>
            <!-- all the dogs - dogs are a type of pet -->
            <reference attribute="dogs"/>
            <!-- all the cats - cats are a type of pet -->
            <reference attribute="cats"/>

            <!-- CER will not allow "children" in this
                 expression; a child is not a pet regardless
                 of whether he or she is adorable or claws
                 the furniture.  -->
            <!-- CANNOT BE USED -->
            <!-- <reference attribute="children"/> -->
            <!-- CANNOT BE USED -->

          </members>
        </fixedlist>

      </joinlists>
    </derivation>
  </Attribute>


</Class>

<Class abstract="true" name="Pet">
```

```
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Dog" extends="Pet">

    <Attribute name="favoriteTrick">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Cat" extends="Pet">

    <Attribute name="numberOfLives">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- It's well known that every cat
             has 9 lives. -->
        <Number value="9"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

CER ermöglicht die Verwendung jedes beliebigen Typs (inklusive Regelobjekte) für Parameter, die `java.lang.Object` zulassen. CER setzt die starke Typisierung selbst dann durch, wenn der Parameter (z. B. eine Regelklasse) dynamisch definiert ist. CER erkennt außerdem die Übernahmehierarchie von Regelklassen, wenn ermittelt wird, ob eine parametrisierte Klasse einer anderen Klasse zugeordnet werden kann.

Codetabellen: Jede Anwendungscodetabelle kann in einem CER-Regelwerk als Datentyp verwendet werden.

Tipp: Die Codetabelle muss *nicht* zwangsläufig zur Entwicklungszeit vorhanden sein. Falls ein Benutzer mit Administratorberechtigung die Onlineanwendung zum Erstellen einer neuen Codetabelle verwendet, kann diese Codetabelle anschließend in dynamisch definierten CER-Regelwerken als Datentyp verwendet werden.

Um eine Instanz eines Codetabelleneintrags zu erstellen (um also auf einen bestimmten Eintrag in der Codetabelle zu verweisen), verwenden Sie den Ausdruck "code" (siehe„Code“ auf Seite 188).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_codetableentryDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
```

```
  <Class name="Person">

    <Attribute name="gender">
      <type>
        <!-- The value of this attribute will
             be an entry from the "Gender" Cúram
             code table. -->
        <codetableentry table="Gender"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMale">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Use "Code" to create a codetableentry value
        for comparison. -->
        <equals>
          <reference attribute="gender"/>
          <Code table="Gender">
            <!-- The code from the code table -->
            <String value="MALE"/>
          </Code>
        </equals>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Angabe von Datentypen

Für jedes (berechnete oder initialisierte) Attribut (siehe „Attribut” auf Seite 169) muss mit dem Element `type` ein Typ angegeben sein.

Bei den meisten Ausdrücken ist das Element `type` entweder festgelegt (beispielsweise gibt der Ausdruck "all" - siehe „all” auf Seite 175 - immer den Typ `Boolean` zurück) oder kann übernommen werden (ein Ausdruck "reference" - siehe „reference” auf Seite 229 - gibt den Typ zurück, der durch das Attribut deklariert ist, auf das verwiesen wird).

Bei manchen Ausdrücken muss das Element `type` jedoch explizit angegeben werden. Hierbei handelt es sich um die folgenden Ausdrücke:

- Ausdruck "call" (siehe „call” auf Seite 183)
- „choose” auf Seite 185.

Außerdem deklariert der Ausdruck "fixedlist" (siehe „fixedlist” auf Seite 206) den Typ des im Element `List` zurückgegebenen Elements in seiner Anweisung `listof`.

Verarbeitung von Daten, die sich mit der Zeit ändern

Seit Cúram Version 6 unterstützt CER eine leistungsfähige Funktion namens "Zeitlinie". Eine CER-Zeitlinie ist einfach ein Wert, der im Lauf der Zeit variiert. Der Einfachheit dieses Konzepts ist es zu verdanken, dass durch die Verwendung von Zeitlinien in der Anwendung eine große Wirkung erzielt werden kann.

Zeitliniendaten - Beschreibung

Eine Zeitlinie ist eine Sequenz von Werten mit einem bestimmten Typ, bei der jeder Wert ab einem bestimmten Datum gültig ist (bis er durch einen anderen Wert abgelöst wird). Für jedes spezielle Datum besitzt eine Zeitlinie einen Wert, der auf dieses Datum anwendbar ist.

Beispiele von Daten, die als CER-Zeitlinien modelliert werden können: Das Konzept einer Zeitlinie soll anhand von einigen Beispielen für alltägliche Daten vorgestellt werden, die im Lauf der Zeit variieren können:

- Das Gesamteinkommen einer Person kann im Lauf der Zeit schwanken, wenn die Person Gehaltserhöhungen erhält oder den Arbeitgeber wechselt. Da das Einkommen einer Person zu einem bestimmten Zeitpunkt als Zahl dargestellt werden kann, können die Einkommensschwankungen einer Person, die im Lauf der Zeit auftreten können, als Zeitlinie von Zahlen (Typ "Number") dargestellt werden, die hier aus Gründen der Einfachheit mit `Timeline<Number>` bezeichnet wird.

  Anmerkung: Die im vorliegenden Handbuch verwendete Schreibweise ist absichtlich an generische Java-Angaben angelehnt.
- Ungeachtet des Gesamteinkommens ändert sich möglicherweise auch der Beschäftigungsdatensatz einer Person, wenn sie den Arbeitgeber wechselt oder falls Zeiten ohne Beschäftigung vorliegen. Falls eine Person zu einem bestimmten Zeitpunkt höchstens eine *primäre* Beschäftigung besitzt, kann der Verlauf für die primäre Beschäftigung der Person als `Timeline<Employment>` dargestellt werden (hierbei steht "Employment" für eine Regelklasse oder einen Java-Typ mit Beschäftigungsdetails). In Zeiten ohne primäre Beschäftigung besteht der Wert der Zeitlinie aus einem speziellen Markierungswert wie beispielsweise `null` (als Darstellung für "ohne Beschäftigung").
- Eine Person besitzt möglicherweise einen Gegenstand, der später dann nicht mehr vorhanden ist (jemand hat z. B. ein Fahrzeug erworben und anschließend verkauft). Für jedes beliebige Datum kann bestimmt werden, ob die Person Besitzer des Fahrzeugs ist oder nicht. Dies kann als boolescher Wert modelliert werden. Für den Zeitverlauf kann die Bedingung, ob die Person zu einem bestimmten Zeitpunkt Besitzer des Fahrzeugs war, als `Timeline;Boolean>` modelliert werden. Der Wert der Zeitlinie lautet vor dem Kaufdatum des Fahrzeugs `false`. Ab dem Kaufdatum lautet der Wert `true`, der bis einschließlich zum Verkaufsdatum gültig ist (oder "bis zu einer anderslautenden Mitteilung", falls ein Verkauf des Fahrzeugs nicht bekannt ist).
- Ähnlich gibt es für jede Person ein Geburtsdatum und eines Tages ein Sterbedatum. Für Personen, die noch am Leben sind, wird als Sterbedatum ein leerer Wert eingetragen. An jedem beliebigen Datum ist die Person entweder am Leben oder verstorben, weshalb der abgeleitete Wert für die Frage "Lebt die Person?" als boolescher Wert modelliert werden kann. Für den Zeitverlauf kann die Bedingung, ob die Person am Leben ist, als `Timeline<Boolean>` modelliert werden. Der Wert der Zeitlinie lautet vor dem Geburtsdatum der Person `false`. Ab dem Geburtsdatum lautet der Wert `true`, der bis einschließlich zum Sterbedatum gültig ist (oder "bis zu einer anderslautenden Mitteilung", falls für die Person kein Sterbedatum bekannt ist).
- Elternteile können viele Kinder haben, die an unterschiedlichen Tagen geboren wurden. Für jedes bestimmte Datum gibt es für das Elternteil eine Liste der Kinder, die zu diesem Zeitpunkt am Leben sind. Dies kann als `List<Person>` modelliert werden. Im Lauf der Zeit ändert sich die Liste der Kinder dadurch, dass weitere Kinder geboren werden oder Kinder die Volljährigkeit erreichen (bzw. im traurigeren Fall vor dem Erreichen der Volljährigkeit versterben). Dies kann als `Timeline<List<Person>>` modelliert werden.

Die obigen Beispiele sind in der folgenden Abbildung als Grafik dargestellt.

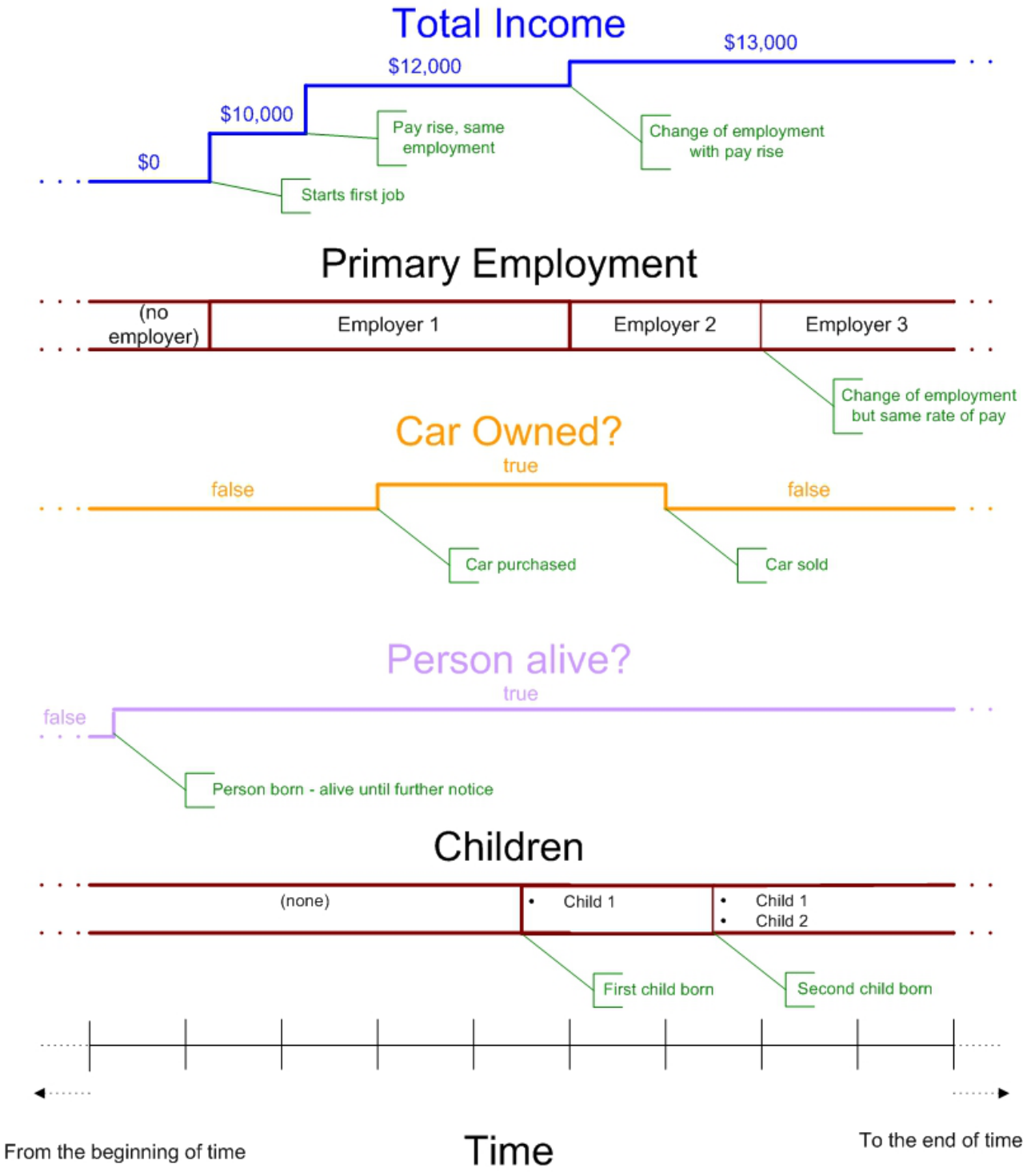

Abbildung 9. Beispiele für Zeitliniendaten

Beispiele für Daten, die nicht für Zeitlinien geeignet sind: Bevor Zeitlinien detaillierter beschrieben werden, soll an dieser Stelle vorsichtshalber darauf hingewiesen werden, dass es Daten gibt, die *nicht* für Zeitlinien geeignet sind.

Bestimmte Datentypen können nicht für Zeitlinien verwendet werden, da die Daten nicht mit der Zeit variieren. Gängige Beispiele:

- **Eindeutige Kennungen**

  Ein Charakteristikum einer eindeutigen Kennung ist die Tatsache, dass sie sich mit Absicht nicht im Lauf der Zeit ändert. Beispielsweise kann jeder Person eine eindeutige Sozialversicherungsnummer zugewiesen sein. Eine solche Sozialversicherungsnummer sollte als Zahl (Typ "Number") und nicht mit `Timeline<Number>` modelliert werden. Der Name einer Person kann sich hingegen mit der Zeit aufgrund von Eheschließung oder amtlicher Namensänderung ändern, was einer der Gründe dafür ist, dass der Name (abgesehen vom Mangel an Eindeutigkeit) keine gute Wahl für eine Kennung ist.

- **Datumsangaben**

  Daten des Typs "Date" ändern sich im Verlauf der Zeit nicht. Beispielsweise ist das Geburtsdatum einer Person ein bestimmtes Datum und sollte daher nicht als `Timeline<Date>`, sondern als Datum modelliert werden. Datumsangaben können verwendet werden, um eine Zeitlinie *aufzubauen*. Das Geburtsdatum und das Sterbedatum einer Person könnten beispielsweise verwendet werden, um eine Zeitlinie des Typs `Timeline<Boolean>` dafür aufzubauen, ob die Person am Leben ist. Die Datumsangaben selbst sind jedoch keine Zeitlinien.

  Anmerkung: In Cúram werden für bestimmte Datenteile (z. B. Angaben) möglicherweise *zwei* Verlaufsarten gespeichert:

  – Ein Verlauf der *Abfolgen* von Daten bezogen auf geänderte Lebensumstände. Dies bedeutet, dass das Auftreten eines Ereignisses in der Realität dazu führt, dass die Darstellung dieser Daten im System nicht mehr aktuell ist (z. B. Einkommensänderung einer Person aufgrund einer Gehaltserhöhung).

  – Verlauf von *Korrekturen* an Daten im System, bei denen festgestellt wird, dass im System eine falsche Darstellung der Lebensumstände vorliegt (wenn beispielsweise das Einstellungsdatum einer Person falsch eingetragen wurde).

  Dateneinträge mit Datumstyp können daher im System *korrigiert* werden, jedoch nie *Abfolgen* darstellen. Insofern kann es für den Dateneintrag einen *Korrekturverlauf* geben, aber dieser Korrekturverlauf (also die Datumsangaben für die Erstellung des Dateneintrags) ist bei der Erstellung von CER-Regeln nur selten von Belang.

  In CER-Regeln verwendete Datentypen sollten generell Lebensumstände und nicht Korrekturen an der Systemdarstellung modellieren. Verwenden Sie eine Zeitlinie, wenn sich diese Lebensumstände im Lauf der Zeit ändern. Verwenden Sie keine Zeitlinie, wenn die realen Daten nicht mit der Zeit variieren können.

- **Zeitpunktdaten**

  Einige Daten erfassen absichtlich Angaben, die nur für ein bestimmtes Datum gelten. Ein Datenelement für `surnameAtBirth` (Geburtsname) sollte beispielsweise als Zeichenfolge (Typ "String") und nicht als `Timeline<String>` modelliert werden. Ein Datenelement für `incomeAtRetirement` (Einkommen bei Renteneintritt) sollte als Zahl (Typ "Number") und nicht als `Timeline<Number>` modelliert werden.

Wichtig: Bei der Modellierung von Daten muss unbedingt beachtet werden, ob sich das Datenelement tatsächlich mit der Zeit ändert. Verwenden Sie eine Zeitlinie nur für diejenigen Datenelemente, deren Wert im Lauf der Zeit variieren kann.

Vergleich von Zeitlinienperspektive und Zeitpunktperspektive

CER behandelt Zeitlinien auf wirklichkeitsgetreue Weise. Beim Entwerfen von CER-Regeln können Sie daher gedanklich zwischen einer Zeitpunktperspektive und einer Zeitlinienperspektive wechseln.

In den folgenden Abschnitten sind diese Perspektiven anhand von Beispielen beschrieben. Als Erstes wird die Zeitpunktperspektive behandelt, die keine Zeitlinien beinhaltet. Anschließend wird das Beispiel nochmals betrachtet, dann jedoch aus einer Zeitlinienperspektive.

Zeitpunktperspektive: Angenommen, für eine Ableitung gilt die folgende Geschäftsanforderung:

Regel: Für ein Datum gilt eine Person jeweils als *alleinerziehendes Elternteil eines minderjährigen Kindes*, wenn Folgendes am entsprechenden Datum zutrifft:

- Die Person ist *nicht verheiratet*.
- Die Person hat ein *Kind im Alter von weniger als 16 Jahren*.

Ausgehend von dieser einfachen Anforderung kann eine einfache Gültigkeitstabelle dafür erstellt werden, ob eine Person *an einem bestimmten Datum* alleinerziehendes Elternteil eines minderjährigen Kindes ist:

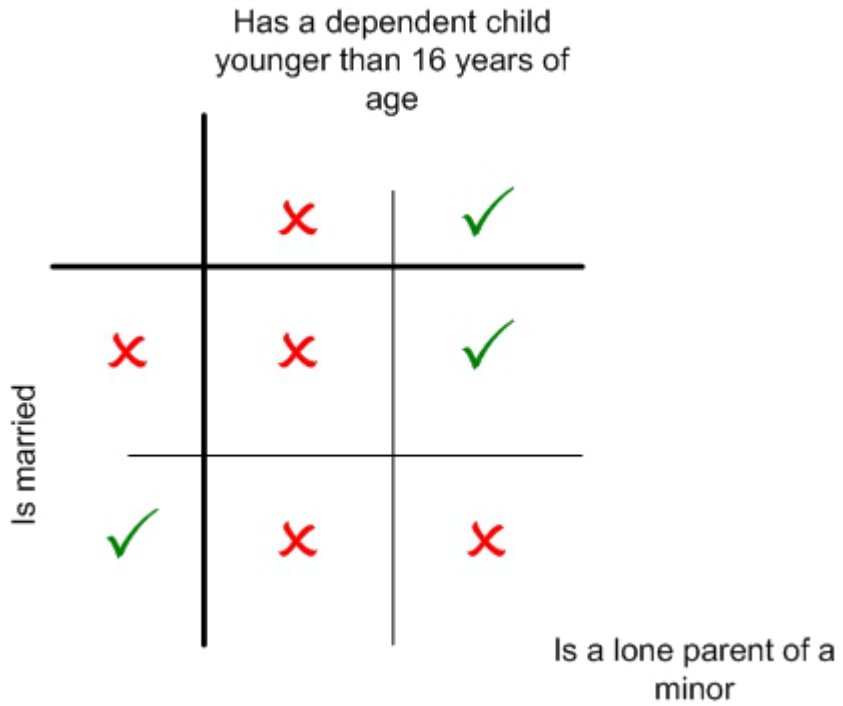

Abbildung 10. Gültigkeitstabelle für Regel "Alleinerziehendes Elternteil eines minderjährigen Kindes"

Die reale Änderung von Umständen soll anhand eines Beispiels veranschaulicht werden. Mary und Joe heiraten am 1. Januar 2001. Aus einer vorangegangenen Ehe, die am 30. November 1998 geschieden wurde, hat Joe einen Sohn namens James. James wurde am 1. Juni 1990 geboren. Am 30. April 2004 stirbt Joe (und Marys Ehe endet somit durch Verwitwung).

Anhand der obigen Gültigkeitstabelle kann für jede der Personen ermittelt werden, ob sie an einem bestimmten Datum *alleinerziehendes Elternteil eines minderjährigen Kindes* ist:

- Am 1. Oktober 1997 (um ein zufällig gewähltes Datum zu verwenden) ist Mary kein alleinerziehendes Elternteil eines minderjährigen Kindes, weil sie an diesem Datum zwar nicht verheiratet ist, jedoch keine Kinder hat.
- Am 2. Oktober 1997 ist Mary noch immer kein alleinerziehendes Elternteil eines minderjährigen Kindes, da sich ihre Lebensumstände seit dem Vortag nicht geändert haben.
- Am 30. November 1998 ist Joe kein alleinerziehendes Elternteil eines minderjährigen Kindes, weil sein Sohn an diesem Tag zwar jünger als 16 Jahre ist, Joe jedoch noch verheiratet ist.
- Am 1. Dezember 1998 wird Joe ein alleinerziehendes Elternteil eines minderjährigen Kindes, weil sein Sohn immer noch jünger als 16 Jahre ist, Joe jedoch nicht mehr verheiratet ist.
- Am 1. Januar 2001 ist Mary noch immer kein alleinerziehendes Elternteil eines minderjährigen Kindes. Sie hat zwar nun ein Kind, das jünger als 16 Jahre ist, aber sie ist verheiratet. Sie ist lediglich aus etwas anderen Gründen als zuvor kein alleinerziehendes Elternteil eines minderjährigen Kindes.
- Am 1. Januar 2001 ist Joe kein alleinerziehendes Elternteil eines minderjährigen Kindes mehr. Sein Kind ist zwar weiterhin jünger als 16 Jahre, aber Joe ist wieder verheiratet.
- Am 1. Mai 2004 wird Mary zu einem alleinerziehenden Elternteil eines minderjährigen Kindes, da ihre Ehe durch Joes Tod endet, James jedoch weiterhin ihr Kind und jünger als 16 Jahre ist.
- Am 1. Juni 2006 ist Mary kein alleinerziehendes Elternteil eines minderjährigen Kindes mehr, da James an diesem Tag 16 Jahre alt wird.
- Am 1. März (um erneut ein zufälliges Datum zu wählen) ist James kein alleinerziehendes Elternteil eines minderjährigen Kindes, da er nicht verheiratet und kinderlos ist.

Aus diesem Beispiel wird ersichtlich, dass die Gültigkeitstabelle für verschiedene Datumsangaben ausgewertet werden muss, um ein vollständiges Bild darüber zu erhalten, ob eine Person ein alleinerziehendes Elternteil eines minderjährigen Kindes ist. In gewissem Umfang müssen hierbei stichprobenartig Datumsangaben ausgesucht werden, die entweder als "möglicherweise relevant" eingestuft werden oder auch zufällig gewählt werden. Beispielsweise könnte erwartet werden, dass das Hochzeitsdatum von Mary ein relevanter Punkt ist. Es stellt sich jedoch heraus, dass ihre Eheschließung mit Joe *keinen* Einfluss auf ihren Status für die Eigenschaft "alleinerziehendes Elternteil eines minderjährigen Kindes" hat. Der Status von Joe für die Eigenschaft "alleinerziehendes Elternteil eines minderjährigen Kindes" wird durch die Eheschließung mit Mary hingegen *geändert*. Beim obigen Beispiel wurde nicht überprüft, ob Joe vor der Geburt von James ein alleinerziehendes Elternteil eines minderjährigen Kindes war.

Zeitlinienperspektive: Die Regel und die Umstände dieses Beispiels sollen nun aus einer Zeitlinienperspektive betrachtet werden.

Zunächst wird die Anforderung wie folgt etwas umformuliert:

Regel (umformuliert): Eine Person gilt immer dann als *alleinerziehendes Elternteil eines minderjährigen Kindes*, wenn Folgendes für die Person zutrifft:

- Die Person ist *nicht verheiratet*.
- Die Person hat ein *Kind im Alter von weniger als 16 Jahren*.

(Die Ausdrücke "für ein Datum" und "am entsprechenden Datum" wurden durch den Ausdruck "immer dann" ersetzt. Diese Umformulierung ist zwar nur geringfü-

gig, kann jedoch für den gedanklichen Wechsel von einer Betrachtung von Zeitpunkten hin zu einer Betrachtung von Daten, die sich mit der Zeit ändern, wichtig sein.)

Nachfolgend werden nun Zeitlinien (des Typs `Timeline<Boolean>` für die Zeiten gezeichnet, in denen für jede Person Folgendes zutrifft:

- Die Person ist verheiratet.
- Die Person hat ein Kind im Alter von weniger als 16 Jahren.

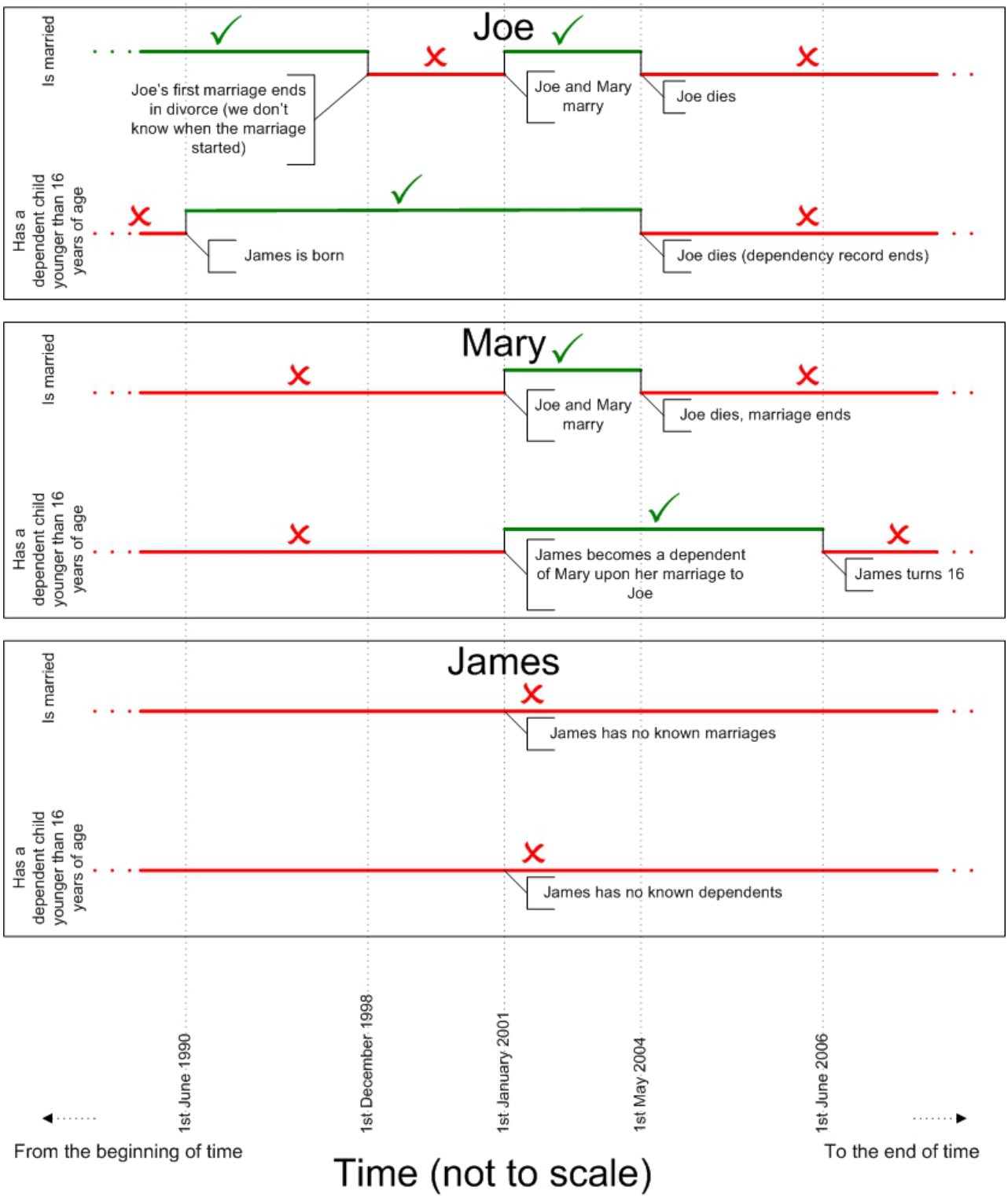

Abbildung 11. Zeitlinien für die Lebensumstände von Joe, Mary und James

Ausgehend von diesen Zeitlinien für die Lebensumstände von Joe, Mary und James können nun neue Zeitlinien erstellt werden, um die Änderung ihres jeweiligen Status für die Eigenschaft "Alleinerziehendes Elternteil eines minderjährigen Kindes" abzuleiten:

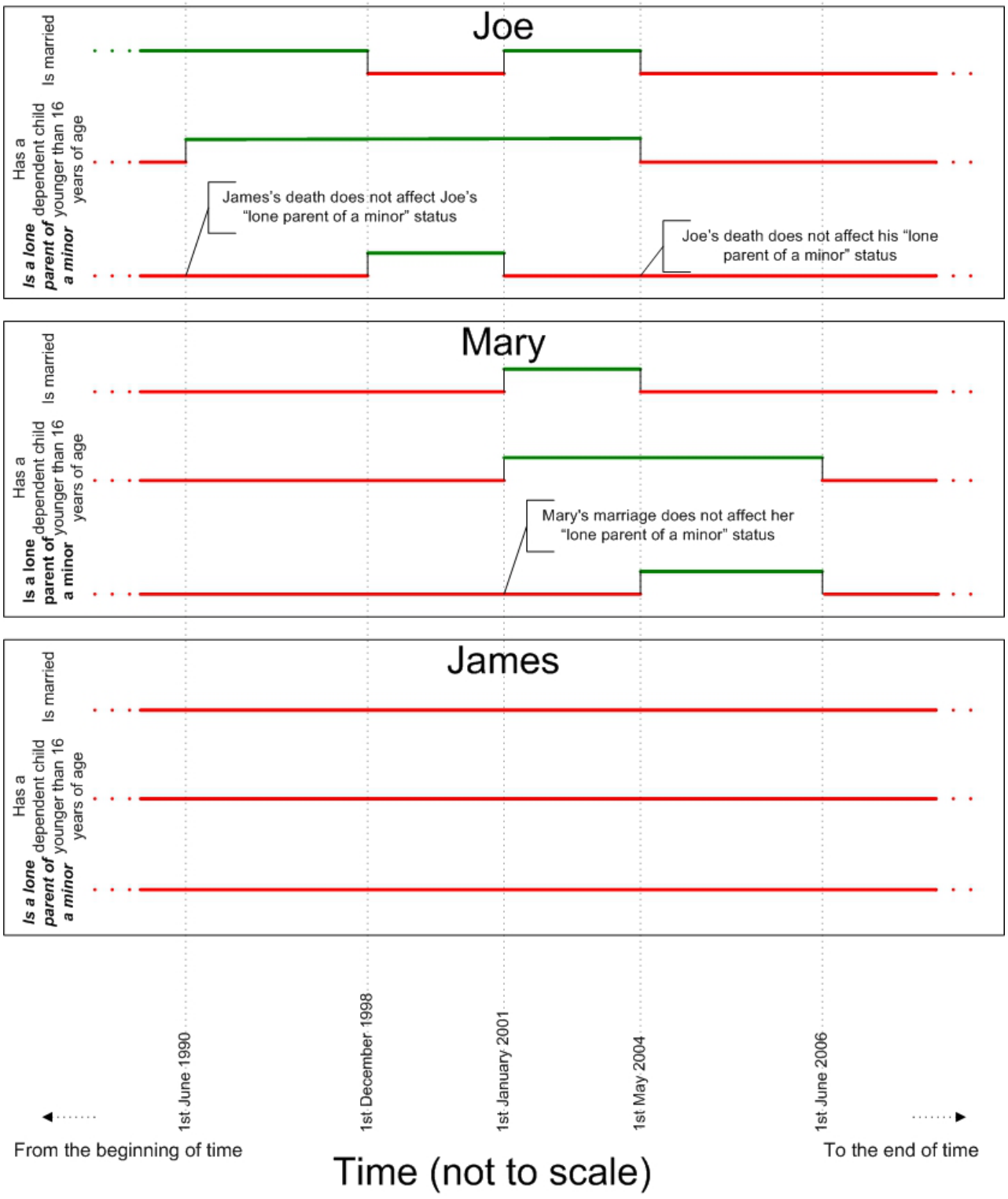

Abbildung 12. Zeitlinien für den Status der Eigenschaft "Alleinerziehendes Elternteil eines minderjährigen Kindes" von Joe, Mary und James

Es lässt sich feststellen, dass sofort erkannt werden kann, wie sich der Status jeder Person für die Eigenschaft "Alleinerziehendes Elternteil eines minderjährigen Kindes" ändert, ohne dass zu diesem Zweck Überlegungen hinsichtlich der möglicherweise relevanten Datumsangaben angestellt werden müssen:

- Joe ist alleinerziehendes Elternteil eines minderjährigen Kindes vom 1. Dezember 1998 bis einschließlich zum 31. Dezember 2001.
- Mary ist alleinerziehendes Elternteil eines minderjährigen Kindes vom 1. Mai 2004 bis einschließlich zum 30. Juni 2006.
- James ist zu keiner Zeit alleinerziehendes Elternteil eines minderjährigen Kindes.

Zeitlinien erstellen

Sie wissen nun, wie Ausdrücke auf bereits vorhandene Zeitliniendaten angewendet werden müssen, um Daten zu berechnen, die selbst eine Zeitlinie darstellen.

Im vorliegenden Abschnitt wird behandelt, wie Zeitliniendaten erstmalig erstellt werden.

Für die Erstellung von Zeitlinien gibt es zwei mögliche Verfahren:

- Unter Verwendung von Java-Code, entweder durch einen CER-Client oder innerhalb von Java-Code, der während eines von CER ausgehenden Aufrufs aufgerufen wird.
- In CER-Regeln durch die Verwendung von CER-Ausdrücken, die Zeitliniendaten aus primitiven (also keine Zeitlinie darstellenden) Daten erstellen.

Zeitlinien in Java-Code erstellen: In Java ist jedes Teil von Zeitliniendaten eine Instanz der parametrisierten Klasse `curam.creole.value.Timeline`. Vollständige Einzelangaben über diese Klasse enthält das JavaDoc, das in einer für Entwicklungszwecke verwendeten Installation der Anwendung unter `EJBServer/components/CREOLEInfrastructure/doc` verfügbar ist.

Jede Zeitlinie enthält eine Sammlung von Intervallen. Ein Intervall ist in diesem Zusammenhang ein *Wert*, der ab einem bestimmten *Startdatum* gültig ist. An den Konstruktor der Zeitlinie muss eine Sammlung von geeigneten Intervallen übergeben werden.

Nehmen wir beispielsweise an, es soll eine Zeitlinie des Typs `Timeline<Number>` mit den folgenden Intervallen erstellt werden (zur Erinnerung: eine Zeitlinie erstreckt sich unbegrenzt in die Vergangenheit und in die Zukunft):

- 0 bis einschließlich 31. Dezember 2000
- 10.000 vom 1. Januar 2001 bis einschließlich 30. November 2003
- 12.000 vom 1. Dezember 2004 bis zu einer anderslautenden Mitteilung

Das folgende Beispiel zeigt einen Java-Code, mit dem eine solche Zeitlinie erstellt werden kann:

```
package curam.creole.example;

import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class CreateTimeline {

  /**
   * Creates a Number Timeline with these interval values:
   * <ul>
   * <li>0 up to and including 31st December 2000;</li>
   * <li>10,000 from 1st January 2001 up to and including 30th
   * November 2003; and</li>
   * <li>12,000 from 1st December 2004 until further notice.</li>
   * </ul>
   */
```

```
  public static Timeline<Number> createNumberTimeline() {

    return new Timeline<Number>(

    // first interval, application from the "start of time"
        new Interval<Number>(null, 0),

        // second interval
        new Interval<Number>(Date.fromISO8601("20010101"), 10000),

        // last interval (until further notice)
        new Interval<Number>(Date.fromISO8601("20041201"), 12000)

    );

  }
}
```

Das nächste Beispiel zeigt Java-Code, der von einem CER-Ausdruck `call` aufgerufen werden kann, um eine Zeitlinie für das Alter einer Person bis zu ihrem 200. Geburtstag zu berechnen (zur Erinnerung: Zeitlinien für das Lebensalter müssen künstlich begrenzt werden, damit die Zeitlinie eine endliche Anzahl von Wertänderungen enthält):

```
package curam.creole.example;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class AgeTimeline {

  /**
   * Creates a timeline for the age of a person, artificially
   * limited to 200 birthdays.
   * <p>
   * Can be invoked from CER rules via a &lt;call&gt; expression.
   */
  public static Timeline<? extends Number> createAgeTimeline(
      final Session session, final Date dateOfBirth) {

    /**
     * The artificial limit, so that the age timeline has a finite
     * number of value changes.
     */
    final int NUMBER_OF_BIRTHDAYS = 200;

    final Collection<Interval<Integer>> intervals =
        new ArrayList<Interval<Integer>>(NUMBER_OF_BIRTHDAYS + 2);

    /*
     * age before date of birth will still be recorded as 0 -
     * create an initial interval application from the
     * "start of time"
     */
    final Interval<Integer> initialInterval =
        new Interval<Integer>(null, 0);
    intervals.add(initialInterval);

    /*
     * Identify each birthday up to the limit. Note that the person
     * is deemed to be age 0 even before the date-of-birth (see
```

```
     * above); so the interval here from the date of birth up to
     * the first birthday will be merged into a single interval by
     * the timeline; no matter (it's clearer to keep the logic as
     * is).
     */

    for (int age = 0; age <= NUMBER_OF_BIRTHDAYS; age++) {

      // compute the birthday date
      final Calendar birthdayCalendar = dateOfBirth.getCalendar();

      /*
       * NB use .roll rather than .add to get the correct
       * processing for leap years
       */
      birthdayCalendar.roll(Calendar.YEAR, age);
      final Date birthdayDate = new Date(birthdayCalendar);

      /*
       * the age applies from this birthday until the next birthday
       */
      intervals.add(new Interval<Integer>(birthdayDate, age));
    }

    final Timeline<Integer> ageTimeline =
        new Timeline<Integer>(intervals);

    return ageTimeline;

  }
}
```

Anmerkung: Zeitliniendaten werden generell eher außerhalb von Regeln durch CER-Clients erstellt.

Insbesondere enthält die Verarbeitung für Anspruchsberechtigung und Leistungshöhe in Cúram Version 6 Logik, die eine Konvertierung von Cúram-Angaben in Zeitliniendaten unterstützt.

Weitere Details finden Sie im Handbuch `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`.

Zeitlinien in CER-Regeln erstellen: Zeitliniendaten werden üblicherweise außerhalb von Regeln durch CER-Clients erstellt und zum Füllen eines CER-Attributs mit dem Mechanismus "specify" verwendet.

CER enthält jedoch auch einige Ausdrücke, mit deren Hilfe Zeitlinien direkt in CER-Regeln erstellt werden können:

- `Timeline` in Verbindung mit `Interval`
- `existencetimeline`

`Timeline` und `Interval`

Ein Zeitlinie kann nativ in CER-Regeln erstellt werden, indem zunächst explizit eine Liste von Intervallen erstellt und anschließend diese Liste zur Erstellung einer Zeitlinie verwendet wird.

In der Praxis sind solche festgelegten Zeitlinien eher als rein temporäre Maßnahmen geeignet, wenn Sie Ihr Regelwerk ausarbeiten. Nachfolgend finden Sie ein Beispiel für die Erstellung einer Zeitlinie mit "Timeline" und "Interval".

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Timeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateTimelines">

    <!-- This example uses <initialvalue> to set the value valid
         from the start of time. -->
    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <!-- Value from start of time -->
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- The remaining intervals -->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <Number value="10000"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2004-12-01"/>
                  </start>
                  <value>
                    <Number value="12000"/>
                  </value>
                </Interval>

              </members>
            </fixedlist>

          </intervals>
        </Timeline>

      </derivation>
    </Attribute>
```

```
    <!-- This example does not use <initialvalue>. -->
    <Attribute name="aStringTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="String"/>
          </intervaltype>

          <!-- The list of intervals must include one valid from the
             null date (start of time), otherwise an error will
             occur at runtime, if this expression is evaluated. -->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="String"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <!-- "from the start of time" -->
                    <null/>
                  </start>
                  <value>
                    <String value="Start of time string"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <String value="2001-only String"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <Date value="2002-01-01"/>
                  </start>
                  <value>
                    <String value="2002-onwards String"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>
          </intervals>
        </Timeline>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>
```

existencetimeline

Einige Geschäftsobjekte besitzen ein natürliches Start- und Enddatum, die zusammen einen Zeitraum für die *Existenz* des Geschäftsobjekts angeben. Das Startdatum und/oder das Enddatum kann optional sein. In diesen Fällen ist für den Existenzzeitraums des Geschäftsobjekts kein Ende definiert.

Beispiele:

- Beschäftigungsverhältnis, das beginnt und zu einem späteren Zeitpunkt endet.
- Gegenstand, der erworben und zu einem späteren Zeitpunkt verkauft wird.
- Person, die geboren wird und zu einem späteren Zeitpunkt verstirbt.

Das Startdatum und das Enddatum für ein Geschäftsobjekt können verwendet werden, um die Zeit in die folgenden drei Perioden (oder weniger, falls entweder kein Startdatum oder kein Enddatum angegeben ist) zu unterteilen:

- **Zeitraum vor der Existenz**

  Der Zeitraum vor dem Startdatum des Geschäftsobjekts (sofern ein Startdatum vorhanden ist).
- **Zeitraum der Existenz**

  Der Zeitraum vom Startdatum des Geschäftsobjekts bis einschließlich zum Enddatum des Geschäftsobjekts.
- **Zeitraum nach der Existenz**

  Der Zeitraum nach dem Enddatum des Geschäftsobjekts (sofern ein Enddatum vorhanden ist).

Häufig ist es möglich, einem Geschäftsobjekt für jeden dieser Zeiträume einen anderen Wert zuzuschreiben und aus diesen Werten eine Zeitlinie zu erstellen. CER enthält einen Ausdruck `existencetimeline`, um auf der Grundlage von optionalen Start- und Enddatumsangaben eine Zeitlinie für Werte vor der Existenz, während der Existenz und nach der Existenz zu erstellen.

Falls das Startdatum nicht vorhanden ist, enthält die Zeitlinie kein Intervall für die Zeit vor der Existenz. Wurde für einen Gegenstand beispielsweise kein Kaufdatum eingetragen, gilt sein effektiver Wert vom Beginn der Zeit an ohne einen Zeitraum mit "Nullwert".

Ist das Enddatum nicht vorhanden, enthält die Zeitlinie kein Intervall für die Zeit nach der Existenz. Falls beispielsweise für einen Gegenstand kein Verkaufsdatum bekannt ist, bleibt der Wert des Gegenstandes bis zu einer anderslautenden Mitteilung (also beliebig weit in die Zukunft hinein) erhalten.

Details über die Verwendung von Zeitlinien im CER-Editor enthält der Abschnitt „Existenzzeitlinie” auf Seite 135.

Operationen für Zeitlinien ausführen

CER-Zeitlinien sind hilfreich für das Speichern von Daten, die sich im Lauf der Zeit ändern. CER unterstützt eine Funktion namens "Zeitlinienoperation", die die Ausführung von CER-Ausdrücken für Zeitliniendatenelemente ermöglicht, um Zeitlinienergebnisse zu erzeugen.

Datumsangaben für Änderungen beibehalten: Alle Ausdrücke können so für Zeitlinien ausgeführt werden, dass die Datumsangaben für die Änderungen der Zeitlinienwerte mittels der Datumswerte für die Änderungen der Werte in der resultierenden Zeitlinie wirklichkeitsgetreu zugeordnet werden.

In den vorangegangenen Beispielen wurde das Konzept der Operationen für Zeitlinien (Zeitlinie für die Eigenschaft *Ist verheiratet*, Zeitlinie für die Eigenschaft *Hat ein Kind im Alter von unter 16 Jahren*) vorgestellt, mit denen eine Ausgabezeitlinie (Zeitlinie für die Eigenschaft *Alleinerziehendes Elternteil eines minderjährigen Kindes*) erzeugt wurde.

Formaler betrachtet lässt CER für jeden CER-Ausdruck, der für einen oder mehrere Werte ausgeführt wird, zu, dass dieser Ausdruck auch für eine Zeitlinie dieser Werte ausgeführt werden kann. Generell kann jede Operation, die auf Werte mit einem primitiven Typ (z. B. Datum, Anzahl, Zeichenfolge, boolescher Wert usw.) angewendet werden kann, um ein Ergebnis zu erzielen, stattdessen auch auf Zeitlinien dieser Werte (z. B. `Timeline<Date>`, `Timeline<Number>`, `Timeline<String>`, `Timeline<Boolean>`) angewendet werden, um als Ergebnis einen Zeitlinienwert zu erzielen.

CER enthält spezielle Ausdrücke namens "timelineoperation" (siehe „`timelineoperation`" auf Seite 248) und "intervalvalue" (siehe „`intervalvalue`" auf Seite 211), die für die anderen CER-Ausdrücke "verbergen", dass die Operation für Zeitlinien ausgeführt wird.

CER enthält beispielsweise den Ausdruck "sum" (siehe „`sum`" auf Seite 243), mit dem eine Liste von Zahlen addiert wird. Falls eine Person über mehrere Einkommen verfügt, können diese Einkommen für einen bestimmten Zeitpunkt summiert werden, um das Gesamteinkommen der Person zu diesem Zeitpunkt abzuleiten. Falls stattdessen jedoch *Zeitlinien* für die zeitraumbezogene Änderung dieser Einkommen vorhanden sind, kann der Ausdruck "sum" ebenso einfach verwendet werden, um abzuleiten, wie sich das Gesamteinkommen mit der Zeit ändert:

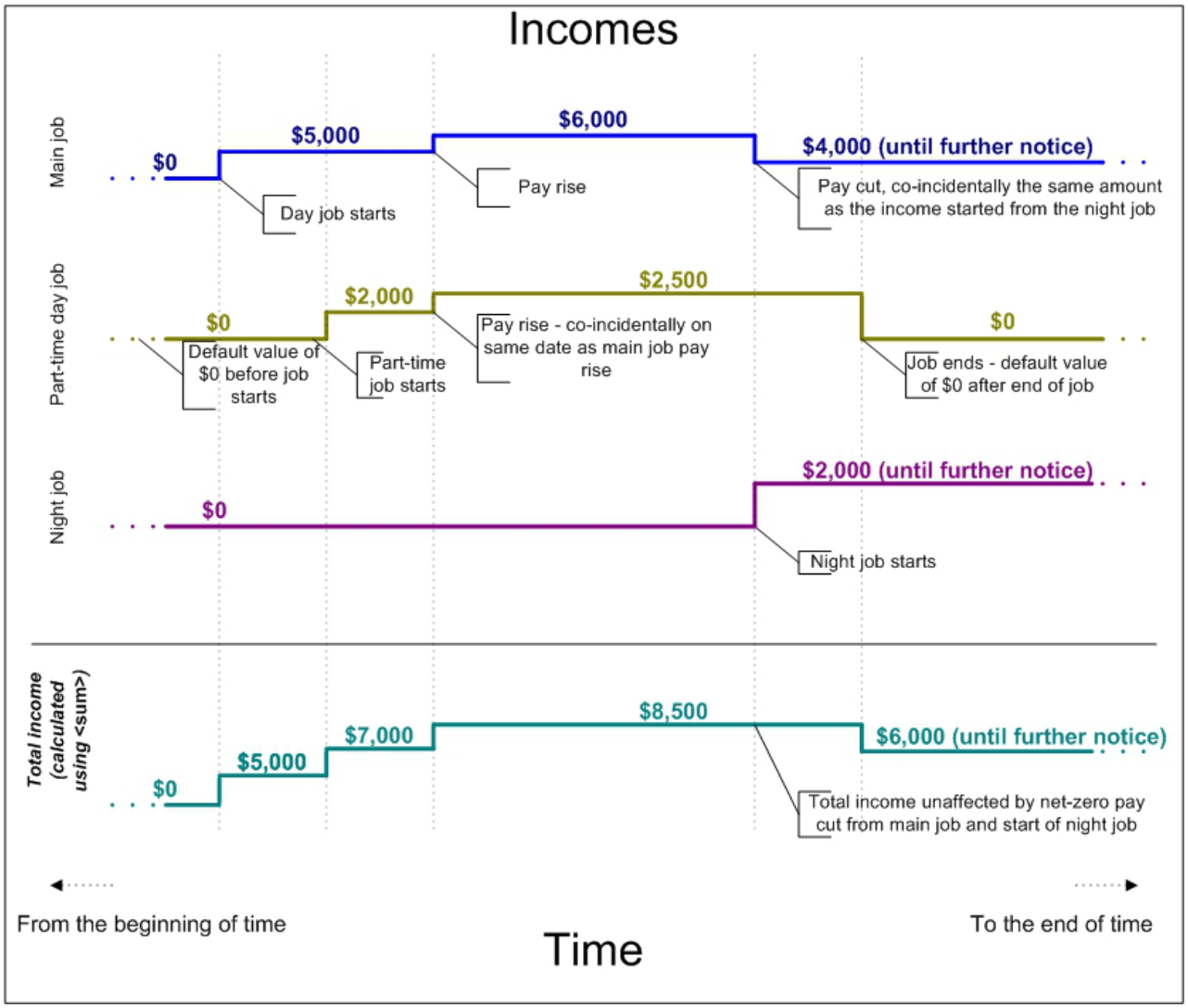

Abbildung 13. Zeitlinie für Gesamteinkommen mit Berechnung durch `sum`

Datumsänderung: Je nach den bestehenden Geschäftsanforderungen kann es sein, dass Sie eine Zeitlinie auf der Grundlage einer anderen Zeitlinie erstellen müssen, wobei die Datumsangaben für die Wertänderung in der resultierenden Zeitlinie von den entsprechenden Angaben in der Eingabezeitlinie abweichen.

CER enthält keine Ausdrücke für Datumsänderungen, da die erforderlichen Typen für die Datumsänderung tendenziell geschäftsspezifisch sind. Die empfohlene Strategie besteht darin, eine statische Java-Methode für die Erstellung der erforderlichen Zeitlinie zu erstellen und die statische Methode unter Verwendung des Ausdrucks "call" (siehe „`call`" auf Seite 183) aus den Regeln heraus aufzurufen.

Wichtig: Beim Implementieren eines Algorithmus für die Datumsänderung müssen Sie unbedingt sicherstellen, dass nicht versucht wird, eine Zeitlinie mit mehreren Werten für ein jeweiliges Datum zu erstellen, weil ein solcher Versuch während der Laufzeit fehlschlägt.

Beim Testen des Algorithmus sollten auch alle Tests für Grenzfälle (z. B. Schaltjahre oder Monate mit unterschiedlicher Anzahl von Tagen) einbezogen werden.

Beispiel für Datumsaddition

Die folgende Geschäftsanforderung ist gegeben: Eine Person kann einen Leistungsbezug erst drei Monate nach Ablauf des Leistungsbezugs erneut beantragen.

Zur Implementierung dieser Geschäftsanforderung ist bereits eine Zeitlinie `isReceivingBenefitTimeline` vorhanden, die die Zeiträume für den Leistungsbezug durch eine Person darstellt.

Sie benötigen nun eine weitere Zeitlinie namens `isDisallowedFromApplyingForBenefitTimeline`, in der die Zeiträume abgebildet sind, in denen der erneute Bezug dieser Leistung durch die Person nicht zulässig ist. Diese Zeitlinie wird durch eine Datumsaddition von 3 Monaten zu den Datumsangaben für die Wertänderung in der Zeitlinie`isReceivingBenefitTimeline` erzeugt:

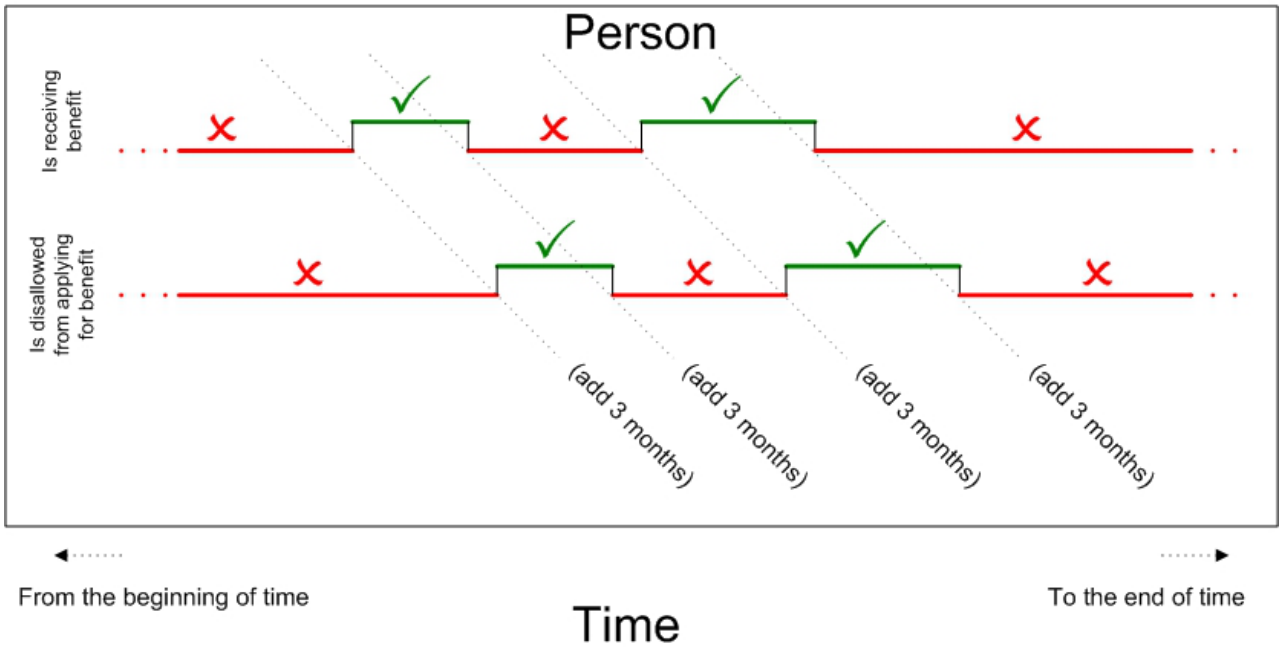

Abbildung 14. Anforderung für eine Zeitlinie mit Datumsaddition

Die folgende Beispielimplementierung zeigt eine statische Methode, die aus CER-Regeln heraus aufgerufen werden kann:

```
package curam.creole.example;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class DateAdditionTimeline {

  /**
   * Creates a Timeline based on the input timeline, with the date
   * shifted by the number of months specified.
```

```
 * <p>
 * Note that the timeline's parameter can be of any type.
 *
 * @param session
 *          the CER session
 * @param inputTimeline
 *          the timeline whose dates must be shifted
 * @param monthsToAdd
 *          the number of months to add to the timeline change
 *          dates
 * @param <VALUE>
 *          the type of value held in the input/output timelines
 * @return a new timeline with the values from the input
 *         timeline, shifted by the number of months specified
 */
public static <VALUE> Timeline<VALUE> addMonthsTimeline(
    final Session session, final Timeline<VALUE> inputTimeline,
    final Number monthsToAdd) {

  /*
   * CER will typically pass a Number, which must be converted to
   * an integer
   */
  final int monthsToAddInteger = monthsToAdd.intValue();

  /*
   * Find the intervals within the input timeline
   */
  final List<? extends Interval<VALUE>> inputIntervals =
      inputTimeline.intervals();

  /*
   * Amass the output intervals. Note that we map by start date,
   * because when adding months, it is possible for several
   * different input dates to be shifted to the same output date.
   *
   * For example 3 months after these dates: 2002-11-28,
   * 2002-11-29, 2002-11-30, are all calculated as 2003-02-28
   *
   * In this situation, we use the value from the earliest input
   * date only - input dates are processed in ascending order
   */
  final Map<Date, Interval<VALUE>> outputIntervalsMap =
      new HashMap<Date, Interval<VALUE>>(inputIntervals.size());

  for (final Interval<VALUE> inputInterval : inputIntervals) {
    // get the interval start date
    final Date inputStartDate = inputInterval.startDate();

    /*
     * Add the number of months - but n months after the start of
     * time is still the start of time
     */

    final Date outputStartDate;
    if (inputStartDate == null) {
      outputStartDate = null;
    } else {
      final Calendar startDateCalendar =
          inputStartDate.getCalendar();

      startDateCalendar.add(Calendar.MONTH, monthsToAddInteger);
      outputStartDate = new Date(startDateCalendar);

    }

    // check that this output date has not yet been processed
```

```
      if (!outputIntervalsMap.containsKey(outputStartDate)) {

        /*
         * the output interval uses the same value as the input
         * interval, but with a shifted start date
         */

        final Interval<VALUE> outputInterval =
            new Interval<VALUE>(outputStartDate,
                inputInterval.value());
        outputIntervalsMap.put(outputStartDate, outputInterval);
      }
    }

    // create a timeline from the output intervals
    final Collection<Interval<VALUE>> outputIntervals =
        outputIntervalsMap.values();
    final Timeline<VALUE> outputTimeline =
        new Timeline<VALUE>(outputIntervals);
    return outputTimeline;

  }
}
```

Beispiel für Datumsstreuung

Es besteht die folgende Geschäftsanforderung: Ein Fahrzeug ist für jeden Monat steuerpflichtig, in dem das Fahrzeug an einem oder mehreren Tagen „im Straßenverkehr eingesetzt wird”.

Anmerkung: Falls ein Fahrzeug im Verlauf eines Monats wieder im Straßenverkehr benutzt wird, bedeutet dies also, dass der Fahrzeughalter sicherstellen muss, dass die Steuer für den gesamten Monat rückwirkend gezahlt wird.

Zur Implementierung dieser Geschäftsanforderung ist bereits eine Zeitlinie `isOnRoadTimeline` vorhanden, die die Zeiträume für die Nutzung des Fahrzeugs "im Straßenverkehr" darstellt.

Sie benötigen nun eine weitere Zeitlinie namens `taxDueTimeline`, in denen die Zeiträume für die Steuerpflichtigkeit des Fahrzeugs abgebildet sind. Diese Zeitlinie ist eine Streuung der Datumsangaben aus der Zeitlinie `isOnRoadTimeline`:

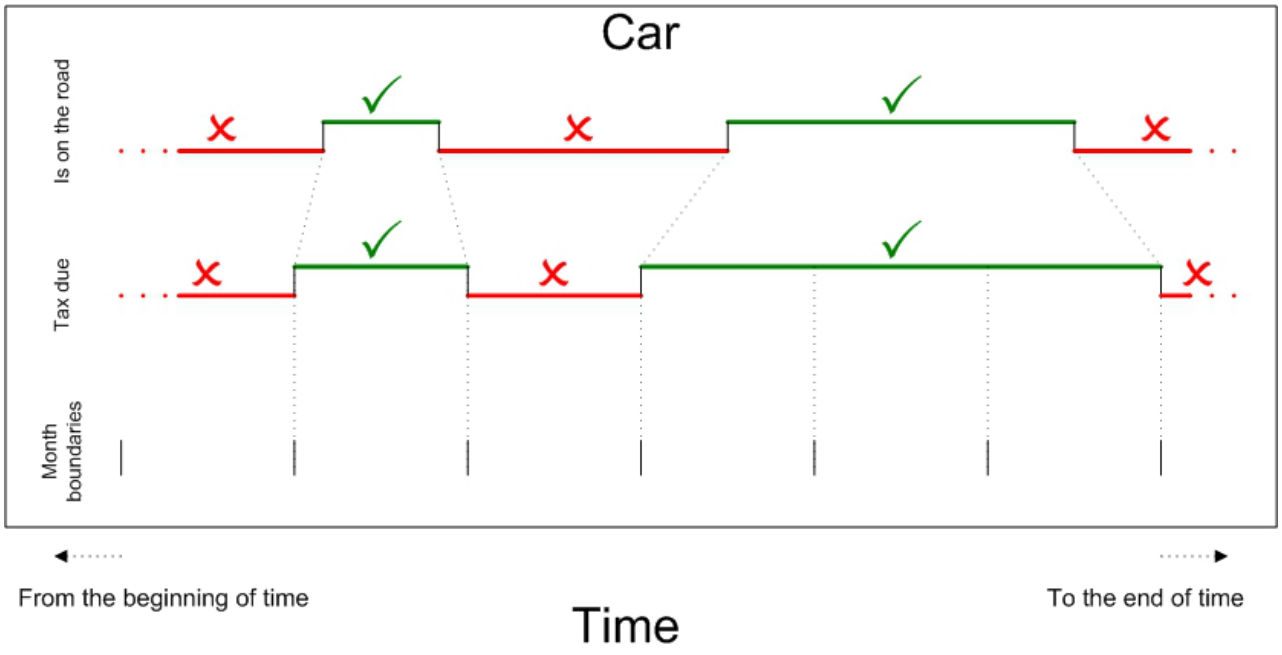

Abbildung 15. Anforderung für eine Zeitlinie mit Datumsstreuung

Die folgende Beispielimplementierung zeigt eine statische Methode, die aus CER-Regeln heraus aufgerufen werden kann:

```
package curam.creole.example;

import java.util.Calendar;
import java.util.Collection;
import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class DateSpreadingTimeline {

  /**
   * Creates a Timeline for the period for which a car must be
   * taxed.
   * <p>
   * The car must be taxed for the entire month for any month where
   * that car is on-the-road for one or more days during that
   * month.
   */
  public static Timeline<Boolean> taxDue(final Session session,
      final Timeline<Boolean> isOnRoadTimeline) {

    /*
     * Find the intervals within the input timeline
     */
    final List<? extends Interval<Boolean>> isOnRoadIntervals =
        isOnRoadTimeline.intervals();

    /*
     * Amass the output intervals. Note that we map by start date;
     * a car may go off the road during a month, which would imply
```

```
     * that no tax is required at the start of the next month, only
     * to return to the road part-way through the next month, in
     * which case it does require taxing after all.
     *
     * For example, car is put back on the road 2001-01-15, so tax
     * is required (retrospectively) from 2001-01-01.
     *
     * On 2001-01-24 the car is taken back off the road, so it's
     * possible that the car does not require taxing from
     * 2001-02-01.
     *
     * However, on 2001-02-05 the car is put back on the road, and
     * so it does require taxing from 2001-02-01 after all. The
     * resultant timeline will merge these periods to show that the
     * car requires taxing from 2001-01-01 onwards (thus covering
     * from 2001-02-01 too).
     */
    final Map<Date, Interval<Boolean>> taxDueIntervalsMap =
        new HashMap<Date, Interval<Boolean>>(
            isOnRoadIntervals.size());

    for (final Interval<Boolean> isOnRoadInterval :
isOnRoadIntervals) {
      // get the interval start date
      final Date isOnRoadStartDate = isOnRoadInterval.startDate();

      if (isOnRoadStartDate == null) {
        // at the start of time, the car must be taxed if it is on
        // the road
        taxDueIntervalsMap.put(null, new Interval<Boolean>(null,
            isOnRoadInterval.value()));
      } else if (isOnRoadInterval.value()) {
        /*
         * start of a period of the car being on-the-road - the car
         * must be taxed from the start of the month containing the
         * start of this period
         */

        final Calendar carOnRoadStartCalendar =
            isOnRoadStartDate.getCalendar();
        final Calendar startOfMonthCalendar =
            new GregorianCalendar(
                carOnRoadStartCalendar.get(Calendar.YEAR),
                carOnRoadStartCalendar.get(Calendar.MONTH), 1);
        final Date startOfMonthDate =
            new Date(startOfMonthCalendar);

        /*
         * Add to the map of tax due periods - note that this will
         * push out of the map any "tax not due" interval
         * speculatively added if the car went off-the-road during
         * the previous month
         */
        taxDueIntervalsMap.put(startOfMonthDate,
            new Interval<Boolean>(startOfMonthDate, true));
      } else {
        /*
         * Start of a period of the car being off the road -
         * speculate that from the start of next month, the car may
         * not require tax. This speculation will hold unless the
         * car is subsequently found to be put back on the road
         * next month, in which case this speculation will be
         * discarded (i.e. pushed out of the map).
         */

        final Calendar carOffRoadStartCalendar =
            isOnRoadStartDate.getCalendar();
```

```
        final Calendar startOfNextMonthCalendar =
            new GregorianCalendar(
                carOffRoadStartCalendar.get(Calendar.YEAR),
                carOffRoadStartCalendar.get(Calendar.MONTH), 1);
        startOfNextMonthCalendar.add(Calendar.MONTH, 1);

        final Date startOfNextMonthDate =
            new Date(startOfNextMonthCalendar);

        /*
         * Add to the map of tax due periods - note that this will
         * push out of the map any "tax not due" interval
         * speculatively added if the car went off-the-road during
         * the previous month
         */
        taxDueIntervalsMap.put(startOfNextMonthDate,
            new Interval<Boolean>(startOfNextMonthDate, false));
      }

    }

    // create a timeline from the tax due intervals
    final Collection<Interval<Boolean>> taxDueIntervals =
        taxDueIntervalsMap.values();
    final Timeline<Boolean> taxDueTimeline =
        new Timeline<Boolean>(taxDueIntervals);
    return taxDueTimeline;

  }
}
```

Ausgaben von Zeitlinien testen

Ein Regelattribut, das eine Zeitlinie von Werten zurückgibt, sollte ähnlich wie primitive (also nicht zeitlinienbezogene) Werte mit JUnit-Tests überprüft werden.

Zur Vereinfachung der Tests müssen Sie nicht überprüfen, ob der Ausdruck "timelineoperation" (siehe „timelineoperation” auf Seite 248) die Datumsangaben für Änderungen korrekt kumuliert (es sei denn, Sie wünschen eine solche Überprüfung). Zur weiteren Vereinfachung der Tests können Sie generell Eingabezeitlinien verwenden, die immer einen konstanten Wert haben.

Im folgenden Beispiel wird ein Regelattribut zugrunde gelegt, das (mit Zeitlinienverfahren) die Summe aus einer Liste von Werten berechnet:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_NumberSumTimeline"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Totalizer">

    <!-- The timelines to total -->
    <Attribute name="inputNumberTimelines">
      <type>
        <javaclass name="List">
          <javaclass name="curam.creole.value.Timeline">
            <javaclass name="Number"/>
          </javaclass>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
    <!-- The resultant total -->
    <Attribute name="totalTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <timelineoperation>
          <sum>
            <dynamiclist>
              <list>
                <reference attribute="inputNumberTimelines"/>
              </list>
              <listitemexpression>
                <intervalvalue>
                  <current/>
                </intervalvalue>
              </listitemexpression>
            </dynamiclist>

          </sum>
        </timelineoperation>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

Sie können nun einen einfachen Test schreiben, der Eingabezeitlinien verwendet, die für den gesamten Zeitraum einen konstanten Wert enthalten:

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Timeline;

public class TestForeverValuedTimelines extends TestCase {

  public void testNumberSumTimeline() {

    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final Totalizer totalizer =
        Totalizer_Factory.getFactory().newInstance(session);

    // use input values that do not vary over time

    final Timeline<Number> inputTimeline1 =
```

```
        new Timeline<Number>(1);
    final Timeline<Number> inputTimeline2 =
        new Timeline<Number>(2);
    final Timeline<Number> inputTimeline3 =
        new Timeline<Number>(3);

    totalizer.inputNumberTimelines().specifyValue(
        Arrays.asList(inputTimeline1, inputTimeline2,
            inputTimeline3));

    // check that the resultant timeline is 6 forever
    CREOLETestHelper.assertEquals(new Timeline<Number>(6),
        totalizer.totalTimeline().getValue());

  }
}
```

Tipp: Die Klasse `Timeline` besitzt einen komfortablen Konstruktor, der die Erstellung einer Zeitlinie mit einem konstanten Wert für den gesamten Zeitraum ermöglicht.

In manchen Situationen - wenn Sie beispielsweise einen eigenen Algorithmus für die Datumsänderung geschrieben haben oder grundsätzlich überprüfen müssen, ob die Datumsangaben für Änderungen von Eingabezeitlinien in den resultierenden Zeitlinien präzise wiedergegeben werden - gibt es andere Strategien, die Sie je nach Anforderung verfolgen können:

- **Strikte Überprüfung**

  Sie können überprüfen, ob die resultierende Zeitlinie einem erwarteten Zeitlinienwert exakt entspricht. (Die Gleichheitssemantik der Klasse `Timeline` funktioniert wie erwartet - zwei Zeitlinien gelten als gleich, wenn sie exakt dieselbe Gruppe von Intervallen enthalten, die Werte aus den beiden Zeitlinien also für jedes mögliche Datum identisch sind.)
- **Oberflächliche Überprüfung**

  Sie können überprüfen, ob die resultierende Zeitlinie den Wert enthält, den Sie für bestimmte Datumsangaben erwarten.

Dieses Beispiel zeigt die genaue Überprüfung einer resultierenden Zeitlinie.

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class TestStrictTimelineChecking extends TestCase {

  public void testNumberSumTimeline() {
```

```
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final Totalizer totalizer =
        Totalizer_Factory.getFactory().newInstance(session);

    // use input values that vary over time

    final Timeline<Number> inputTimeline1 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 1),
          new Interval<Number>(Date.fromISO8601("20010101"), 1.1)
        ));

    final Timeline<Number> inputTimeline2 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 2),
          new Interval<Number>(Date.fromISO8601("20020101"), 2.2)
        ));

    final Timeline<Number> inputTimeline3 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 3),
          new Interval<Number>(Date.fromISO8601("20030101"), 3.3)
         ));

    totalizer.inputNumberTimelines().specifyValue(
        Arrays.asList(inputTimeline1, inputTimeline2,
            inputTimeline3));

    // strictly check the exact value of the resultant timeline
    CREOLETestHelper.assertEquals(

    new Timeline<Number>(Arrays.asList(
      new Interval<Number>(null, 6),
      new Interval<Number>(Date.fromISO8601("20010101"), 6.1),
      new Interval<Number>(Date.fromISO8601("20020101"), 6.3),
      new Interval<Number>(Date.fromISO8601("20030101"), 6.6)
    )),

    totalizer.totalTimeline().getValue());

  }
}
```

Dieses Beispiel zeigt die etwas weniger genaue Überprüfung einer resultierenden Zeitlinie.

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
```

```
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class TestLaxTimelineChecking extends TestCase {

  public void testNumberSumTimeline() {

    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final Totalizer totalizer =
        Totalizer_Factory.getFactory().newInstance(session);

    // use input values that vary over time

    final Timeline<Number> inputTimeline1 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 1),
          new Interval<Number>(Date.fromISO8601("20010101"), 1.1)
        ));

    final Timeline<Number> inputTimeline2 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 2),
          new Interval<Number>(Date.fromISO8601("20020101"), 2.2)
        ));

    final Timeline<Number> inputTimeline3 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 3),
          new Interval<Number>(Date.fromISO8601("20030101"), 3.3)
        ));

    totalizer.inputNumberTimelines().specifyValue(
        Arrays.asList(inputTimeline1, inputTimeline2,
          inputTimeline3));

    /*
     * Do not strictly check that the resultant timeline is exactly
     * as expected - instead check the resultant timeline's value
     * on particular dates.
     *
     * It is possible that the timeline has incorrect values on
     * other dates, but depending on the purpose of your test, you
     * may wish to trade strictness for improved readability.
     */

    final Timeline<? extends Number> resultantTimeline =
        totalizer.totalTimeline().getValue();
    CREOLETestHelper.assertEquals(6.1,
        resultantTimeline.valueOn(Date.fromISO8601("20010101")));
    CREOLETestHelper.assertEquals(6.6,
        resultantTimeline.valueOn(Date.fromISO8601("20130101")));
  }
}
```

Eigenschaften von Zeitlinien

Jede CER-Zeitlinie besitzt einige wichtige Eigenschaften, die Sie kennen sollten, bevor Sie in Ihren CER-Regelwerken, Regeltests und Client-Code von CER mit Zeitlinien arbeiten:

- Jede CER-Zeitlinie ist (wie alle in CER verwendeten Datentypen) unveränderlich.

- Jeder Verweis auf eine Zeitlinie ist mit dem Typ des in der Zeitlinie enthaltenen Wertes parametrisiert. Der Typ kann ein primitiver Typ wie "String", "Date", "Number", "Boolean" usw. oder ein beliebiger komplexer Typ wie eine Regelklasse oder ein anderer parametrisierter Typ (z. B. eine Liste) sein. Auch der Parameter selbst sollte, wie bei anderen parametrisierten Typen in CER, ein unveränderliches Objekt sein.
- Jede CER-Zeitlinie erstreckt sich unendlich weit in die Vergangenheit und unendlich weit in die Zukunft. Anders ausgedrückt enthält jede CER-Zeitlinie für *jedes beliebige* Datum einen Wert (unabhängig davon, wie weit das Datum möglicherweise in der Vergangenheit oder in der Zukunft liegt).

  Anmerkung: Jede Zeitlinie deckt zwar einen unbegrenzten Zeitraum ab, enthält jedoch nur eine endliche Zahl von Datumsangaben, an denen sich ihre Werte ändern.
- Wenn eine CER-Zeitlinie erstellt wird, wird die Zeitlinie in eine Reihe von *Intervallen* aufgeteilt, bei der jedes Intervall einen konstanten Wert für einen Zeitraum innerhalb der Zeitlinie enthält. Aufeinander folgende Intervalle besitzen *immer* unterschiedliche Werte, da sie andernfalls in einem einzigen Intervall zusammengeführt werden würden. Jede CER-Zeitlinie erstreckt sich unendlich weit in die Vergangenheit und in die Zukunft.

  Anmerkung: Identische/unterschiedliche Werte werden durch die Semantik der Java-Methode `Object.equals(...)` erkannt. Alle als parametrisierter Typ für eine Zeitlinie verwendeten Typen müssen sensible Implementierungen von `Object.equals(...)` und `Object.hashCode()` haben.

Diese Eigenschaften haben eine Reihe von Konsequenzen:

- Innerhalb einer Zeitlinie sind keine "Lücken" möglich, denn alle Intervalle in einer Zeitlinie folgen kontinuierlich aufeinander.
- Es besteht keine Möglichkeit, dass eine Zeitlinie an einem bestimmten Datum beginnt. Unter bestimmten Umständen muss ein sensibler Standardwert ausgewählt werden. Wenn beispielsweise eine Zeitlinie des Typs "Timeline<Number>" für das Einkommen aus einer Beschäftigung vorhanden ist, sollte dieses Einkommen für alle Datumsangaben vor dem Startdatum der Beschäftigung 0 sein.
- Es besteht keine Möglichkeit, dass eine Zeitlinie an einem bestimmten Datum endet. Der letzte Wert in einer Zeitlinie gilt immer "bis zu einer anderslautenden Mitteilung", also beliebig weit in die Zukunft hinein. Unter bestimmten Umständen muss ein sensibler Standardwert ausgewählt werden. Wenn beispielsweise eine Zeitlinie des Typs "Timeline<Number>" für das Einkommen aus einer Beschäftigung vorhanden ist, sollte bei einem bekannten Enddatum für diese Beschäftigung das Einkommen für alle Datumsangaben nach der Beendigung der Beschäftigung 0 sein. Ist kein Enddatum für die Beschäftigung bekannt, gilt andernfalls das letzte Einkommen bis zu einer anderslautenden Mitteilung.
- Jeder Versuch, eine Zeitlinie zu erstellen, die nicht für jedes Datum einen Wert enthält, schlägt fehl. Insbesondere muss jede Zeitlinie einen Wert enthalten, der ab dem *Startzeitpunkt* (angegeben durch das Startdatum `Null`) gilt.
- Jede Zeitlinie kann eine endliche Anzahl von Wertänderungen enthalten. Dies stellt eine Einschränkung für Zeitlinien dar, mit denen Werte dargestellt werden, die sich beliebig oft ändern. Beispiel: Mit einer Zeitlinie des Typs "Timeline<Number>" wird das Alter einer Person dargestellt. Sie enthält den Wert 0 bis zum ersten Geburtstag der Person, den Wert 1 bis zu ihrem zweiten Geburtstag usw. Bei noch lebenden Personen kann nicht vorausgesagt werden, wie viele Geburtstage noch erfolgen werden. Aus diesem Grund muss eine prak-

tikable Begrenzung (z. B. 200) vorgenommen werden. Diese Einschränkung führt jedoch in der Praxis für gewöhnlich nicht zu Problemen.

Beispiel für Zeitlinienintervalle: Im Beispiel mit den Lebensumständen von Joe, Mary und James war Mary vor ihrer Heirat mit Joe kein alleinerziehendes Elternteil eines minderjährigen Kindes. Während ihrer Ehe mit Joe war sie ebenfalls kein alleinerziehendes Elternteil eines minderjährigen Kindes, jedoch aus anderen Gründen.

Wenn Marys Wert für `isLoneParentOfMinorTimeline` berechnet wird, werden bei den Voraussetzungen dieses Beispiels als Eingabe Marys Zeitlinien `isMarriedTimeline` und `hasMinorDependentsTimeline` verwendet.

CER erkennt jedes Datum, an dem sich die Eingabezeitlinie ändert. Für jedes dieser Daten berechnet CER folgendermaßen den resultierenden Wert (an diesem Datum) dafür, ob Mary an diesem Tag ein alleinerziehendes Elternteil eines minderjährigen Kindes ist:

- Datumsangaben mit Änderungen in Marys Zeitlinie `isMarriedTimeline`:
  - 1. Januar 2001
  - 1. Mai 2004
- Datumsangaben mit Änderungen in Marys Zeitlinie `hasMinorDependentsTimeline`:
  - 1. Januar 2001
  - 1. Juni 2006
- Datumsangaben mit einem oder mehreren geänderten Eingabewerten:
  - 1. Januar 2001 (beide Eingabezeitlinien von Mary ändern sich an diesem Datum)
  - 1. Mai 2004 (nur Marys Zeitlinie `isMarriedTimeline` ändert sich an diesem Datum)
  - 1. Juni 2006 (nur Marys Zeitlinie `hasMinorDependentsTimeline` ändert sich an diesem Datum)

Für jede dieser Datumsangaben wird der erforderliche Wert für `isLoneParentOfMinorTimeline` unter Verwendung der Logik mit primitiven booleschen Werten und der Gültigkeitstabelle berechnet:

Tabelle 2. Berechnung der Intervallwerte für Wert `isLoneParentOfMinorTimeline` von Mary

Datum mit Wertänderung bei einer oder mehreren Eingabezeitlinien	**Wert von `isMarriedTimeline` an diesem Datum**	**Wert von `hasMinorDependentsTimeline` an diesem Datum**	**Erforderlicher Wert von `isLoneParentOfMinorTimeline` an diesem Datum**
Startzeit (dieses Datum ist immer enthalten)	FALSE	FALSE	FALSE
1. Januar 2001	TRUE	TRUE	FALSE
1. Mai 2004	FALSE	TRUE	TRUE
1. Juni 2006	FALSE	FALSE	FALSE

Abschließend wird eine Zeitlinie mit den erforderlichen Werten für `isLoneParentOfMinorTimeline` erstellt. An diesem Punkt der Zeitlinienerstellung wird erkannt, dass der Wert für den Startzeitpunkt (FALSE) und der Wert für den

1. Januar 2001 (FALSE) identisch ist, und diese Intervalle werden in einem einzigen Intervall zusammengeführt, das sich vom Startzeitpunkt bis zum 1. Mai 2004 (jedoch nicht einschließlich) erstreckt, also dem Tag, an dem sich der Wert in TRUE ändert.

Anmerkung: Die resultierende Zeitlinie weist nur am 1. Mai 2004 und am 1. Juni 2006 eine Wertänderung auf.

Die Zeitlinie enthält absichtlich *keinen* Eintrag dafür, dass bei ihrer Erstellung der 1. Januar 2001 verwendet wurde, da sich der Wert der Zeitlinie an diesem Tag nicht geändert hat. Dieses Datum ist für die resultierende Zeitlinie gänzlich irrelevant.

Neuberechnung bei Datenänderung auslösen

Die CER-Funktionalität für die direkte Ausführung von Neuberechnungen wird nun durch den Abhängigkeitsmanager abgelöst (siehe „Abhängigkeitsmanager").

Es empfiehlt sich, der Verwendung des Abhängigkeitsmanagers den Vorzug vor den Neuberechnungsstrategien von CER zu geben.

Abhängigkeitsmanager

Die Anwendung umfasst einen Abhängigkeitsmanager, der dafür zuständig ist, die Abhängigkeiten zwischen Eingabedatenelementen ("Abhängigkeitsfaktoren") und Ausgabedatenelementen ("Abhängigkeitsprodukte") zu speichern und zu verwalten.

CER und die zugehörigen Clients (z. B. die Engine für Anspruchsberechtigung und Leistungshöhe sowie der Berater) sind nahtlos in den Abhängigkeitsmanager integriert, damit die Neuberechnung von CER-Ergebnissen stets unterstützt wird, wenn sich für CER-Berechnungen verwendete Eingaben ändern.

Dieses Kapitel vermittelt Ihnen in der nachstehenden Reihenfolge einen Überblick über den Abhängigkeitsmanager:

- Zugrunde liegende Konzepte des Abhängigkeitsmanagers und zu ihrer Beschreibung verwendete Terminologie
- Vom Abhängigkeitsmanager ausgeführte Funktionen
- Im Abhängigkeitsmanager enthaltene Batchverarbeitung
- Integration von CER beim Abhängigkeitsmanager
- Konformität für den Abhängigkeitsmanager

Konzepte des Abhängigkeitsmanagers

Immer dann, wenn der Wert eines Datenelements aus den Werten eines oder mehrerer anderer Datenelemente abgeleitet wird, ist der abgeleitete Wert von den Werten *abhängig*, aus denen er abgeleitet wird. Falls einer der Ausgangswerte für die Abhängigkeit anschließend geändert wird, muss das abgeleitete Datenelement neu berechnet werden, um seinen neuen Wert zu erhalten.

Beim Abhängigkeitsmanager werden diese Konzepte mit den folgenden Begriffen wiedergegeben:

- **Abhängigkeitsprodukt**

  Ein abgeleitetes Datenelement, dessen Wert aus anderen Datenelementen (Abhängigkeitsfaktoren) berechnet wird.

- **Abhängigkeitsfaktor**

  Ein Datenelement, dessen Wert zur Berechnung von abgeleiteten Datenelementen (Abhängigkeitsprodukten) verwendet werden kann.
- **Abhängigkeit**

  Ein Datensatz für die Tatsache, dass der Wert eines bestimmten Abhängigkeitsprodukts vom Wert eines bestimmten Abhängigkeitsfaktors abhängig ist.
- **Änderungselement für Abhängigkeitsfaktor**

  Ein Datensatz für die Tatsache, dass sich der Wert eines bestimmten Abhängigkeitsfaktors in gewisser Hinsicht geändert hat.
- **Änderungssatz für Abhängigkeitsfaktor**

  Ein Satz von Änderungselementen für Abhängigkeitsfaktoren, die zur Verarbeitung gruppiert wurden. Mithilfe solcher Sätze können potenziell betroffene Abhängigkeitsprodukte erkannt werden, die neu berechnet werden müssen.
- **Neuberechnung des Abhängigkeitsprodukts**

  Die Neuberechnung eines Abhängigkeitsprodukts, das potenziell von einer oder mehreren Änderungen an den Abhängigkeitsfaktoren in einem Änderungssatz für Abhängigkeitsfaktoren betroffen ist.

Diese Konzepte lassen sich am besten anhand eines Beispiels erläutern.

Ausgangspunkt ist die Berechnung der Leistungsbezugshöhe eines Anspruchstellers, die auf der Grundlage der folgenden Daten erfolgt:

- Persönliche Angaben des Anspruchstellers
- Für den Fall des Anspruchstellers zusammengestellte Angaben
- Leistungsbezugssätze und Einkommensgrenzen

Der Anspruchsteller Joe hat zwei Fälle (123 und 124). Mary, eine weitere Anspruchstellerin, hat einen Fall (125). Es gibt Fälle und persönliche Angaben für weitere Anspruchsteller sowie außerdem Freibetragssätze, die für andere Berechnungen verwendet werden.

In diesem Beispiel ist die berechnete Leistungshöhe für jeden Fall ein *Abhängigkeitsprodukt*. Die persönlichen Angaben, die fallbezogenen Angaben sowie die Sätze und Grenzen sind *Abhängigkeitsfaktoren*.

In einer *einfachen Matrix* können die Abhängigkeiten zwischen den Abhängigkeitsprodukten und den Abhängigkeitsfaktoren dargestellt werden (das Vorhandensein einer Abhängigkeit wird durch ein Kreuz kenntlich gemacht):

Tabelle 3. Beispiel für Abhängigkeitsmatrix

Abhängigkeitsfaktor	Leistungshöhe für Fall 123	Leistungshöhe für Fall 124	Leistungshöhe für Fall 125	Leistungshöhe für Fall 126
Persönliche Angaben von Joe	X	X		
Persönliche Angaben von Mary			X	
Persönliche Angaben von Frank				
Angaben für Fall 123	X			

Tabelle 3. Beispiel für Abhängigkeitsmatrix (Forts.)

Abhängigkeitsfaktor	Leistungshöhe für Fall 123	Leistungshöhe für Fall 124	Leistungshöhe für Fall 125	Leistungshöhe für Fall 126
Angaben für Fall 124		X		
Angaben für Fall 125			X	
Angaben für Fall 126				X
Leistungsbezugssätze	X	X	X	X
Einkommens-grenzen	X	X	X	X
Freibetragssätze				

Nachfolgend einige Beispiele aus der Abhängigkeitsmatrix:

- Die Leistungshöhe für den Fall 123 ist abhängig von den persönlichen Angaben von Joe, jedoch nicht von Mary.
- Die persönlichen Angaben von Joe werden für die Berechnung seiner beiden Fälle (123 und 124) verwendet.
- Alle Fälle verwenden die Leistungsbezugssätze und die Einkommensgrenzen, jedoch nicht die Freibetragssätze.
- Von den persönlichen Angaben für Frank ist keine Leistungshöhe für einen Fall abhängig.

Die Matrix kann folgendermaßen gelesen werden:

- Spaltenweise: Bei dieser Lesart werden alle Abhängigkeitsfaktoren ermittelt, von denen ein Abhängigkeitsprodukt abhängig ist. Es handelt sich hierbei um die Gruppe der Abhängigkeiten, die bei jeder Berechnung eines Abhängigkeitsprodukts erhalten bleiben muss.
- Zeilenweise: Bei dieser Lesart werden alle Abhängigkeitsprodukte ermittelt, die von einem bestimmten Abhängigkeitsfaktor abhängig sind. Es handelt sich hierbei um die Gruppe der Abhängigkeitsprodukte, die bei jeder Wertänderung des Abhängigkeitsfaktors neu berechnet werden muss.

Wenn die Anzahl der Abhängigkeitsfaktoren und Abhängigkeitsprodukte im System wächst, wird die Abhängigkeitsmatrix sehr umfangreich. Da die obige Matrix nur spärlich gefüllt ist (jedes Abhängigkeitsprodukt also nur von einem kleinen Teil der verfügbaren Abhängigkeitsfaktoren abhängig ist), werden die Daten in der Matrix nur für vorhandene Abhängigkeiten wie folgt gespeichert:

Tabelle 4. Beispiel für Abhängigkeitsspeicher

Abhängigkeitsprodukt		Abhängigkeitsfaktor
Leistungshöhe für Fall 123	ist abhängig von	Persönliche Angaben von Joe
Leistungshöhe für Fall 123	ist abhängig von	Angaben für Fall 123
Leistungshöhe für Fall 123	ist abhängig von	Leistungsbezugssätze
Leistungshöhe für Fall 123	ist abhängig von	Einkommensgrenzen
Leistungshöhe für Fall 124	ist abhängig von	Persönliche Angaben von Joe
Leistungshöhe für Fall 124	ist abhängig von	Angaben für Fall 124
Leistungshöhe für Fall 124	ist abhängig von	Leistungsbezugssätze

Tabelle 4. Beispiel für Abhängigkeitsspeicher (Forts.)

Abhängigkeitsprodukt		**Abhängigkeitsfaktor**
Leistungshöhe für Fall 124	ist abhängig von	Einkommensgrenzen
Leistungshöhe für Fall 125	ist abhängig von	Persönliche Angaben von Mary
Leistungshöhe für Fall 125	ist abhängig von	Angaben für Fall 125
Leistungshöhe für Fall 125	ist abhängig von	Leistungsbezugssätze
Leistungshöhe für Fall 125	ist abhängig von	Einkommensgrenzen
Leistungshöhe für Fall 126	ist abhängig von	Angaben für Fall 126
Leistungshöhe für Fall 126	ist abhängig von	Leistungsbezugssätze
Leistungshöhe für Fall 126	ist abhängig von	Einkommensgrenzen

(Die obige Tabelle ist nach Abhängigkeitsprodukten sortiert. Dies vereinfacht die Feststellung der Gruppe von Abhängigkeiten für jedes Abhängigkeitsprodukt. Die Tabelle könnte jedoch auch nach Abhängigkeitsfaktoren sortiert werden, was die potenziell von einer Wertänderung des entsprechenden Abhängigkeitsfaktors betroffenen Abhängigkeitsprodukte einfach erkennen lässt.)

Angenommen, die persönlichen Angaben von Joe ändern sich. Da Abhängigkeiten von den persönlichen Angaben von Joe aufgezeichnet sind, kann der Abhängigkeitsmanager erkennen, dass die Fälle 123 und 124 neu berechnet werden müssen. Bei der Neuberechnung der Fälle ändert sich der entsprechende Wert für die Leistungshöhe (aufgrund der Änderung bei den persönlichen Angaben von Joe). Bitte beachten Sie jedoch, dass sich in normalen Situationen die Abhängigkeiten selbst nicht ändern - vor der Neuberechnung ist der Fall 123 von den persönlichen Angaben von Joe, von den fallbezogen gespeicherten Angaben, von den Leistungsbezugssätzen und den Einkommensgrenzen abhängig, was auch nach der Neuberechnung zutrifft.

Es ist möglich, dass sich gleichzeitig mehrere Werte von Abhängigkeitsfaktoren ändern. Wenn die Behörde beispielsweise sowohl die Leistungsbezugssätze als auch die Einkommensgrenzen ändert, müssen alle Fälle neu berechnet werden. Im Prinzip würde jeder Fall zwei Mal angegeben werden (ein Mal aufgrund der Änderung an den Leistungsbezugssätzen und ein weiteres Mal aufgrund der Änderung an den Einkommensgrenzen). Der Abhängigkeitsmanager unterstützt jedoch die Gruppierung dieser beiden Änderungen von Abhängigkeitsfaktoren in einem einzigen Änderungssatz für Abhängigkeitsfaktoren. Wenn der Abhängigkeitsmanager den Änderungssatz für Abhängigkeitsfaktoren verarbeitet, werden alle doppelt angegebenen Abhängigkeitsprodukte automatisch herausgefiltert, damit nur die Arbeit ausgeführt wird, die zur Neuberechnung der Abhängigkeitsprodukte wirklich erforderlich ist.

Funktionen des Abhängigkeitsmanagers

Der Abhängigkeitsmanager führt die folgenden Hauptfunktionen aus:

- Er speichert Abhängigkeitsdatensätze, die von einem Client ermittelt wurden (z. B. von der Engine für Anspruchsberechtigung und Leistungshöhe bei der Berechnung einer anfänglichen Bewertungsfeststellung).
- Er erfasst Wertänderungen bei Abhängigkeitsfaktoren, die sich potenziell auf die Werte von Abhängigkeitsprodukten auswirken.
- Er erkennt die Abhängigkeitsprodukte, die potenziell von Elementen in einem Änderungssatz für Abhängigkeitsfaktoren betroffen sind.

- Er steuert die Neuberechnung dieser erkannten Abhängigkeitsprodukte.

Diese Funktionen werden in den nachfolgenden Abschnitten detaillierter beschrieben.

Speicherung von Abhängigkeitsdatensätzen

Der Abhängigkeitsmanager ist dafür zuständig, neue Abhängigkeitsdatensätze in der Datenbank zu erstellen und vorhandene Abhängigkeitsdatensätze zu entfernen, die nicht mehr benötigt werden.

Anmerkung: Ein Abhängigkeitsdatensatz enthält keine änderbaren Informationen und der Abhängigkeitsmanager nimmt in keinem Fall *Änderungen* an vorhandenen Abhängigkeitsdatensätzen vor, sondern erstellt lediglich neue Datensätze oder entfernt vorhandene Datensätze.

Immer dann, wenn ein Client des Abhängigkeitsmanagers den Wert eines Abhängigkeitsprodukts berechnet, muss der Client die für diese Berechnung verwendeten Abhängigkeitsfaktoren angeben und das Abhängigkeitsprodukt sowie die Gruppe seiner Abhängigkeitsfaktoren an den Abhängigkeitsmanager übergeben. Der Abhängigkeitsmanager verwendet dieses Abhängigkeitsprodukt, um die (gegebenenfalls) vorhandene Gruppe der gespeicherten Abhängigkeiten aus der Datenbank abzurufen, und erstellt oder entfernt Abhängigkeitsdatensätze in Übereinstimmung mit der vom Client angegebenen neuen Gruppe von Abhängigkeitsprodukten.

In der Regel erstellt der Abhängigkeitsmanager beim erstmaligen Aufruf für ein Abhängigkeitsprodukt mehrere neue Zeilen in der Datenbank, um die Abhängigkeiten von den angegebenen Abhängigkeitsfaktoren zu speichern.

Bei nachfolgenden Aufrufen des Abhängigkeitsmanagers für dasselbe Abhängigkeitsprodukt stellt der Abhängigkeitsmanager jedoch sehr häufig fest, dass die neu übergebene Gruppe von Abhängigkeiten exakt mit den Abhängigkeiten identisch ist, die bereits in der Datenbank gespeichert ist. In solchen Fällen muss der Abhängigkeitsmanager keine Schreiboperationen für die Datenbank ausführen. Gelegentlich stellt der Abhängigkeitsmanager fest, dass einige wenige neue Abhängigkeitszeilen benötigt werden und/oder einige wenige vorhandene Abhängigkeitszeilen jetzt irrelevant sind und entfernt werden müssen. In diesen Fällen führt der Abhängigkeitsmanager eine geringe Anzahl von Schreiboperationen für die Datenbank aus, um die gespeicherten Zeilen auf den aktuellen Stand der erforderlichen Abhängigkeiten zu bringen, wobei der Großteil der Abhängigkeitsdatensätze für das Abhängigkeitsprodukt nicht geändert wird.

Clients des Abhängigkeitsmanagers können angeben, dass Abhängigkeitsdatensätze für ein Abhängigkeitsprodukt nicht mehr benötigt werden, und den Abhängigkeitsmanager anweisen, alle Abhängigkeitsdatensätze für dieses Abhängigkeitsprodukt zu entfernen.

Beispiel: Bei der erstmaligen Feststellung der Leistungshöhe für einen Fall speichert der Abhängigkeitsmanager neue Abhängigkeitsdatensätze, um darzustellen, dass die Leistungshöhe des Falls von den persönlichen Angaben des Anspruchstellers, von den fallbezogen aufgezeichneten Daten, den Sätzen usw. abhängig ist.

Falls der Fall anschließend (entweder aufgrund einer Änderung der persönlichen Angaben vom Abhängigkeitsmanager automatisch oder aber wegen einer manuellen Anforderung durch einen Benutzer) neu berechnet wird, vergleicht der Abhängigkeitsmanager nach der Neuberechnung die Abhängigkeiten, die während der

Berechnung erkannt wurden, mit den bereits in der Datenbank gespeicherten Abhängigkeiten und stellt keinen Unterschied fest.

Wird zum Fall ein neues Haushaltsmitglied hinzugefügt, wird bei der Neuberechnung der Leistungshöhe eine neue Abhängigkeit erkannt. Die Leistungshöhe des Falls ist jetzt nämlich neben den bereits für den Fall gespeicherten Abhängigkeiten auch von den persönlichen Angaben des neuen Haushaltsmitglieds abhängig. Der Abhängigkeitsmanager erstellt einen neuen Abhängigkeitsdatensatz in der Datenbank, um die zusätzliche Abhängigkeit zu speichern.

Falls das neue Mitglied des Haushalts später entfernt wird, liegt bei der Neuberechnung der Leistungshöhe keine Abhängigkeit von den persönlichen Angaben des entfernten Haushaltsmitgliedes vor. Der Abhängigkeitsmanager stellt fest, dass die gespeicherte Abhängigkeit für die persönlichen Angaben des Haushaltmitglieds jetzt irrelevant ist, und entfernt diese aus der Datenbank. Dabei bleiben andere Abhängigkeitsdatensätze (zu den persönlichen Angaben des Anspruchstellers, den fallbezogenen aufgezeichneten Daten, den Sätzen usw.) erhalten.

Wird der Fall zu einem späteren Zeitpunkt abgeschlossen, ist eine Unterstützung von Neuberechnungen nicht mehr erforderlich und die Abhängigkeitsdatensätze werden nicht weiter benötigt.

Anmerkung: Vorausgesetzt wird hier die Neubewertungsstrategie "Abgeschlossene Fälle nicht neu bewerten". Weitere Informationen finden Sie im Handbuch *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.
Im Zuge der Systemverwaltung wird der Abhängigkeitsmanager aufgerufen, um alle Abhängigkeitsdatensätze für die Leistungshöhe des Falls zu entfernen. Wird der Fall später erneut geöffnet, kann seine Leistungshöhe neu berechnet werden. Der Abhängigkeitsmanager erstellt dann alle erforderlichen Abhängigkeitsdatensätze erneut.

Keine Kenntnis von Abhängigkeitsprodukten oder Abhängigkeitsfaktoren: Der Abhängigkeitsmanager verwaltet absichtlich *keine* Datensätze aller bekannten Abhängigkeitsprodukte oder Abhängigkeitsfaktoren im System, da dies Folgendes verursachen würde:

- Doppelt vorhandene Daten im System
- Engpässe bei der Verarbeitung, die möglicherweise zu Problemen beim gemeinsamen Zugriff führen, wenn das System feststellt, dass neue Daten in Berechnungen als Abhängigkeitsfaktoren verwendet werden

Der Abhängigkeitsmanager zeichnet vielmehr nur Datensatzinformationen zu Abhängigkeiten auf. Jede Abhängigkeit ist lediglich eine Verknüpfung zwischen einem bestimmten Abhängigkeitsprodukt und einem bestimmten Abhängigkeitsfaktor. Falls ein bestimmtes Abhängigkeitsprodukt keine Abhängigkeitsfaktoren hat (oder umgekehrt), werden dafür einfach keine Abhängigkeitsdatensätze gespeichert.

Der Abhängigkeitsmanager "versteht" die in einem Abhängigkeitsdatensatz gespeicherten Informationen zum Abhängigkeitsprodukt und zum Abhängigkeitsfaktor nicht. Für jeden Abhängigkeitsprodukt- und Abhängigkeitsfaktortyp ist vielmehr ein "Handler" beim Abhängigkeitsmanager registriert. Der Abhängigkeitsmanager ruft diese Handler auf, um die für den Typ geeignete geschäftsspezifische Verarbeitung auszuführen, z. B. einen Abhängigkeitsfaktor oder ein Abhängigkeitsprodukt in eine verständliche Beschreibung zu decodieren oder ein Abhängigkeitsprodukt bei Bedarf neu zu berechnen.

Abhängigkeitsspeicher ist optional: Es ist ein wichtiger Aspekt, dass die Verwendung des Abhängigkeitsmanagers zum Speichern von Abhängigkeitsdatensätzen optional ist.

Clients des Abhängigkeitsmanagers können auswählen, ob Abhängigkeitsdatensätze erforderlich sind oder nicht, ob der Client also die Funktionalität des Abhängigkeitsmanagers für die automatische Erkennung und Neuberechnung von Abhängigkeitsprodukten benötigt oder nicht.

Die Engine für die Anspruchsberechtigung und Leistungshöhe verwendet beispielsweise den Abhängigkeitsmanager, um Abhängigkeitsdatensätze für Feststellungen von Fallbewertungen zu speichern (d. h. für Feststellungen, die normalerweise zu Zahlungen und/oder Rechnungsstellungen führen). Die Engine für Anspruchsberechtigung und Leistungshöhe muss vom Abhängigkeitsmanager benachrichtigt werden, wenn ein Fall neu bewertet werden muss. Aus diesem Grund müssen die Abhängigkeitsdatensätze gespeichert werden.

Die Engine für Anspruchsberechtigung und Leistungshöhe enthält jedoch auch eine Funktion, mit der ein Fallbearbeiter die Anspruchsberechtigung und Leistungshöhe eines Falls aufgrund von Angaben in Bearbeitung manuell überprüfen kann. Diese manuellen Berechnungen der Anspruchsberechtigung und Leistungshöhe verwenden dieselben Berechnungsmethoden, benötigen jedoch keinen Abhängigkeitsspeicher, da das System solche Feststellungen in keinem Fall neu berechnen muss. Manuelle Berechnungen der Anspruchsberechtigung und Leistungshöhe werden vielmehr immer durch eine explizite Anforderung eines Fallbearbeiters ausgelöst.

Granularität von Abhängigkeiten: Die im vorherigen Beispiel verwendeten Datenelemente der Abhängigkeitsfaktoren sind übrigens absichtlich vage. Der Begriff "persönliche Angaben" würde aller Wahrscheinlichkeit nach viele einzelne Felder wie Geburtsdatum, demografische Angaben usw. abdecken. Der Abhängigkeitsmanager kennt weder die Bedeutungen der Abhängigkeiten, die er zwischen Abhängigkeitsprodukten und Abhängigkeitsfaktoren speichert, noch berücksichtigt er sie. Zuständig für die Zuordnung von Bedeutungen zu diesen Abhängigkeiten und für die Speicherung der Abhängigkeiten mit einer angemessenen Granularität sind die Clients des Abhängigkeitsmanagers.

Bei der Auswahl der Granularität muss ein annehmbarer Kompromiss zwischen den beiden folgenden Extremwerten gefunden werden:

- **Sehr feine Granularität**

  Bei dieser Granularität werden sehr präzise Abhängigkeiten zwischen Abhängigkeitsprodukten und einzelnen Datenfeldern gespeichert, was eine extrem treffsichere Erkennung der von geänderten Abhängigkeitsfaktoren betroffenen Abhängigkeitsprodukte ermöglicht, jedoch mit dem Nachteil verbunden ist, dass viele Abhängigkeitsdatensätze gespeichert werden.
- **Sehr grobe Granularität**

  Bei dieser Granularität werden sehr allgemein gehaltene Abhängigkeiten zwischen Abhängigkeitsprodukten und Gruppierungen vieler einzelner Datenfelder in einem einzigen Datenelement gespeichert. Dies führt dazu, dass zwar nur wenige Abhängigkeitsdatensätze gespeichert werden, jedoch möglicherweise unnötige Neuberechnungen angefordert werden (also Neuberechnungen, die sich als überflüssig herausstellen, weil die Berechnung nicht durch das jeweils geänderte Datenfeld beeinflusst wird).

Dieser Kompromiss muss von den Entwicklern der Clients des Abhängigkeitsmanagers berücksichtigt werden. Ziel ist eine sensible Auswahl der Stufe, mit der Abhängigkeitsinformationen im Abhängigkeitsmanager gespeichert werden.

Als Beispielfall sollen die folgenden persönlichen Angaben dienen, die das System für einen Anspruchsteller aufzeichnet (in einem realen System werden möglicherweise viel mehr Felder als "persönliche Angaben" betrachtet):

- Geburtsdatum (wird bei Berechnungen der Leistungshöhe verwendet)
- Anzahl der Kinder (wird bei Berechnungen der Leistungshöhe verwendet)
- Geburtsname der Mutter (dient als Antwort auf eine Sicherheitsfrage und wird nur verwendet, um die Identität des Anspruchsteller zu bestätigen, jedoch nicht, um die Leistungshöhe zu berechnen)

Bei einer sehr differenzierten Gruppe von Abhängigkeiten würde die Leistungshöhe des Falls vom Geburtsdatum und von der Anzahl der Kinder, jedoch nicht vom Geburtsnamen der Mutter abhängig sein (da bei den Berechnungen auf diesen nicht zugegriffen wird):

Tabelle 5. Beispiel für differenzierte Abhängigkeitsmatrix

Abhängigkeitsfaktor	**Leistungshöhe für Fall 127**
Geburtsdatum von Frank	X
Anzahl der Kinder von Frank	X
Geburtsname der Mutter von Frank	

Dieser differenzierte Abhängigkeitsspeicher könnte dazu führen, dass viele Zeilen gespeichert werden müssen. Eine Neuberechnung der Leistungshöhe für den Fall wird jedoch nur von Änderungen am Geburtsdatum und/oder an der Anzahl der Kinder ausgelöst (falls ein Schreibfehler im Geburtsnamen der Mutter korrigiert wird, wird keine Neuberechnung der Leistungshöhe für den Fall ausgelöst).

Eine sehr allgemein definierte Gruppe von Abhängigkeiten würde im Gegensatz einen viel einfacheren Datensatz ergeben, da die Leistungshöhe des Falls von allen persönlichen Angaben abhängig ist:

Tabelle 6. Beispiel für allgemein definierte Abhängigkeitsmatrix

Abhängigkeitsfaktor	**Leistungshöhe für Fall 127**
Persönliche Angaben von Frank	X

Bei diesem allgemein definierten Abhängigkeitsspeicher werden weniger Abhängigkeitsdatensätze gespeichert, aber falls ein Schreibfehler im Geburtsnamen der Mutter korrigiert wird, ändert sich die Gesamtheit der persönlichen Angaben und eine Neuberechnung der Leistungshöhe für den Fall wird ausgelöst, obwohl die Neuberechnung ergibt, dass sich das Berechnungsergebnis nicht geändert hat.

Erfassung von Änderungselementen für Abhängigkeitsfaktoren

Clients müssen den Abhängigkeitsmanager benachrichtigen, wenn sich der Wert eines Abhängigkeitsfaktors geändert hat. Der Abhängigkeitsmanager kumuliert diese Änderungen für die spätere Verarbeitung in einem Änderungssatz für Abhängigkeitsfaktoren.

Das am weitesten verbreitete Beispiel ist die Engine für Anspruchsberechtigung und Leistungshöhe, die so genannte "Regelobjektpropagatoren" enthält. Diese Pro-

pagatoren haben die Aufgabe, Änderungen an Einheiten und Angaben zu überwachen und den Abhängigkeitsmanager über diese Änderungen zu benachrichtigen.

Der Abhängigkeitsmanager unterstützt für die Verarbeitung von Änderungen bei Abhängigkeitsfaktoren die folgenden Modi:

- Warteschlangeneinreihung für die zurückgestellte Verarbeitung (Standardeinstellung)
- Warteschlangeneinreihung für die Batchverarbeitung (wird von Clients verwendet, von denen Änderungen bei Abhängigkeitsfaktoren erkannt wurden, die wahrscheinlich zu einer großen Anzahl von neu berechneten Abhängigkeitsprodukten führen)

Diese Modi werden in den nachfolgenden Abschnitten detaillierter beschrieben.

Warteschlangeneinreihung für die zurückgestellte Verarbeitung: Dies ist der Standardmodus für die Verarbeitung von geänderten Abhängigkeitsfaktoren.

Falls während des Geltungsbereichs einer Datenbanktransaktion Änderungselemente für Abhängigkeitsfaktoren festgestellt werden, führt der Abhängigkeitsmanager Folgendes aus:

- Er erstellt einen einzigen neuen Änderungssatz für Abhängigkeitsfaktoren in der Datenbank.
- Er fügt alle Änderungselemente für Abhängigkeitsfaktoren, die während der Transaktion erkannt wurden, zu diesem neuen Änderungssatz für Abhängigkeitsfaktoren hinzu.
- Er stellt eine Anforderung für die zurückgestellte Verarbeitung dieses neuen Änderungssatzes für Abhängigkeitsfaktoren in die Warteschlange.

Warteschlangeneinreihung für die Batchverarbeitung: Dies ist ein besonderer Modus, der nur von Clients verwendet wird, die geänderte Werte von Abhängigkeitsfaktoren festgestellt haben, von denen wahrscheinlich die Neuberechnung vieler Abhängigkeitsprodukte ausgelöst werden. Der Abhängigkeitsmanager verwaltet einen speziellen systemweiten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung, in der Änderungen von Abhängigkeitsfaktoren über (möglicherweise) viele verschiedene Transaktionen hinweg kumuliert werden.

Falls während des Geltungsbereichs einer Datenbanktransaktion Änderungselemente für Abhängigkeitsfaktoren festgestellt werden, führt der Abhängigkeitsmanager Folgendes aus:

- Er ruft den speziellen systemweiten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung auf.
- Er fügt alle Änderungselemente für Abhängigkeitsfaktoren, die während der Transaktion erkannt wurden, zu diesem Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung hinzu.
- Er schreibt eine Informationsnachricht an die Anwendungsprotokolle, um einen Administrator aufzufordern, die Batchverarbeitung für diesen Änderungssatz für Abhängigkeitsfaktoren zu planen.

Beispiel: Nachdem Joes persönliche Angaben im System aktualisiert wurden, benachrichtigt der Regelobjektpropagator den Abhängigkeitsmanager über die Änderung. Der Abhängigkeitsmanager schreibt die Änderung des Abhängigkeitsfaktors in einen neuen Änderungssatz für Abhängigkeitsfaktoren und stellt einen Prozess

für die zurückgestellte Verarbeitung in die Warteschlange. Die zurückgestellte Verarbeitung erkennt, dass die beiden Fälle von Joe potenziell betroffen sind und bewertet sie neu.

Später veröffentlicht ein Administrator einige Änderungen an CER-Regelwerken. Der Abhängigkeitsmanager zeichnet diese Änderungen an CER-Regelwerken auf, indem zum systemweiten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung ein Datensatz für ein Änderungselement für Abhängigkeitsfaktoren hinzugefügt wird. Der Administrator veröffentlicht darüber hinaus einige Änderungen an Sätzen. Der Abhängigkeitsmanager zeichnet diese Änderungen an Sätzen auf, indem ein weiterer Datensatz für ein Änderungselement für Abhängigkeitsfaktoren zum systemweiten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung hinzugefügt wird. Der Administrator ordnet die Ausführung der Batch-Suite des Abhängigkeitsmanagers an, damit die betroffenen Fälle erkannt und neu bewertet werden.

Lebenszyklus eines Änderungssatzes für Abhängigkeitsfaktoren: Jeder Änderungssatz für Abhängigkeitsfaktoren durchläuft den folgenden einfachen Lebenszyklus:

- **Geöffnet**

  Diesen Status weist ein Änderungssatz für Abhängigkeitsfaktoren auf, wenn er anfänglich erstellt wird. In diesem Status können neue Änderungselemente für Abhängigkeitsfaktoren zum Änderungssatz für Abhängigkeitsfaktoren hinzugefügt werden.
- **Übermittelt**

  Der Änderungssatz für Abhängigkeitsfaktoren wurde an die Verarbeitung für die Erkennung von Abhängigkeitsprodukten übermittelt. Weitere Änderungselemente für Abhängigkeitsfaktoren können nicht zum Änderungssatz für Abhängigkeitsfaktoren hinzugefügt werden.
- **Vollständig**

  Die Neuberechnung aller Abhängigkeitsprodukte, die von den Änderungen der Abhängigkeitsfaktoren betroffen sind, ist abgeschlossen. Der Änderungssatz für Abhängigkeitsfaktoren wird lediglich aufbewahrt, um den Verlauf zu dokumentieren.

Die Übergänge zwischen den einzelnen Statuswerten finden abhängig von dem Modus, in dem die Änderungen der Abhängigkeitsfaktoren erfasst wurden, unterschiedlich statt:

- Warteschlangeneinreihung für die zurückgestellte Verarbeitung:
  - Die Transaktion für die Erfassung von geänderten Abhängigkeitsfaktoren *öffnet* einen neuen Änderungssatz für Abhängigkeitsfaktoren und *übermittelt* ihn durch die Anforderung eines zurückgestellten Prozesses.
  - Der zurückgestellte Prozess akzeptiert den *übermittelten* Änderungssatz für Abhängigkeitsfaktoren, erkennt betroffene Abhängigkeitsprodukte, berechnet sie neu und *schließt* den Änderungssatz für Abhängigkeitsfaktoren *ab*.
- Warteschlangeneinreihung für die Batchverarbeitung:
  - Die Transaktion für die Erfassung von geänderten Abhängigkeitsfaktoren schreibt die Änderungen in den gegenwärtig *geöffneten* Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung.
  - Die Suite der Batchprozesse des Abhängigkeitsmanagers führt die folgenden Schritte aus:
    - Sie ruft den gegenwärtig *geöffneten* Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung auf, *übergibt* ihn an den nächsten Schritt im

Batchprozess und erstellt einen neuen *geöffneten* Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung, um alle weiteren vorhergehenden Änderungssätze, die für die Batchverarbeitung ermittelt wurden, zu erfassen.

Anmerkung: In einem aktiven System wird auf diese Weise ein neuer Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung erstellt (d. h., durch die Übergabe des vorhergehenden Änderungssatzes). Der *anfänglich* geöffnete Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung wird durch eine in der Anwendung enthaltene DMX-Datei bereitgestellt.

- Die Suite führt die im Datenstrom übertragene Batchverarbeitung aus, um die Abhängigkeitsprodukte zu ermitteln und neu zu berechnen, die von den Änderungen im jetzt *übermittelten* Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung betroffen sind.
- Danach *schließt* sie den Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung *ab*.

Ermittlung potenziell betroffener Abhängigkeitsprodukte

Sobald der Abhängigkeitsmanager eines oder mehrere Änderungselemente für Abhängigkeitsfaktoren erfasst und in einem Änderungssatz für Abhängigkeitsfaktoren gruppiert hat, kann er ermitteln, welche Abhängigkeitsprodukte potenziell von einem oder mehreren dieser Änderungselemente für Abhängigkeitsfaktoren betroffen sind. Hierzu werden die gespeicherten Abhängigkeitsdatensätze für jeden Abhängigkeitsfaktor im Änderungssatz für Abhängigkeitsfaktoren untersucht. An dieser Stelle filtert der Abhängigkeitsmanager alle Abhängigkeitsprodukte heraus, die mehrfach angegeben sind.

Diese Ermittlung der potenziell betroffenen Abhängigkeitsprodukte findet entweder in einer zurückgestellten Verarbeitung oder in der Batchverarbeitung statt. Dies ist von dem Modus abhängig, der bei der Erfassung der Änderungselemente für Abhängigkeitsfaktoren gültig war (siehe „Erfassung von Änderungselementen für Abhängigkeitsfaktoren" auf Seite 88).

Falls beispielsweise eine einzelne Datenbanktransaktion Änderungen in mehrere Datenbankzeilen schreibt, erkennt der Schritt für die zurückgestellte Verarbeitung jeden betroffenen Fall nur ein einziges Mal, auch wenn jede geänderte Datenbankzeile wiederum selbst einen bestimmten Fall betrifft.

Falls der Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung Änderungen sowohl an CER-Regelwerken als auch an Sätzen enthält, erkennt der Schritt für die Batchverarbeitung analog jeden betroffenen Fall nur ein einziges Mal, auch wenn die Änderung des CER-Regelwerks - wie auch die Satzänderung - selbst wiederum einen bestimmten Fall betrifft.

Neuberechnung von erkannten Abhängigkeitsprodukten

Nachdem der Abhängigkeitsmanager alle Abhängigkeitsprodukte erkannt hat, die potenziell von geänderten Abhängigkeitsfaktoren betroffen sind, fordert er für jedes dieser Abhängigkeitsprodukte eine Neuberechnung an. Der Abhängigkeitsmanager hat keine Kenntnis davon, was durch die einzelnen Abhängigkeitsprodukte dargestellt wird. Die geeignete Neuberechnung wird daher dadurch erreicht, dass der Abhängigkeitsmanager für jedes Abhängigkeitsprodukt nach einem registrierten Handler sucht und die Zuständigkeit für die Neuberechnung an den Handler delegiert.

Der registrierte Handler für Abhängigkeitsprodukte bei Feststellungen der Fallbewertung "versteht" beispielsweise, dass die geeignete Neuberechnung für einen Fall in der Neubewertung des Falles besteht.

Die Neuberechnung eines Abhängigkeitsprodukts findet entweder in einer zurückgestellten Verarbeitung oder in der Batchverarbeitung statt. Dies ist von dem Modus abhängig, der bei der Erfassung der Änderungselemente für Abhängigkeitsfaktoren gültig war (siehe „Erfassung von Änderungselementen für Abhängigkeitsfaktoren" auf Seite 88). Sobald dieser Schritt abgeschlossen ist, ist das System hinsichtlich der erfassten Änderungen von Abhängigkeitsfaktoren auf dem aktuellen Stand.

Anmerkung: Da jeder Handler für Abhängigkeitsprodukte von Online-Transaktionen, verzögerten Transaktionen oder Stapeltransaktionen aufgerufen werden kann, ist es wichtig, dass *keiner* dieser Handler Annahmen zum verwendeten Transaktionstyp macht.

Zurückgestellte Verarbeitung des Abhängigkeitsmanagers

Im vorangegangenen Abschnitt („Funktionen des Abhängigkeitsmanagers" auf Seite 84) wurde beschrieben, wie der Abhängigkeitsmanager die Erfassung von Änderungselementen für Abhängigkeitsfaktoren in einem systemweiten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung unterstützt und mithilfe der zurückgestellten Verarbeitung potenziell betroffene Abhängigkeitsprodukte erkennt und neu berechnet.

Im vorliegenden Abschnitt soll nun die zurückgestellte Verarbeitung des Abhängigkeitsmanagers detaillierter beschrieben werden.

Die zurückgestellte Verarbeitung berechnet vorab zunächst die Gesamtsumme der potenziell betroffenen Abhängigkeitsprodukte und somit die Anzahl der auszuführenden Neuberechnungen. Anschließend trifft sie eine der beiden folgenden Entscheidungen:

- Falls diese Berechnung unterhalb des Systemgrenzwerts liegt, werden die Neuberechnungen für den Änderungssatz für Abhängigkeitsfaktoren durch die zurückgestellte Verarbeitung sofort ausgeführt.
- Falls diese Berechnung einen Systemgrenzwert überschreitet, werden die Berechnungen des Änderungssatzes für Abhängigkeitsfaktoren an die Batchverarbeitung des Abhängigkeitsmanagers übergeben (siehe „Batchverarbeitung des Abhängigkeitsmanagers" auf Seite 93). In diesem Moment wird der Änderungssatz für Abhängigkeitsfaktoren mit dem Status "Verzögerte Verarbeitung zu Batchverarbeitung" gekennzeichnet.

Systemgrenzwert für zurückgestellte Verarbeitung festlegen

Der Systemgrenzwert für die zurückgestellte Verarbeitung kann über die Anwendungseigenschaft `curam.dependency.deferred.processing.limit` festgelegt werden.

Für diese Eigenschaft ist zwar standardmäßig der Wert `50` festgelegt, aber der geeignete Wert ist bei jedem System von einer Vielzahl Faktoren (z. B. dem verfügbaren Hauptspeicher) abhängig. Es empfiehlt sich infolgedessen, den Wert im Rahmen eines Systemtests festzulegen, mit dem ein angemessener Wert ermittelt wird, der zum einen die meisten (wenn nicht gar alle) zurückgestellten Prozesse normal verarbeiten kann, problematischere Prozesse jedoch an die Batchverarbeitung übergibt.

Fehlerbehandlung bei der zurückgestellten Verarbeitung

Falls ein Fehler auftritt, während der zurückgestellte Prozess des Abhängigkeitsmanagers ausgeführt wird, übergibt das System nach der üblichen Anzahl von Wiederholungen die Arbeit stattdessen an die Batchverarbeitung. Dies schützt das System vor Störungen durch Überschreitungen von Transaktionszeitlimits und stellt sicher, dass jeder Versuch für die Verarbeitung der Berechnungen unternommen wurde.

Der Status des Änderungssatzes für Abhängigkeitsfaktoren ändert sich in diesem Fall zudem in "Verzögerte Verarbeitung an Batchverarbeitung", um kenntlich zu machen, dass der zurückgestellte Prozess fehlerhaft war. An den Absender des zurückgestellten Prozesses wird außerdem eine entsprechende Benachrichtigung ausgegeben.

Batchverarbeitung des Abhängigkeitsmanagers

In einem vorangegangenen Abschnitt („Funktionen des Abhängigkeitsmanagers” auf Seite 84) wurde bereits beschrieben, wie der Abhängigkeitsmanager die Erfassung von Änderungselementen für Abhängigkeitsfaktoren in einem systemweiten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung unterstützt und mithilfe der Batchverarbeitung potenziell betroffene Abhängigkeitsprodukte erkennt und neu berechnet.

Die Batchverarbeitung des Abhängigkeitsmanagers soll im vorliegenden Abschnitt nun detaillierter beschrieben werden.

Der Abhängigkeitsmanager verwaltet Steuerdatensätze für die Datenbank, um auf die folgenden Änderungssätze für Abhängigkeitsfaktoren zur Batchverarbeitung zu verweisen:

- Gegenwärtig geöffneter Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung (es gibt immer genau einen Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung, der für die Aufnahme neuer Änderungselemente für Abhängigkeitsfaktoren geöffnet ist)
- Gegenwärtig von der Batch-Suite des Abhängigkeitsmanagers verarbeiteter Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung (sofern vorhanden - dieser Satz wird nur während der Batchverarbeitung gefüllt; meistens gibt es keinen Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung, der sich in diesem Status befindet)

Diese Steuerdatensätze sind für das Verhalten der Batch-Suite des Abhängigkeitsmanagers von fundamentaler Bedeutung.

Immer dann, wenn sich Änderungen von Abhängigkeitsfaktoren im Modus "Warteschlangeneinreihung für die Batchverarbeitung" befinden, enthalten die Anwendungsprotokolle eine Nachricht, die den Administrator darüber benachrichtigt, dass die Batch-Suite des Abhängigkeitsmanagers ausgeführt werden muss. Für den Benutzer, von dem die Änderungen vorgenommen wurden, die zur Batchverarbeitung in die Warteschlange eingereiht wurden, wird in der Anzeige ebenfalls eine Informationsnachricht ausgegeben, dass die Batch-Suite des Abhängigkeitsmanagers ausgeführt werden muss.

Die Batch-Suite des Abhängigkeitsmanagers besteht aus den folgenden separaten Batchprozessen:

- **Änderungssatz für Abhängigkeitsfaktoren übermitteln**

Dies ist der Ausgangspunkt der Batch-Suite. Dieser einfache Einzeldatenstromprozess übermittelt den gegenwärtig geöffneten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung.

- **Neuberechnung im Batchbetrieb aus Änderungssatz für Abhängigkeitsfaktoren ausführen**

  Dieser auslastungsintensive Mehrdatenstromprozess erkennt die Abhängigkeitsprodukte, die potenziell von den Änderungen im übermittelten Änderungssatz für Abhängigkeitsfaktoren betroffen sind, und berechnet sie neu. Die für die Ausführung dieses Prozesses benötigte Zeit variiert abhängig davon, wie viele Abhängigkeitsprodukte neu berechnet werden müssen, und kann beträchtlich sein.

- **Änderungssatz für Abhängigkeitsfaktoren abschließen**

  Dies ist der Endpunkt der Batch-Suite. Dieser einfache Einzeldatenstromprozess schließt den gegenwärtig geöffneten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung ab.

Die obigen Batchprozesse sind in den nachfolgenden Abschnitten detaillierter beschrieben.

Änderungssatz für Abhängigkeitsfaktoren übermitteln

Dieser Batchprozess ist der Ausgangspunkt der Batch-Suite. Es handelt sich um einen einfachen Einzeldatenstromprozess, der den gegenwärtig geöffneten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung übermittelt und einen neuen geöffneten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung erstellt, in der anschließend alle Änderungen von Abhängigkeitsfaktoren erfasst werden, die erkannt und für die Batchverarbeitung in die Warteschlange gestellt wurden.

Zur Ausführung dieses Batchprozesses führen Sie den folgenden Befehl aus (der Befehl muss in einer einzigen Zeile eingegeben werden):

```
build runbatch -Dbatch.program=

curam.dependency.intf.SubmitPrecedentChangeSet.process

-Dbatch.username=SYSTEM
```

Falls dieser Batchprozess erfolgreich abgeschlossen wird, gibt er in einer einfachen Nachricht die Bestätigung aus, dass der geöffnete Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung übermittelt wurde.

Tipp: Häufig wird irrtümlicherweise versucht, diesen Batchprozess auszuführen, während sich ein anderer Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung noch im Status "Übermittelt" befindet.

Dieser Prozess gibt eine einfache Fehlernachricht aus, wenn ein anderer Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung noch nicht vollständig durch die Batch-Suite verarbeitet wurde.

Als Zusatzinformation gibt dieser Batchprozess außerdem eine Liste der Abhängigkeitsprodukttypen aus, die beim Abhängigkeitsmanager registriert sind. Diese Liste der Abhängigkeitsprodukttypen ist bei der Ausführung des nächsten Schritts wichtig (siehe „Neuberechnung im Batchbetrieb aus Änderungssatz für Abhängigkeitsfaktoren ausführen” auf Seite 95).

Die Anwendung umfasst die folgenden Abhängigkeitsprodukttypen:

- **Fallbewertung - Feststellungsergebnis**

  Dies ist die Berechnung einer Bewertungsfeststellung für einen Produktbereitstellungsfall.

  Weitere Informationen finden Sie im Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

- **Beratungskontext**

  Die Berechnung einer Beratung.

  Weitere Informationen enthält das Handbuch Cúram Advisor Configuration Guide.

- **Gespeicherter Attributwert**

  Die Berechnung eines Attributs, das in den CER-Datenbanktabellen gespeichert ist.

  Weitere Informationen finden Sie im Abschnitt „Speicherung von Abhängigkeiten für in CER gespeicherte Attributwerte durch CER mit Abhängigkeitsmanager" auf Seite 105.

Neuberechnung im Batchbetrieb aus Änderungssatz für Abhängigkeitsfaktoren ausführen

Dieser auslastungsintensive Mehrdatenstromprozess erkennt die Abhängigkeitsprodukte, die potenziell von den Änderungen im übermittelten Änderungssatz für Abhängigkeitsfaktoren betroffen sind, und berechnet sie neu. Die für die Ausführung dieses Prozesses benötigte Zeit variiert abhängig davon, wie viele Abhängigkeitsprodukte neu berechnet werden müssen, und kann beträchtlich sein.

Der Schritt "Neuberechnung im Batchbetrieb aus Änderungssatz für Abhängigkeitsfaktoren ausführen" muss mehrmals ausgeführt werden, nämlich ein Mal für jeden Abhängigkeitsprodukttyp, der beim Abhängigkeitsmanager registriert ist (siehe Informationen zur Ausgabe des vorherigen Schrittes im Abschnitt „Änderungssatz für Abhängigkeitsfaktoren übermitteln" auf Seite 94). Sie können die gewünschte Reihenfolge für die Verarbeitung der Abhängigkeitsprodukttypen frei wählen. Beispielsweise ist die Neubewertung von Fallfeststellungen wahrscheinlich in Ihrer Organisation ein kritischerer Prozess (siehe Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules) als die Ermittlung von nicht mehr aktuellen Beratungen (siehe Handbuch Advisor Configuration Guide). Darüber hinaus können Sie jederzeit die Verarbeitung unterschiedlicher Abhängigkeitsprodukttypen auf mehrere Tage verteilen. Bitte beachten Sie jedoch, dass alle weiteren Änderungselemente für Abhängigkeitsfaktoren, die zur Batchverarbeitung in die Warteschlange gestellt werden, erst dann verarbeitet werden können, wenn der gegenwärtig übermittelte Änderungssatz für Abhängigkeitsfaktoren die Batch-Suite des Abhängigkeitsmanagers vollständig durchlaufen hat.

Der Schritt "Neuberechnung im Batchbetrieb aus Änderungssatz für Abhängigkeitsfaktoren ausführen" verwendet die Batch-Streaming-Architektur von Cúram (siehe Handbuch *Cúram Batch Performance Mechanisms Guide*); die Verarbeitung ist folglich in die folgenden Phasen unterteilt:

- **Segmente erkennen**

  Diese Phase muss in einem einzigen Prozess ausgeführt werden. Sie erkennt die Abhängigkeitsprodukte (eines bestimmten Abhängigkeitsprodukttyps), die von den Änderungen im übermittelten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung potenziell betroffen sind. Die IDs der erkannten Abhängigkeitsprodukte werden zur Verarbeitung durch die nächste Phase in so genannte "Segmente" geschrieben.

- **Segmente verarbeiten**

  Diese für die gleichzeitige Ausführung durch mehrere Prozesse offene Phase verwendet ein Segment mit erkannten Abhängigkeitsprodukten als Eingabe und berechnet jedes Abhängigkeitsprodukt neu.

Zur Ausführung dieses Batchprozesses für einen Abhängigkeitsprodukttyp führen Sie den folgenden Befehl aus (der Befehl muss in einer einzigen Zeile eingegeben werden):

```
build runbatch -Dbatch.program=
curam.dependency.intf.PerformBatchRecalculationsFromPrecedentChangeSet.process
-Dbatch.username=SYSTEM
 -Dbatch.parameters="dependentType= >code_für_abhängigkeitsprodukttyp "
```

Beide Phasen werden standardmäßig in einem einzigen Prozess ausgeführt. Sie können jedoch zusätzliche Datenstromprozesse gleichzeitig auf anderen Computern ausführen, um die zweite Phase parallel auszuführen (weitere Informationen zur Parallelverarbeitung und zu den Umgebungsvariablen, die das Parallelverarbeitungsverhalten dieses Prozesses für "Neuberechnung im Batchbetrieb vom vorhergehenden Änderungssatz ausführen" steuern, finden Sie im Handbuch *Cúram Batch Performance Mechanisms Guide*). Zur Ausführung eines "Streaming"-Prozesses für einen Abhängigkeitsprodukttyp setzen Sie den folgenden Befehl ab (der Befehl muss in einer einzigen Zeile eingegeben werden):

```
build runbatch -Dbatch.program=
curam.dependency.intf.PerformBatchRecalculationsFromPrecedentChangeSetStream.process
-Dbatch.username=SYSTEM
 -Dbatch.parameters="dependentType= code_für_abhängigkeitsprodukttyp "
```

Das Starten des Batchprozesses schlägt mit einem nicht behebbaren Fehler fehl, wenn eine der folgenden Bedingungen eintritt:

- Der Abhängigkeitsprodukttyp (Parameter "dependentType") wird nicht angegeben oder enthält nicht den Code für einen beim Abhängigkeitsmanager registrierten Abhängigkeitsprodukttyp.
- Es befindet sich kein Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung im Status "Übermittelt" (d. h., der Prozess 'Vorhergehenden Änderungssatz übermitteln' wurde noch nicht seit der letzten Ausführung von 'Vorhergehenden Änderungssatz abschließen' ausgeführt).

Andernfalls wird der Batchprozess gestartet und versucht, die betroffenen Abhängigkeitsprodukte zu erkennen und neu zu berechnen. Der Versuch, ein bestimmtes Abhängigkeitsprodukt neu zu berechnen, hat eines der folgenden Ergebnisse:

- **Erfolg**

  Das Abhängigkeitsprodukt wurde gefunden und ordnungsgemäß neu berechnet. Die Verarbeitung wird normal fortgesetzt.

- **Nicht gefunden**

  Das Abhängigkeitsprodukt wurde nicht gefunden und konnte daher nicht verarbeitet werden. Diese Situation kann eintreten, wenn ein Client des Abhängigkeitsmanagers beschließt, dass ein Abhängigkeitsprodukt nicht mehr vorhanden sein solle, jedoch den Abhängigkeitsmanager nicht angewiesen hat, die Abhängigkeitsdatensätze für dieses Abhängigkeitsprodukt zu entfernen. Unter diesen Umständen entfernt der Abhängigkeitsmanager automatisch die irrelevanten Abhängigkeitsdatensätze und schreibt eine Warnung in das Anwendungsprotokoll bzw. die Batchdatenstromausgabe.

- **Fehler**

Während der Neuberechnung des Abhängigkeitsprodukts wurde eine Ausnahmebedingung ausgelöst (falls eine CER-Berechnung beispielsweise ein Problem aufgrund einer Division durch 0 festgestellt hat). Die ausgelöste Ausnahmebedingung wird in die Batchdatenstromausgabe geschrieben und die Wiederherstellung wird durch die "Übersprungsverarbeitung" der Batch-Streaming-Architektur von Cúram abgewickelt.

Wenn der Prozess "Neuberechnung im Batchbetrieb vom vorhergehenden Änderungssatz ausführen" vollständig ausgeführt wurde, wird ein umfassender Bericht geschrieben, der Details darüber enthält, wie viele Abhängigkeitsprodukte erfolgreich verarbeitet wurden, wie viele Abhängigkeitsprodukte nicht gefunden wurden und bei wie vielen Abhängigkeitsprodukten Fehler festgestellt wurden. Falls Fehler aufgetreten sind, ermitteln Sie die Details über die Fehler in den Ausgabeprotokollen der Batchdatenströme.

Wichtig: Wenn Sie die Cúram-Standardprotokollstufe mit "verbose" (= ausführlich) oder höher angeben, gibt der Abhängigkeitsmanager vor der Neubewertung der einzelnen Abhängigkeitsprodukte Folgendes aus:

- Verständliche Beschreibung des Abhängigkeitsprodukts
- Subset der Änderungen von Abhängigkeitsfaktoren (aus dem Änderungssatz für Abhängigkeitsfaktoren), die die Erkennung des Abhängigkeitsprodukts verursachten

Diese Protokollstufe wird nur für Entwicklungsumgebungen empfohlen. Die ausführliche Protokollierung kann das Leistungsverhalten und die Skalierbarkeit in einem Produktionssystem beeinträchtigen.

Diese Ausgabe kann hilfreich sein, wenn Sie ermitteln wollen, warum für ein bestimmtes Abhängigkeitsprodukt eine erforderliche Neuberechnung festgestellt wurde.

Anmerkung: Falls Sie diesen Batchprozess für denselben Abhängigkeitsprodukttyp versehentlich mehrmals ausgeführt haben, findet zwar eine Neuberechnung der Abhängigkeitsprodukte statt, es wird jedoch festgestellt, dass das Abhängigkeitsprodukt bereits auf dem aktuellen Stand ist.

Diese versehentliche zusätzliche Ausführung für einen Abhängigkeitsprodukttyp stellt an sich zwar kein Problem für das System dar, verwendet jedoch wertvolle Bearbeitungszeit.

Lesen Sie sich die Liste der Abhängigkeitsprodukttypen sorgfältig durch und achten Sie darauf, welche Abhängigkeitsprodukttypen verarbeitet wurden bzw. welche Typen noch verarbeitet werden müssen.

Wichtig: To determine the number of cases that will be affected by the batch suite, the following SQL can be used, where the 'DependentType' SQL value is the dependentType value that will be passed to the 'Perform Batch Recalculations From Precedent Change Set' batch process:

```
SELECT DISTINCT CaseID
FROM CaseHeader, PrecedentChangeItem, PrecedentChangeSet, Dependency
WHERE Dependency.DependentID = CaseHeader.CaseID AND
Dependency.PrecedentType = PrecedentChangeItem.PrecedentType AND
Dependency.PrecedentID = PrecedentChangeItem.PrecedentID AND
PrecedentChangeItem.PrecedentChangeSetID = PrecedentChangeSet.PrecedentChangeSetID AND
PrecedentChangeSet.Status = 'OPEN' AND
Dependency.DependentType = 'CADETERRES';
```

Änderungssatz für Abhängigkeitsfaktoren abschließen

Dieser einfache Einzeldatenstromprozess schließt den gegenwärtig geöffneten Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung ab.

Wichtig: Führen Sie diesen Batchprozess erst dann aus, wenn Sie sicher sind, dass der vorherige Schritt („Neuberechnung im Batchbetrieb aus Änderungssatz für Abhängigkeitsfaktoren ausführen" auf Seite 95) *für jeden beim Abhängigkeitsmanager registrierten Abhängigkeitsprodukttyp* abgeschlossen wurde.

Zur Ausführung dieses Batchprozesses führen Sie den folgenden Befehl aus (der Befehl muss in einer einzigen Zeile eingegeben werden):

```
build runbatch -Dbatch.program=

curam.dependency.intf.CompletePrecedentChangeSet.process

-Dbatch.username=SYSTEM
```

Falls dieser Batchprozess erfolgreich abgeschlossen wird, gibt er in einer einfachen Nachricht die Bestätigung aus, dass der übermittelte Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung abgeschlossen wurde.

Tipp: Häufig wird irrtümlicherweise versucht, diesen Batchprozess auszuführen, bevor der Prozess "Änderungssatz für Abhängigkeitsfaktor übermitteln" ausgeführt wurde.

Dieser Prozess gibt eine einfache Fehlernachricht aus, wenn sich kein Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung im Status "Übermittelt" befindet.

Nachdem der Änderungssatz für Abhängigkeitsfaktoren zur Batchverarbeitung vollständig ausgeführt wurde, überprüft die Verarbeitung, ob seit dem Beginn der Batch-Suite neue Änderungen von Abhängigkeitsfaktoren für die Batchverarbeitung in die Warteschlange gestellt wurden. Diese Situation kann in den folgenden Fällen eintreten:

- Die Neuberechnung eines Abhängigkeitsprodukts während der Batchausführung führte zu Änderungen an Daten, die auch als Abhängigkeitsfaktoren verwendet werden.
- Das Onlinesystem wurde für die Batch-Suite ausgeführt, während ein Benutzer gleichzeitig Änderungen vorgenommen hat, die dazu führten, dass Änderungselemente für Abhängigkeitsfaktoren zur Batchverarbeitung in die Warteschlange gestellt wurden.

Die Verarbeitung gibt eine einfache Nachricht mit einer der folgenden Angaben aus:

- Es befinden sich keine weiteren Änderungselemente für Abhängigkeitsfaktoren zur Batchverarbeitung in der Warteschlange (das System ist daher hinsichtlich der Änderungselemente für Abhängigkeitsfaktoren zur Batchverarbeitung auf dem aktuellen Stand).
- Es wurden weitere Änderungselemente für Abhängigkeitsfaktoren zur Batchverarbeitung in die Warteschlange gestellt und eine weitere Ausführung der Batch-Suite des Abhängigkeitsmanagers ist erforderlich, um diese zusätzlichen Elemente zu verarbeiten. In dieser Situation müssen Sie entscheiden, ob die zusätzliche Ausführung sofort erfolgen oder auf einen späteren Zeitpunkt verschoben wer-

den soll (wenn möglicherweise noch mehr Änderungselemente für Abhängigkeitsfaktoren zur Batchverarbeitung in die Warteschlange gestellt wurden).

Batchtools des Abhängigkeitsmanagers

Informationen zu den Tools, die für die Ausführung der Batch-Suite des Abhängigkeitsmanagers bereitgestellt werden bereitgestellt werden, finden Sie im Abschnitt 'Batchtools des Abhängigkeitsmanagers' im 'Cúram Operations Guide' .

Integration zwischen CER und Abhängigkeitsmanager

CER ist auf mehreren wichtigen Arten beim Abhängigkeitsmanager integriert:

- CER kann Abhängigkeiten erkennen, die von den Clients des Abhängigkeitsmanagers gespeichert werden sollen.
- CER verwendet den Abhängigkeitsmanager, um Abhängigkeiten für alle berechneten Attributwerte zu speichern, die in den CER-Datenbanktabellen gespeichert sind.
- Der Abhängigkeitsmanager fordert von CER die Neuberechnung aller berechneten Attributwerte an, die in den CER-Datenbanktabellen gespeichert sind, wenn sich einer ihrer Abhängigkeitsfaktoren ändert.

Diese Integrationspunkte zwischen CER und dem Abhängigkeitsmanager werden in den nachfolgenden Abschnitten detaillierter erläutert.

CER-Dienstprogramm für die Erkennung zu speichernder Abhängigkeiten

Ein CER-Client führt mit CER komplexe Berechnungen aus. Häufig muss der CER-Client auch Abhängigkeiten im Abhängigkeitsmanager speichern, damit der Abhängigkeitsmanager den Client benachrichtigen kann, wenn sich ein Abhängigkeitsfaktor ändert, und der Client anschließend CER erneut aufrufen kann, um die zugehörige Ausgabe unter Berücksichtigung der Änderungen an den Daten des Abhängigkeitsfaktors neu zu berechnen.

CER enthält ein Dienstprogramm, das den Clients bei der Erkennung der Abhängigkeiten hilft, die im Abhängigkeitsmanager gespeichert werden müssen. Das Dienstprogramm verwendet als Eingabe einen (von CER berechneten) Attributwert und gibt eine Gruppe von Abhängigkeitsfaktoren für diesen Attributwert zurück. Ein CER-Client kann dann sein Abhängigkeitsprodukt und die erkannten Abhängigkeitsfaktoren an den Abhängigkeitsmanager übergeben, damit dieser Abhängigkeitsdatensätze speichert.

Wenn CER einen Attributwert berechnet, wird hierbei eine umfassende Struktur logischer Abhängigkeiten berücksichtigt, die Folgendes umfasst:

- Als Stammelement den berechneten Attributwert selbst
- Als Verzweigungsknoten die berechneten Zwischenwerte (normalerweise für interne Regelobjekte)
- Als Blattknoten die externen Eingabedaten, die während der CER-Berechnung abgerufen wurden

Das Dienstprogramm kann diese Struktur logischer Abhängigkeiten syntaktisch analysieren, um eine viel kleinere Gruppe von Abhängigkeitsfaktoren zu erstellen, die normalerweise auf den Blattknoten der Struktur basiert. Anders ausgedrückt werden die Zwischenberechnungsergebnisse in der Regel ignoriert und anhand der gespeicherten Abhängigkeiten ist erkennbar, dass der berechnete Attributwert letztendlich von den externen Eingabedaten abhängig ist, auf die während der Berechnungen zugegriffen wurde.

Anmerkung: Für alle nicht trivialen Berechnungen (z. B. die normalerweise von CER ausgeführten Berechnungen) werden "zwischen" dem Gesamtabhängigkeitsprodukt und den eingegebenen Abhängigkeitsfaktoren eine Reihe von Zwischenwerten abgeleitet.

Diese Zwischenwerte werden nicht an den Abhängigkeitsmanager übergeben. Der Abhängigkeitsmanager speichert vielmehr Abhängigkeitsdatensätze, die übergeordnete Abhängigkeitsprodukte (z. B. die Leistungshöhe eines Falls) direkt mit ihren untergeordneten Abhängigkeitsfaktoren (z. B. Daten für Entität, Angaben und Sätze) verknüpfen.

Zwischenwerte sind für die Speicherung von Abhängigkeiten nicht relevant.

Die vom Dienstprogramm erkannten Abhängigkeitsfaktoren sind eine Kombination aus Folgendem:

- Von CER direkt erkannte Abhängigkeitsfaktoren
- Von den beim CER-Datenbankdatenspeicher registrierten Regelobjektkonvertern erkannte Abhängigkeitsfaktoren

Von CER direkt erkannte Abhängigkeitsfaktoren:

Das Dienstprogramm erkennt die folgenden Typen von Abhängigkeiten für einen berechneten Attributwert direkt.

Die Gesamtberechnung ist die Berechnung für das eigentliche Attribut oder für einen der internen Attributwerte, von dem es letztendlich abhängig ist (die "Zwischenberechnungen" zwischen dem berechneten Attributwert und seinen externen Dateneingaben).

Tabelle 7. Von CER direkt erkannte Abhängigkeitsfaktoren

Name	Zeitpunkt der Erkennung	Auslöser für Neuberechnung
Gespeicherter Attributwert	Erkennt einen "eingegebenen" Attributwert, der in den CER-Datenbanktabellen gespeichert ist und während der Gesamtberechnung des Attributwerts abgerufen wurde. Die Abhängigkeitsfaktor-ID verweist auf die interne ID für die Datenbankzeile des Attributs in den CER-Datenbanktabellen.	Falls sich der Wert des gespeicherten Attributs ändert, wird ein Änderungselement für Abhängigkeitsfaktoren für den gespeicherten Attributwert in einen Änderungssatz für Abhängigkeitsfaktoren geschrieben.
Regelwerkdefinitionen	Erkennt jedes Regelwerk, das eines der Attribute enthält, die in der Gesamtberechnung des Attributwerts verwendet werden. Die Abhängigkeitsfaktor-ID verweist auf den Namen des Regelwerks, in dem eine oder mehrere bei der Gesamtberechnung festgestellte Attributdefinitionen enthalten sind.	Falls Änderungen an einem CER-Regelwerk veröffentlicht werden, wird ein Änderungselement für Abhängigkeitsfaktoren für das geänderte Regelwerk in einen Änderungssatz für Abhängigkeitsfaktoren geschrieben.

Tabelle 7. Von CER direkt erkannte Abhängigkeitsfaktoren (Forts.)

Name	Zeitpunkt der Erkennung	Auslöser für Neuberechnung
Suche mit "readall"	Erkennt alle Ausdrücke "readall" (siehe „readall" auf Seite 224) ohne verschachtelte Ausdrücke `match`, die während der Gesamtberechnung festgestellt wurden und die in CER-Datenbanktabellen gespeicherte Regelobjekte abgerufen haben (im Gegensatz zu Regelobjekten, die mit Regelobjektkonvertern abgerufen wurden; siehe hierzu stattdessen „Von Regelobjektkonvertern erkannte Abhängigkeitsfaktoren" auf Seite 105). Die Abhängigkeitsfaktor-ID verweist auf den Namen der Regelklasse, die vom Ausdruck "readall" (siehe „readall" auf Seite 224) gesucht wurde.	Ein Änderungselement für Abhängigkeitsfaktoren für die Regelklasse wird in einen Änderungssatz für Abhängigkeitsfaktoren geschrieben, wenn eine der folgenden Bedingungen zutrifft: • In den CER-Datenbanktabellen wird ein neues Regelobjekt für diese Regelklasse gespeichert. • Aus den CER-Datenbanktabellen wird ein vorhandenes Regelobjekt für diese Regelklasse entfernt.
Suche mit "readall"/"match"	Erkennt alle Ausdrücke "readall" (siehe „readall" auf Seite 224), die während der Gesamtberechnung festgestellt wurden und die in CER-Datenbanktabellen gespeicherte Regelobjekte abgerufen haben (im Gegensatz zu Regelobjekten, die mit Regelobjektkonvertern abgerufen wurden; siehe hierzu stattdessen „Von Regelobjektkonvertern erkannte Abhängigkeitsfaktoren" auf Seite 105). Die Abhängigkeitsfaktor-ID verweist auf den Namen der Regelklasse, die vom Ausdruck "readall" (siehe „readall" auf Seite 224) gesucht wurde, zusammen mit dem als Suchbedingung verwendeten Attributnamen und -wert.	Ein Änderungselement für Abhängigkeitsfaktoren für die Regelklasse und ihren Attributnamen sowie Übereinstimmungswert wird in einen Änderungssatz für Abhängigkeitsfaktoren geschrieben, wenn eine der folgenden Bedingungen zutrifft: • In den CER-Datenbanktabellen wird ein neues Regelobjekt für diese Regelklasse gespeichert. • Aus den CER-Datenbanktabellen wird ein vorhandenes Regelobjekt für diese Regelklasse entfernt. • Der Wert des als Suchbedingung verwendeten Attributs ändert sich für ein vorhandenes Regelobjekt (in diesem Fall werden zwei Änderungselemente für Abhängigkeitsfaktoren geschrieben, eines für den alten Wert des Attributs und ein weiteres für den neuen Wert des Attributs).

Diese Typen von Abhängigkeiten lassen sich am besten anhand eines Beispiels veranschaulichen.

Ausgangspunkt des Beispiels ist ein neu geschriebenes System, das die Steuerpflicht einer Person mithilfe von CER berechnet. Dieses Steuerpflichtsystem ver-

wendet den Abhängigkeitsmanager, um Abhängigkeiten zu speichern, damit die Steuerpflicht (mit CER) neu berechnet werden kann, falls sich die Lebensumstände der Person ändern.

Das Steuerpflichtsystem speichert systemweite Informationen zu "Steuergrenzwerten" in Regelobjekten. Diese Regelobjekte werden in den CER-Datenbanktabellen gespeichert. Die CER-Regeln für die Berechnung der Steuerpflicht einer Person enthalten einen Ausdruck "readall" (siehe „readall" auf Seite 224), um alle Steuergrenzwerte im System abzurufen.

Das Steuerpflichtsystem speichert auch systemweite Informationen zu "Vermögen" in Regelobjekten. Diese Regelobjekte werden in den CER-Datenbanktabellen gespeichert. Für jedes Vermögen ist der Eigentümer und der Marktwert angegeben. Die CER-Regeln für die Berechnung der Steuerpflicht einer Person enthalten einen Ausdruck "readall" (siehe „readall" auf Seite 224), um das gesamte Vermögen im Eigentum dieser Person (also der Person mit einer Übereinstimmung von `Asset.ownedByPersonID` für `Person.personID`) abzurufen. Der Marktwert eines Vermögens kann geändert werden. Ebenso kann auch ein Vermögen von einer Person auf eine andere übertragen werden. Hierzu wird der Wert für `Asset.ownedByPersonID` von einer Personen-ID in eine andere Personen-ID geändert.

Die CER-Regeln für die Berechnung der Steuerpflicht einer Person beinhalten die Summierung des Wertes von `Asset.marketValue` für alle Vermögensbestandteile, die im Besitz dieser Person sind.

Das Steuerpflichtsystem enthält getrennte CER-Regelwerke für das Abrufen der Eingabedaten, die für die Berechnung der Steuerpflicht erforderlich sind, und für die eigentlichen Geschäftsberechnungen, die die Steuerpflicht einer Person anhand der abgerufenen Daten berechnen.

Ein Benutzer berechnet mit dem Steuerpflichtsystem die Steuerpflicht von Joe (personID 456) und von Mary (personID 457), die jeweils über ein Vermögenselement verfügen. Das Steuerpflichtsystem erkennt mithilfe des CER-Dienstprogramms Abhängigkeiten und übergibt diese zur Speicherung an den Abhängigkeitsmanager. Dies führt dazu, dass die folgenden Abhängigkeiten gespeichert werden:

Tabelle 8. Im Steuerpflichtbeispiel gespeicherte Abhängigkeiten

Typ des Abhängigkeitsparameters	ID des Abhängigkeitsparameters	Typ des Abhängigkeitsfaktors	ID des Abhängigkeitsfaktors
Steuerpflicht	456 (Personen-ID von Joe)	Suche mit "readall"	Regelklasse: TaxThreshold Wird gespeichert, weil die Berechnung von Joes Steuerpflicht den Abruf aller TaxThreshold-Regelobjekte verursachte.
Steuerpflicht	456	Suche mit "readall"/ "match"	Regelklasse: Asset, mit Attributwert "ownedByPersonID=456" Wird gespeichert, weil die Berechnung von Joes Steuerpflicht einen Abruf aller Asset-Regelobjekte verursacht hat, deren Eigentümer Joe ist.

Tabelle 8. Im Steuerpflichtbeispiel gespeicherte Abhängigkeiten (Forts.)

Typ des Abhängigkeitspaarobjekts	ID des Abhängigkeitspaarobjekts	Typ des Abhängigkeitsfaktors	ID des Abhängigkeitsfaktors
Steuerpflicht	456	Gespeicherter Attributwert	789, also die interne ID von `Asset.marketValue` für Joes Vermögenselement Wird gespeichert, weil die Berechnung von Joes Steuerpflicht auf den gespeicherten Wert des Attributs "marketValue" für das einzige für Joe abgerufene Vermögenselement zugegriffen hat.
Steuerpflicht	456	Regelwerkdefinition	TaxLiabilityDataRetrievalRuleSet Wird gespeichert, weil an der Berechnung von Joes Steuerpflicht die Definitionen von Regelattributen in "TaxLiabilityDataRetrievalRuleSet" beteiligt waren (verwendet zum Abrufen der TaxThreshold- und Asset-Regelobjekte).
Steuerpflicht	456	Regelwerkdefinition	TaxLiabilityBusinessCalculationsRuleSet Wird gespeichert, weil an der Berechnung von Joes Steuerpflicht die Definitionen von Regelattributen in "TaxLiabilityBusinessCalculationsRuleSet" beteiligt waren (verwendet zum Berechnen der Gesamtsteuerpflicht anhand der Eingabedaten).
Steuerpflicht	457 (Personen-ID von Mary)	Suche mit "readall"	Regelklasse: TaxThreshold Wird gespeichert, weil die Berechung von Marys Steuerpflicht den Abruf aller TaxThreshold-Regelobjekte verursachte.
Steuerpflicht	457	Suche mit "readall"/ "match"	Regelklasse: Asset, mit Attributwert "ownedByPersonID=457" Wird gespeichert, weil die Berechnung von Marys Steuerpflicht einen Abruf aller Asset-Regelobjekte verursacht hat, deren Eigentümerin Mary ist.
Steuerpflicht	457	Gespeicherter Attributwert	780, also die interne ID von `Asset.marketValue` für Marys Vermögenselement Wird gespeichert, weil die Berechnung von Marys Steuerpflicht auf den gespeicherten Wert des Attributs "marketValue" für das einzige für Mary abgerufene Vermögenselement zugegriffen hat.
Steuerpflicht	457	Regelwerkdefinition	TaxLiabilityDataRetrievalRuleSet Wird gespeichert, weil an der Berechnung von Marys Steuerpflicht die Definitionen von Regelattributen in "TaxLiabilityDataRetrievalRuleSet" beteiligt waren (verwendet zum Abrufen der TaxThreshold- und Asset-Regelobjekte).

Tabelle 8. Im Steuerpflichtbeispiel gespeicherte Abhängigkeiten (Forts.)

Typ des Abhängigkeitspaketes	ID des Abhängigkeitspaketes	Typ des Abhängigkeitsfaktors	ID des Abhängigkeitsfaktors
Steuerpflicht	457	Regelwerkdefinitionen	TaxLiabilityBusinessCalculationsRuleSet Wird gespeichert, weil an der Berechnung von Marys Steuerpflicht die Definitionen von Regelattributen in "TaxLiabilityBusinessCalculationsRuleSet" beteiligt waren (verwendet zum Berechnen der Gesamtsteuerpflicht anhand der Eingabedaten).

Nachfolgend ist dargestellt, wie Neuberechnungen der Steuerpflicht durch verschiedene Änderungen der Daten ausgelöst werden:

Tabelle 9. Beispiel für Änderungselemente für Abhängigkeitsfaktoren bei Steuerpflicht

Datenänderung	Aufgezeichnete Änderungselemente für Abhängigkeitsfaktoren	Ausgelöste Neuberechnungen
Der Marktwert von Joes Vermögenselement steigt von 100 auf 120 Euro.	• Gespeicherter Attributwert, 789	• Joes Steuerpflicht wird neu berechnet.
Mary verkauft ihr Vermögenselement und das zugehörige Regelobjekt wird entfernt.	• Suche mit "readall"/"match", Regelklasse: Asset, mit Attributwert "ownedByPersonID=457"	• Marys Steuerpflicht wird neu berechnet.
Joe erhält ein neues Vermögenselement, das als neues Regelobjekt gespeichert wird.	• Suche mit "readall"/"match", Regelklasse: Asset, mit Attributwert "ownedByPersonID=456"	• Joes Steuerpflicht wird neu berechnet.
Joe überträgt sein erstes Vermögenselement auf Mary. Der Wert des Attributs "ownedByPersonID" für das Vermögenselement ändert sich daher von 456 in 457.	• Suche mit "readall"/"match", Regelklasse: Asset, mit Attributwert "ownedByPersonID=456" (alter Wert) • Suche mit "readall"/"match", Regelklasse: Asset, mit Attributwert "ownedByPersonID=457" (neuer Wert)	• Joes Steuerpflicht wird neu berechnet. • Marys Steuerpflicht wird neu berechnet.
Ein Administrator führt einen neuen Steuergrenzwert ein, der als neues Regelobjekt gespeichert wird.	• Suche mit "readall", Regelklasse: TaxThreshold	• Joes Steuerpflicht wird neu berechnet. • Marys Steuerpflicht wird neu berechnet.
Ein Administrator entfernt einen vorhandenen Steuergrenzwert. Das zugehörige Regelobjekt wird daher entfernt.	• Suche mit "readall", Regelklasse: TaxThreshold	• Joes Steuerpflicht wird neu berechnet. • Marys Steuerpflicht wird neu berechnet.
Ein Administrator veröffentlicht Änderungen am Regelwerk "TaxLiabilityBusinessCalculationsRuleSet".	• Regelwerkdefinitionen, TaxLiabilityBusinessCalculationsRuleSet	• Joes Steuerpflicht wird neu berechnet. • Marys Steuerpflicht wird neu berechnet.

Von Regelobjektkonvertern erkannte Abhängigkeitsfaktoren: Auch die Regelobjektkonverter, die beim CER-Datenbankdatenspeicher registriert sind, können Beiträge zu den vom CER-Dienstprogramm erkannten Abhängigkeiten beisteuern.

Details über die erkannten Abhängigkeiten finden Sie in der Dokumentation des jeweiligen Regelobjektkonverters.

Speicherung von Abhängigkeiten für in CER gespeicherte Attributwerte durch CER mit Abhängigkeitsmanager

Bei Regelobjekten, die in den CER-Datenbanktabellen gespeichert sind, kann jeder Attributwert für diese Regelobjekte eine der folgenden Rollen übernehmen:

- Abhängigkeitsprodukt - ein Attributwert ist also durch die Berechnung seiner Werte aus Eingabedaten abgeleitet worden.
- Abhängigkeitsfaktor - ein Attributwert wird während der Berechnung eines Abhängigkeitsprodukts als Eingabedaten verwendet.

CER registriert beim Abhängigkeitsmanager einen Handler für Abhängigkeitsprodukte und einen Handler für Abhängigkeitsfaktoren, damit die zugehörigen gespeicherten Attributwerte in gespeicherten Abhängigkeiten als Abhängigkeitsprodukt und/oder Abhängigkeitsfaktor verwendet werden können.

Falls beispielsweise ein Attribut `totalIncome` für ein Regelobjekt `Person` vorhanden ist, das in CER-Datenbanktabellen gespeichert ist, und die Berechnung von `totalIncome` den Abruf der ebenfalls in den CER-Datenbanktabellen gespeicherten Attributwerte für `Income.amount` beinhaltete, weist CER den Abhängigkeitsmanager an, die Tatsache zu speichern, dass der Attributwert `Person.totalIncome` von den Attributwerten `Income.amount` abhängig ist.

Anforderung des Abhängigkeitsmanagers an CER zur Neuberechnung aller gespeicherten berechneten Attributwerte

Wenn sich Werte von Abhängigkeitsfaktoren ändern, erkennt der Abhängigkeitsmanager alle gespeicherten Attributwerte in CER, die von diesen geänderten Werten der Abhängigkeitsfaktoren abhängig sind, und fordert von CER die Neuberechnung der Werte für die zugehörigen Abhängigkeitsprodukte an. Hierzu wird der Handler für Abhängigkeitsprodukte verwendet, den CER beim Abhängigkeitsmanager registriert hat.

Falls sich beispielsweise der Wert eines in den CER-Datenbanktabellen gespeicherten Attributwerts `Income.amount` ändert, benachrichtigt CER den Abhängigkeitsmanager, dass der Abhängigkeitsfaktor `Income.amount` geändert wurde. Der Abhängigkeitsmanager erkennt anschließend, dass das Abhängigkeitsprodukt `Person.totalIncome` neu berechnet werden muss, und ruft CER für die Neuberechnung auf.

Konformität

Die Konformitätsinformationen zum Abhängigkeitsmanager finden Sie im Abschnitt „Abhängigkeitsmanager" auf Seite 273.

Informationen zum CER-Editor

Der CER-Editor vereinfacht die Erstellung und Verwaltung von CER-Regelwerken. Manche Regelwerke werden aus gesetzlichen Vorschriften abgeleitet, andere Regelwerke helfen Benutzern bei der Ausführung bestimmter Aktivitäten. Daher ist der CER-Editor mit zwei Basisansichten ausgestattet. Die Geschäftsansicht ist für Benutzer gedacht, die mit den für gesetzliche Vorschriften erforderlichen Strukturen

und Formulierungen vertraut sind. Die technische Ansicht wird für Benutzer bereitgestellt, die sich mit den Implementierungsaspekten der Regelentwicklung auskennen.

Der CER-Editor enthält eine globale Menüleiste, die allgemeine Benutzerfunktionen wie *Speichern*, *Suchen* und *Rückgängig* bzw. *Wiederholen* bietet. Mithilfe weiterer Speicherungsoptionen kann der Benutzer für Regelwerke die Aktionen *Export*, *Validieren* und *Alles speichern* ausführen.

Globales Menü des CER-Editors

Der CER-Editor bietet zum einfacheren Zugriff im Rahmen des globalen Menüs allgemeine Funktionen.

Globales Menü des CER-Editors

Tabelle 10. Menüleiste

Name	Beschreibung
Rückgängig	Wählen Sie in der Menüleiste die Option "Rückgängig" aus, um die letzte Bearbeitungsauswahl zu stornieren.
Wiederholen	Wählen Sie in der Menüleiste die Option "Wiederholen" aus, um die letzte Auswahl von "Rückgängig" zu stornieren. Dies macht die zuletzt ausgeführte Aktion rückgängig.
Regelwerke einschließen	Nach Auswahl dieser Option werden die eingeschlossenen Regelwerke angezeigt und Sie können durch die Angabe eines Klassenpfads Regelwerke hinzufügen.
Export	Der CER-Editor unterstützt das Exportieren aller Diagramme in der Regelgliederungsansicht in Form von PNG-Images, die als ZIP-Archivdatei auf der lokalen Festplatte des Benutzers gespeichert werden können.
Speichern	Ein im Editor geöffnetes Regelwerk kann jederzeit gespeichert werden. Das Speichern eines Regelwerks mit dem Status "Veröffentlicht" führt dazu, dass für dieses Regelwerk ein Datensatz des Typs "In Bearbeitung" erstellt wird. Alle vorgenommenen Änderungen werden im Datensatz des Typs "In Bearbeitung" gespeichert. Beim Speichern eines Regelwerks mit dem Status "In Bearbeitung" werden die Änderungen direkt gespeichert. Regelwerke können auch den Status "Neu erstellt" aufweisen. Das Speichern von Änderungen an einem neu erstellten Regelwerk führt zur direkten Speicherung der Änderungen im neu erstellten Regelwerk.
Alles speichern	Der CER-Editor unterstützt das Öffnen einer Anzahl von Regelwerken in einer einzigen Instanz des Editors. Eine Reihe von CER-Elementen kann auf Klassen oder Attribute verweisen, die in den anderen Regelwerken definiert sind. Für CER-Elemente, die auf ein Objekt in einem anderen Regelwerk verweisen, wird die Menüoption *Regelwerk öffnen* angezeigt. Mithilfe dieser Option kann ein Benutzer gleichzeitig eine beliebige Anzahl von Regelwerken im Editor öffnen. Die Menüoption *Alles speichern* speichert alle im Editor geöffneten und geänderten Regelwerke.

Tabelle 10. Menüleiste (Forts.)

Name	Beschreibung
Validieren	Mit dieser Option kann ein Benutzer seine Änderungen an einem Regelwerk validieren. Hierzu wird der Regelwerkvalidierer der CER-Regelengine aufgerufen. CER-Regelwerke sind XML-Dateien, die das von CER bereitgestellte Regelschema einhalten. CER enthält darüber hinaus einen umfassenden Regelwerkvalidierer, der Fehler in einem Regelwerk erkennen kann, bevor die Regelwerke ausgeführt werden. Der CER-Regelwerkvalidierer meldet in der Anzeige "Eigenschaften, Validierungen und Suchergebnisse" eine Liste von Fehlern, damit diese vom Benutzer korrigiert werden können. Falls Fehler vorliegen, meldet der CER-Regelwerkvalidierer die Fehler und stoppt die Verarbeitung.
Suchen	Mithilfe dieser Option kann ein Benutzer eine textbasierte Schnellsuche ausführen, die ein Suchergebnisfenster mit passenden Ergebnissen öffnet. Weitere Informationen zur bereitgestellten Funktionalität und zur Eingrenzung einer Suche finden Sie im Abschnitt "Suchtools des CER-Editors".

Suchtools des CER-Editors

Suchkriterien des CER-Editors: Ein Benutzer kann eine allgemeine textbasierte Suche für die folgenden Kriterien (Regeln und Regelverweise, Beschreibungen, technische Daten und Codetabellen) ausführen:

Tabelle 11. Suchkriterien

Name	Beschreibung
Regeln und Regelverweise	Geben Sie zur Suche nach einer Regel den Namen der Regel ein.
Beschreibungen	Geben Sie zur Suche nach Regeln oder nach Attributen in einer Regel einen Teil des Textes ein, der zur Beschreibung der Regel oder des Attributs verwendet wird.
Ordner	Geben Sie zur Suche nach einem Ordner den Namen des Ordners ein.
Technische Daten	Sie können nach allen Attributen suchen, die angegeben sind oder die nicht angegeben sind.
Codetabellen	Sie können nach allen Attributen suchen, die als Codetabellen angegeben sind.

Geschäftsansicht

Die Geschäftsansicht bietet einem Geschäftsbenutzer eine regelorientierte Ansicht, in der die relevanten Elemente für die Erstellung und Verwaltung der Geschäftsregeln verfügbar sind. Sie besteht aus einer Baumstruktur oder hierarchischen Ansicht der Regeln, Regelgrafiken, Geschäftselementpaletten sowie der Anzeige für Eigenschaften und Validierungen.

Regelgliederungsansicht: In dieser Ansicht werden alle übergeordneten Regeln sowie die Ordner angezeigt, in denen sich die Regeln befinden. Für jedes Element in der Gliederungsansicht ist ein Kontextmenü verfügbar:

Tabelle 12. Neue Menüeinträge

Name	Beschreibung
Ordner (Schaltfläche)	Hiermit können Sie einen Ordner erstellen, indem Sie einen neuen Ordnernamen angeben.
Regel (Schaltfläche)	Hiermit können Sie eine neue Regel erstellen, indem Sie einen neuen Regelnamen und den Datentyp für diese Regel angeben (Standardeinstellung ist der Typ "Boolesch"). Die Regel wird im ausgewählten Ordner erstellt. Falls kein Ordner ausgewählt ist, wird die neue Regel auf der Stammebene erstellt.
Neuer Ordner	Hiermit können Sie einen Ordner erstellen, indem Sie einen neuen Ordnernamen angeben.
Neue Regel	Hiermit können Sie eine neue Regel erstellen, indem Sie einen neuen Regelnamen und den Datentyp für diese Regel angeben (Standardeinstellung ist der Typ "Boolesch"). Die Regel wird im ausgewählten Ordner erstellt. Falls kein Ordner ausgewählt ist, wird die neue Regel auf der Stammebene erstellt.
Löschen	Hiermit können Sie je nach dem in der Gliederungsansicht hervorgehobenen Objekt einen Ordner oder eine Regel löschen.
Regeltyp festlegen	Hiermit können Sie eine Ableitung für ein Attribut erstellen. Für eine neue Regel wird eine Liste mit Datentypen bereitgestellt.
Regel verschieben	Hiermit kann ein Regelentwickler durch die Auswahl des Ordnernamens eine Regel von einem Ordner in einen anderen Ordner verschieben.
Verweise auf diese Regel suchen	Hiermit kann ein Benutzer alle Verweise auf die ausgewählte Regel suchen.

Technische Ansicht

Auf der Registerkarte "Technisch" werden alle Klassen und Attribute für die Regel angezeigt, die gerade bearbeitet wird. Sie können alle Klassen und Attribute entfernen, indem Sie erneut auf das Kontextmenü klicken, das der Klasse oder dem Attribut zugeordnet ist. Auf der Registerkarte "Technisch" der Gliederungsansicht sind die folgenden Aktionen verfügbar:

Klassengliederungsansicht: In dieser Ansicht werden alle übergeordneten Regeln sowie die Ordner angezeigt, in denen sich die Regeln befinden. Für jedes Element in der Gliederungsansicht ist ein Kontextmenü verfügbar:

Tabelle 13. Neue Menüeinträge

Name	Beschreibung
Klasse (Schaltfläche)	Hiermit können Sie eine Klasse erstellen, indem Sie einen neuen Klassennamen angeben.
Attribut (Schaltfläche)	Hiermit können Sie in einer bestimmten Klasse ein neues Attribut erstellen, indem Sie einen neuen Attributnamen angeben. Die Schaltfläche ist erst dann aktiviert, nachdem in der Gliederungsansicht eine Klasse ausgewählt wurde. Andernfalls ist die Schaltfläche inaktiviert.
Neues Attribut	Hiermit können Sie in einer bestimmten Klasse ein neues Attribut erstellen, indem Sie einen neuen Attributnamen angeben.
Löschen	Hiermit können Sie je nach dem in der Baumstrukturansicht hervorgehobenen Objekt eine Klasse oder ein Attribut löschen.

Tabelle 13. Neue Menüeinträge (Forts.)

Name	Beschreibung
Regeltyp festlegen	Hiermit kann ein Regelentwickler eine Ableitung für ein Attribut erstellen.
Verweise auf diese Regel suchen	Hiermit kann ein Benutzer alle Verweise auf die ausgewählte Regel suchen.

Diagrammgrafik

Die Diagrammgrafik bietet Steuerelemente zum Ziehen und Übergeben, zum Umschalten in die technische Ansicht sowie zum Schwenken und zum Zoomen.

Ziehen und Übergeben

Ein Regelelement kann aus einer beliebigen Palette gezogen und im Diagramm übergeben werden. Beispielsweise kann ein Element "Regel" gezogen und auf einem Attribut übergeben werden. Auf dem Ziel wird ein visueller Indikator dargestellt, falls das Ziel das gezogene Regelelement akzeptieren kann.

Regelelemente können außerdem zwischen Regelelementcontainern gezogen und übergeben werden. Beispielsweise kann ein Element "Regel" aus einem Ergebnisabschnitt des Regelelements "Wann" in den Ergebnisabschnitt eines anderen Regelelements "Wann" gezogen werden.

Auf Anzeige von detaillierten Diagrammen umschalten

Der CER-Editor bietet zwei verschiedene Typen von Ansichten für Regelelemente. Die Geschäftsansicht ist eine einfache Version, die für Geschäftsregelverfasser gedacht ist. Die technische Ansicht ist detaillierter und primär für technische Regelentwickler bestimmt. Der CER-Editor stellt eine Umschaltfunktion bereit, mit der zwischen der technischen Ansicht und der Geschäftsansicht gewechselt werden kann. Das Umschaltsymbol befindet sich in der Regelgrafik über den Steuerelementen für Schwenk und Zoom.

Steuerelemente für Schwenk und Zoom

Der CER-Editor bietet Steuerelemente für Schwenk und Zoom, mit denen ein Benutzer ein Regelwerk besser anzeigen und bearbeiten kann. Die Pfeile auf dem kreisförmigen Steuerelement können für Schwenks in umfangreichen Regeln eingesetzt werden. Wenn Sie auf die Schaltfläche in der Mitte des Steuerelements für die Schwenkfunktion klicken, wird das Image wieder an der Ausgangsposition zentriert. Über die Schaltfläche mit dem Pluszeichen kann die Darstellung vergrößert werden. Die Schaltfläche mit dem Minuszeichen dient zum Verkleinern der Darstellung:

Paletten für Tools und Vorlagen

Der CER-Editor bietet vier Typen von Paletten für Regelelemente sowie drei Paletten für Regelvorlagen. Die Paletten für Regelelemente heißen "Geschäftlich (Standard)", "Geschäftlich (Erweitert)", "Datentypen" und "Technisch". Die Vorlagen sind in den Paletten "Haushaltseinheiten", "Finanzeinheiten", "Einheiten für Unterstützung bei Nahrung" und "Entscheidungstabelle" zusammengefasst.

Paletten 'Geschäftlich'

Die Paletten "Geschäftlich" (sowohl der Typ "Standard" als auch der Typ "Erweitert") enthalten eine Reihe von Regelelementen, die zum Entwerfen der Geschäftslogik im Diagrammformat verwendet werden können.

Tabelle 14. Regelelemente der Paletten "Geschäftlich"

Bild	Name	Beschreibung
	Regel	Das Element "Regel" stellt eine grafische Darstellung des Ausdrucks "reference" bereit. Weitere Informationen finden Sie im Abschnitt „Regel" auf Seite 119.
	Regelgruppe UND	Das Element "Regelgruppe UND" stellt eine grafische Darstellung des Ausdrucks "all" bereit und gibt einen booleschen Wert zurück. Weitere Informationen finden Sie im Abschnitt „Regelgruppe UND" auf Seite 122.
	Regelgruppe ODER	Das Element "Regelgruppe ODER" stellt eine grafische Darstellung des Ausdrucks "any" bereit und gibt einen booleschen Wert zurück. Weitere Informationen finden Sie im Abschnitt „Regelgruppe ODER" auf Seite 122.
	Nicht	Das Element "Nicht" stellt eine grafische Darstellung des Ausdrucks "not" bereit und verneint einen booleschen Wert. Weitere Informationen finden Sie im Abschnitt „Nicht" auf Seite 122.
	Auswählen	Das Element "Auswählen" stellt eine grafische Darstellung des Ausdrucks "choose" bereit und wählt einen Wert auf der Grundlage einer erfüllten Bedingung aus. Weitere Informationen finden Sie im Abschnitt „Auswählen" auf Seite 123.
	Vergleichen	Das Element "Vergleichen" stellt eine grafische Darstellung des Ausdrucks "compare" bereit und vergleicht einen Wert auf der linken Seite mit einem Wert auf der rechten Seite gemäß dem angegebenen Vergleich. Weitere Informationen finden Sie im Abschnitt „Vergleichen" auf Seite 123.
	Wann	Das Element "Wann" ist Teil des Elements "Auswählen". Es enthält eine zu testende Bedingung und einen Wert, der zurückgegeben wird, wenn die Bedingung erfüllt ist. Weitere Informationen finden Sie im Abschnitt „Wann" auf Seite 124.
	Arithmetisch	Das Element "Arithmetisch" stellt eine grafische Darstellung des Ausdrucks "arithmetic" bereit und führt eine arithmetische Berechnung für zwei Zahlen (eine Zahl auf der linken Seite und eine Zahl auf der rechten Seite) aus. Das Ergebnis wird optional auf die angegebene Anzahl von Dezimalstellen gerundet. Weitere Informationen finden Sie im Abschnitt „Arithmetisch" auf Seite 124.
	Min.	Das Element "Min." stellt eine grafische Darstellung des Ausdrucks "min" bereit und ermittelt den kleinsten Wert in einer Liste (bzw. Null, falls die Liste leer ist). Weitere Informationen finden Sie im Abschnitt „Min." auf Seite 124.
	Max.	Das Element "Max." stellt eine grafische Darstellung des Ausdrucks "max" bereit und ermittelt den größten Wert in einer Liste (bzw. Null, falls die Liste leer ist). Weitere Informationen finden Sie im Abschnitt „Max." auf Seite 124.

Tabelle 14. Regelelemente der Paletten "Geschäftlich" (Forts.)

Bild	Name	Beschreibung
	Summe	Das Element "Summe" stellt eine grafische Darstellung des Ausdrucks "sum" bereit und berechnet die numerische Summe einer Liste von Zahlenwerten. Weitere Informationen finden Sie im Abschnitt „Summe” auf Seite 125.
	Wiederholungsregel	Das Element "Wiederholungsregel" stellt eine grafische Darstellung des Ausdrucks "dynamiclist" bereit und erstellt eine neue Liste, indem ein Ausdruck für jeden Eintrag einer vorhandenen Liste ausgewertet wird. Weitere Informationen finden Sie im Abschnitt „Wiederholungsregel” auf Seite 125.
	Filter	Das Element "Filter" stellt eine grafische Darstellung des Ausdrucks "filter" bereit und erstellt eine neue Liste mit allen Einträgen aus einer vorhandenen Liste, die die Filterbedingung erfüllen. Weitere Informationen finden Sie im Abschnitt „Filter” auf Seite 126.
	Größe	Das Element "Größe" stellt eine grafische Darstellung des Ausdrucks "property" bereit. Der Name der Eigenschaft wird auf "size" gesetzt. Weitere Informationen finden Sie im Abschnitt „Größe” auf Seite 127.
	Andernfalls	Das Element "Andernfalls" ist Teil des Elements "Auswählen". Es enthält einen zurückzugebenden Wert. Weitere Informationen finden Sie im Abschnitt „Andernfalls” auf Seite 127.
	Gesetzgebungsänderung	Das Element "Gesetzgebungsänderung" stellt eine grafische Darstellung des Ausdrucks "legislationchange" bereit und kann eines oder mehrere Elemente "Ära" enthalten. Weitere Informationen finden Sie im Abschnitt „Gesetzgebungsänderung” auf Seite 127.
	Ära	Das Element "Ära" ist Teil des Elements "Gesetzgebungsänderung" und enthält einen Datumseintrag (Leer ab) sowie einen Wert, der an das Element "Gesetzgebungsänderung" zurückgegeben wird. Weitere Informationen finden Sie im Abschnitt „Ära” auf Seite 127.

Palette 'Datentypen'

Die Palette "Datentypen" enthält eine Reihe von Regelelementen, die zum Entwerfen der Datentypen im Diagramm verwendet werden können.

Tabelle 15. Regelelemente der Palette "Datentypen"

Bild	Name	Beschreibung
	Boolean	Das Element "Boolesch" stellt eine grafische Darstellung des Ausdrucks "true" und "false" bereit. Es ist standardmäßig auf "true" gesetzt. Weitere Informationen finden Sie im Abschnitt „Boolean” auf Seite 128.

Tabelle 15. Regelelemente der Palette "Datentypen" (Forts.)

Bild	Name	Beschreibung
	String (Zeichenfolge)	Das Element "Zeichenfolge" stellt eine grafische Darstellung des Ausdrucks "String" bereit, die ein literaler konstanter Zeichenfolgewert ist. Weitere Informationen finden Sie im Abschnitt „String (Zeichenfolge)" auf Seite 128.
	Zahl	Das Element "Nummer" stellt eine grafische Darstellung des Ausdrucks "Number" bereit, die ein literaler konstanter Zahlenwert ist. Weitere Informationen finden Sie im Abschnitt „Zahl" auf Seite 128.
	Date (Datum)	Das Element "Datum" stellt eine grafische Darstellung des Ausdrucks "Date" bereit, die ein literaler konstanter Datumswert ist. Weitere Informationen finden Sie im Abschnitt „Date (Datum)" auf Seite 129.
	Codetabelle	Das Element "Codetabelle" stellt eine grafische Darstellung des Ausdrucks "Code" bereit, die ein literaler konstanter Wert ist, der einen Code aus einer Cúram-Codetabelle repräsentiert. Weitere Informationen finden Sie im Abschnitt „Codetabelle" auf Seite 129.
	Satz	Das Element "Satz" stellt eine grafische Darstellung des Ausdrucks "rate" bereit, die ein literaler konstanter Wert ist, der einen Satz aus einer Satztabelle repräsentiert. Weitere Informationen finden Sie im Abschnitt „Satz" auf Seite 129.
	Intervallmuster	Das Element "Intervallmuster" stellt eine grafische Darstellung des Ausdrucks "FrequencyPattern" bereit, die ein literaler konstanter Wert des Typs "FrequencyPattern" ist. Weitere Informationen finden Sie im Abschnitt „Intervallmuster" auf Seite 130.
	Ressourcennachricht	Das Element "Ressourcennachricht" stellt eine grafische Darstellung des Ausdrucks "ResourceMessage" bereit und erstellt aus einer Eigenschaftenressource eine lokalisierbare Nachricht. Weitere Informationen finden Sie im Abschnitt „Ressourcennachricht" auf Seite 130.
	XML-Nachricht	Das Element "XML-Nachricht" stellt eine grafische Darstellung des Ausdrucks "XMLMessage" bereit und erstellt aus einer Eigenschaftenressource eine lokalisierbare Nachricht. Weitere Informationen finden Sie im Abschnitt „XML-Nachricht" auf Seite 131.
	Null	Das Element "Null" stellt eine grafische Darstellung des Ausdrucks "null" bereit. Das Element "Null" kann in Elementen wie "Auswählen", "Wann", "Vergleichen" usw. verwendet werden, um einen beliebigen Wert mit Null zu vergleichen. Weitere Informationen finden Sie im Abschnitt „Null" auf Seite 131.

Palette 'Technisch'

Die Palette "Technisch" enthält eine Reihe von Regelelementen, die zum Entwerfen der technischen Logik im Diagrammformat verwendet werden können.

Tabelle 16. Regelelemente der Palette "Technisch"

Bild	Name	Beschreibung
	Erstellen	Das Element "Erstellen" stellt eine grafische Darstellung des Ausdrucks "create" bereit und ruft eine neue Instanz einer Regelklasse im Hauptspeicher der Sitzung ab. Weitere Informationen finden Sie im Abschnitt „Erstellen" auf Seite 131.
	Suchen	Das Element "Suchen" stellt eine grafische Darstellung des Ausdrucks "readall" bereit und ruft alle Regelobjektinstanzen einer Regelklasse ab, die durch Client-Code erstellt wurden. Weitere Informationen finden Sie im Abschnitt „Suchen" auf Seite 132.
	Feste Liste	Das Element "Feste Liste" stellt eine grafische Darstellung des Ausdrucks "fixedlist" bereit und erstellt eine neue Liste aus Elementen, die zum Zeitpunkt des Regelwerkentwurfs bekannt sind. Weitere Informationen finden Sie im Abschnitt „Feste Liste" auf Seite 133.
	Eigenschaft	Das Element "Eigenschaft" stellt eine grafische Darstellung des Ausdrucks "property" bereit. Es enthält eine Eigenschaft eines Java-Objekts. Weitere Informationen finden Sie im Abschnitt „Eigenschaft" auf Seite 133.
	Benutzerdefinierter Ausdruck	Das Element "Benutzerdefinierter Ausdruck" stellt eine grafische Darstellung für einen beliebigen benutzerdefinierten und gültigen XML-Knoten bereit. Weitere Informationen finden Sie im Abschnitt „Benutzerdefinierter Ausdruck" auf Seite 134.
	Existenzzeitlinie	Das Element "Existenzzeitlinie" stellt eine grafische Darstellung des Ausdrucks "existencetimeline" bereit. Weitere Informationen finden Sie im Abschnitt „Existenzzeitlinie" auf Seite 135.
	Timeline	Das Element "Zeitlinie" stellt eine grafische Darstellung des Ausdrucks "Timeline" bereit. Weitere Informationen finden Sie im Abschnitt „Timeline" auf Seite 135.
	Interval	Das Element "Intervall" stellt eine grafische Darstellung des Ausdrucks "Interval" bereit. Weitere Informationen finden Sie im Abschnitt „Interval" auf Seite 136.
	Kombinierter Folgesatz	Das Element "Kombinierter Folgesatz" stellt eine grafische Darstellung des Ausdrucks "combineSuccessionSet" bereit. Weitere Informationen finden Sie im Abschnitt „Kombinierter Folgesatz" auf Seite 136.
	Aufruf	Das Element "Aufruf" stellt eine grafische Darstellung des Ausdrucks "call" bereit und ruft eine statische Java-Methode auf, um eine komplexe Berechnung durchzuführen. Weitere Informationen finden Sie im Abschnitt „Aufruf" auf Seite 137.

Tabelle 16. Regelelemente der Palette "Technisch" (Forts.)

Bild	Name	Beschreibung
	Dauer des Zeitraums	Das Element "Dauer des Zeitraums" stellt eine grafische Darstellung des Ausdrucks "periodlength" bereit und berechnet die Menge der Zeiteinheiten zwischen zwei Datumsangaben. Weitere Informationen finden Sie im Abschnitt „Dauer des Zeitraums" auf Seite 138.
	ALLE	Das Element "ALLE" stellt eine grafische Darstellung des Ausdrucks "all" bereit und gibt einen booleschen Wert zurück. Weitere Informationen finden Sie im Abschnitt „ALLE" auf Seite 138.
	BELIEBIG	Das Element "BELIEBIG" stellt eine grafische Darstellung des Ausdrucks "any" bereit und gibt einen booleschen Wert zurück. Weitere Informationen finden Sie im Abschnitt „BELIEBIG" auf Seite 139.
	Diese/s/r	Das Element "Diese/s/r" stellt eine grafische Darstellung des Ausdrucks "this" bereit, die ein Verweis auf das aktuelle Regelobjekt ist. Weitere Informationen finden Sie im Abschnitt „Diese/s/r" auf Seite 139.
	Sortiert	Das Element "Sortieren" stellt eine grafische Darstellung des Ausdrucks "sort" bereit. Das Element verwendet eine Liste als untergeordnetes Element und sortiert sie. Weitere Informationen finden Sie im Abschnitt „Sortiert" auf Seite 139.
	Verweis auf gemeinsam verwendete Regel	Das Element "Verweis auf gemeinsam verwendete Regel" ist im Grunde genommen ein normaler Verweis, der ein Element "Erstellen" enthält. Weitere Informationen finden Sie im Abschnitt „Verweis auf gemeinsam verwendete Regel" auf Seite 139.
	Verketten	Das Element "Verketten" stellt eine grafische Darstellung des Ausdrucks "concat" bereit und erstellt durch das Verketten einer Liste von Werten eine Nachricht. Weitere Informationen finden Sie im Abschnitt „Verketten" auf Seite 141.
	Listen zusammenführen	Das Element "Listen zusammenführen" stellt eine grafische Darstellung des Ausdrucks "joinlists" bereit und erstellt eine neue Liste, indem einige vorhandene Listen zusammengeführt werden. Weitere Informationen finden Sie im Abschnitt „Listen zusammenführen" auf Seite 141.

Vorlage 'Haushaltseinheiten'

Die Vorlage "Haushaltseinheiten" enthält eine Reihe von Regelelementen, die zum Entwerfen der Haushaltseinheiten im Diagrammformat verwendet werden können.

Tabelle 17. Regelelemente der Vorlagenpalette "Haushaltseinheiten"

Bild	Name	Beschreibung
	Zusammensetzung des Haushalts	Weitere Informationen finden Sie im Abschnitt „Zusammensetzung des Haushalts" auf Seite 142.
	Kategorie "Haushalt"	Weitere Informationen finden Sie im Abschnitt „Kategorie "Haushalt"" auf Seite 142.

Vorlage 'Finanzeinheiten'

Die Vorlage "Finanzeinheiten" enthält eine Reihe von Regelelementen, die zum Entwerfen der Finanzeinheiten im Diagrammformat verwendet werden können.

Tabelle 18. Regelelemente der Vorlagenpalette "Finanzeinheiten"

Bild	Name	Beschreibung
	Finanzeinheit	Weitere Informationen finden Sie im Abschnitt „Finanzeinheit" auf Seite 142.
	Kategorie 'Finanzeinheit'	Weitere Informationen finden Sie im Abschnitt „Kategorie 'Finanzeinheit'" auf Seite 142.
	Mitglied von Finanzeinheit	Weitere Informationen finden Sie im Abschnitt „Mitglied von Finanzeinheit" auf Seite 142.

Vorlage 'Einheiten für Unterstützung bei Nahrung'

Die Vorlage "Einheiten für Unterstützung bei Nahrung" enthält eine Reihe von Regelelementen, die zum Entwerfen der Einheiten für die Unterstützung bei Nahrung im Diagrammformat verwendet werden können.

Tabelle 19. Regelelemente der Vorlagenpalette "Einheiten für Unterstützung bei Nahrung"

Bild	Name	Beschreibung
	Einheit für Unterstützung bei Nahrung	Weitere Informationen finden Sie im entsprechenden Abschnitt zur Einheit zur Unterstützung von Nahrung.
	Kategorie "Unterstützung bei Nahrung" für alleinstehende Person	Weitere Informationen finden Sie im Abschnitt „Kategorie "Unterstützung bei Nahrung" für alleinstehende Person" auf Seite 143.
	Kategorie 'Unterstützung bei Nahrung' für mehrere Personen	Weitere Informationen finden Sie im Abschnitt „Kategorie 'Unterstützung bei Nahrung' für mehrere Personen" auf Seite 143.
	Mitglieder der Verpflegungsgruppe	Weitere Informationen finden Sie im Abschnitt „Mitglieder der Verpflegungsgruppe" auf Seite 143.
	Verwandte	Weitere Informationen finden Sie im Abschnitt „Verwandte" auf Seite 143.
	Verwerfen	Weitere Informationen finden Sie im Abschnitt „Verwerfen" auf Seite 144.
	Mitglieder der Haushaltseinheit	Weitere Informationen finden Sie im Abschnitt „Mitglieder der Haushaltseinheit" auf Seite 144.
	Optionale Mitglieder	Weitere Informationen finden Sie im Abschnitt „Optionale Mitglieder" auf Seite 144.
	Ausnahmen	Weitere Informationen finden Sie im Abschnitt „Ausnahmen" auf Seite 144.

Vorlage 'Entscheidungstabelle'

Die Vorlage "Entscheidungstabelle" enthält das Element "Entscheidungstabelle", das zum Erstellen von Entscheidungstabellen verwendet werden kann.

Tabelle 20. Element der Palette "Entscheidungstabelle"

Bild	Name	Beschreibung
	Entscheidungstabelle	Das Element "Entscheidungstabelle" stellt eine grafische Darstellung einer Entscheidungstabelle bereit. Weitere Informationen finden Sie im Abschnitt „Entscheidungstabelle" auf Seite 144.

Popup-Menüs für Regelelemente

Das Popup-Menü für Regelelemente enthält allgemeine Optionen (Funktionen) für alle Regelelemente sowie spezielle Optionen (Funktionen) für ein einzelnes Regelelement. Informationen zu den speziellen Optionen für Regelelemente im Popup-Menü enthält „Referenzinformationen zu Regelelementen in Paletten des CER-Editors" auf Seite 119.

Tabelle 21. Allgemeine Optionen der Popup-Menüs

Name	Beschreibung
Ausschneiden	Mit dieser Option können Sie das Regelelement ausschneiden, damit es an einer anderen Position eingefügt werden kann.
Kopieren	Mit dieser Option können Sie ein vorhandenes Regelelement duplizieren, damit es an eine andere Position kopiert werden kann.
Löschen	Mit dieser Option können Sie das Regelelement aus dem Diagramm löschen.
Reduzieren	Mit dieser Option können Sie die untergeordneten Elemente des Regelelements ausblenden.
Erweitern	Mit dieser Option können Sie die untergeordneten Elemente des Regelelements einblenden.
Zeitlinie anfertigen	Diese Option ist nur in der technischen Ansicht verfügbar. Mit ihrer Hilfe können Sie eine Zeitlinie für das Regelelement erstellen.
Zeitlinienintervall anfertigen	Diese Option ist nur in der technischen Ansicht verfügbar. Mit ihrer Hilfe können Sie ein Zeitlinienintervall für das Regelelement erstellen.
Zeitlinie entfernen	Diese Option ist nur in der technischen Ansicht verfügbar. Mit ihrer Hilfe können Sie eine Zeitlinie aus dem Regelelement entfernen.
Zeitlinienintervall entfernen	Diese Option ist nur in der technischen Ansicht verfügbar. Mit ihrer Hilfe können Sie ein Zeitlinienintervall aus dem Regelelement entfernen.

Eigenschaften für Regelelemente

In diesem Abschnitt sind die allgemeinen Eigenschaften, die Eigenschaften für Regelklassen und die Attributeigenschaften beschrieben. Informationen zu den speziellen Eigenschaften eines Regelelements enthält „Referenzinformationen zu Regelelementen in Paletten des CER-Editors" auf Seite 119.

Reduzierbare Anzeigen für Eigenschaften und Validierung

Die Anzeigen für Eigenschaften und Validierung können Sie erweitern oder reduzieren, indem Sie auf die Umschaltfläche klicken, die sich auf dem Rand der Anzeigencontainer befindet. Wenn die Anzeigen für Eigenschaften und Validierung erweitert werden, werden die Eigenschaftendetails für das gegenwärtig ausgewähl-

te Diagramm eingeblendet. Durch eine Reduzierung der Anzeigen für Eigenschaften und Validierung werden diese Details ausgeblendet, damit mehr Platz für die Ansicht der Diagrammgrafik zur Verfügung steht.

Die Eigenschaftenanzeigen sind auf den Registerkarten "Geschäftlich" und "Technisch" gruppiert, damit je nach Perspektive des Regelentwicklers unterschiedliche Informationen angezeigt werden. Beispielsweise wird von einem Geschäftsregelentwickler nicht erwartet, dass er Kennzeichen für ein Regelelement eingibt, da diese Detailebene vom technischen Entwickler betreut wird.

Allgemeine Eigenschaften für alle Regelelemente

Tabelle 22. Allgemeine Eigenschaften

Name	Beschreibung
Anzeigename	Dies ist ein lokalisierbarer Name. Das Feld unterstützt sowohl Mehrbytezeichen als auch Zeichen mit Akzent. Es ist auf der Registerkarte "Geschäftlich" verfügbar.
Beschreibung	Das Feld "Beschreibung" unterstützt Texteinträge mit freiem Format für Regeln oder Attribute. Mehrbytezeichen und Zeichen mit Akzent werden unterstützt. In diesem Feld muss der Regelentwickler eine aussagekräftige Geschäftsbeschreibung der Regel oder des Attributs eingeben. Dieses Feld ist auf der Registerkarte "Geschäftlich" verfügbar. Der Inhalt des Feldes "Beschreibung" ist lokalisierbar und kann Mehrbytezeichen oder Zeichen mit Akzent umfassen.
Gesetzgebungslink	Dieses Feld enthält einen Link zu einer Website mit Gesetzgebungsinformationen. Es ist auf der Registerkarte "Geschäftlich" verfügbar.
Anzeigename-ID	Dieses Feld enthält eine ID für den Namen des Regelelements. Es ist auf der Registerkarte "Technisch" verfügbar.
Kennzeichen	Bei der Anmerkung für das Kennzeichen werden Kennzeicheneinträge aus Text mit freiem Format für jedes Element unterstützt, von dem Anmerkungen unterstützt werden. Dieser Text wird zur Suche nach einem Regelelement verwendet. Es ist auf der Registerkarte "Geschäftlich" verfügbar.

Eigenschaften für Regelklassen

Tabelle 23. Eigenschaften für Regelklassen

Name	Beschreibung
Primäres Attribut	Hier können Sie ein Attribut in einer Dropdown-Liste auswählen, das als primäres Attribut dieser Regelklasse verwendet werden soll. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Abstrakt	Bei Auswahl dieser Eigenschaft ist die Klasse eine abstrakte Klasse. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Erweitert	Benutzer können diese Regelklasse aus einer anderen Regelklasse erweitern, die entweder zum aktuellen oder zum externen Regelwerk gehört (im Assistenten für das Ändern von Regelwerken und Regelklassen festgelegt). Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Klasse	Der Name der Regelklasse. Sie wird auf der Registerkarte "Technisch" angezeigt.

Tabelle 23. Eigenschaften für Regelklassen (Forts.)

Name	Beschreibung
Folgesatz	Bei Auswahl dieser Eigenschaft wird die Anmerkung für den Folgesatz (SuccessionSet) hinzugefügt. Benutzer müssen in den Dropdown-Listen "Startdatumsattribut" und "Enddatumsattribut" Datumsattribute auswählen. Diese Eigenschaft wird auf der Registerkarte "Technisch" angezeigt.
Aktiver in Bearbeitung befindlicher Folgesatz	Bei Auswahl dieser Eigenschaft wird die Anmerkung für den aktiven in Bearbeitung befindlichen Folgesatz (ActiveInEditSuccessionSet) hinzugefügt. Benutzer müssen in den Dropdown-Listen "Startdatumsattribut" und "Enddatumsattribut" Datumsattribute auswählen. Diese Eigenschaft wird auf der Registerkarte "Technisch" angezeigt.

Eigenschaften für Attribute

Tabelle 24. Eigenschaften für Attribute

Name	Beschreibung
Datentyp	Der Datentyp eines Attributs. Falls der Datentyp in eine Zeitlinie geändert werden soll, wählen Sie das Kästchen "Zeitlinie" aus. Soll der Datentyp in eine Liste geändert werden, wählen Sie das Kästchen "Liste" aus. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Display	Bei Auswahl dieser Eigenschaft wird die Anmerkung für die Anzeige hinzugefügt. Der Benutzer muss einen Wert eingeben. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Unteranzeige anzeigen	Bei Auswahl dieser Eigenschaft wird die Anmerkung für das Anzeigen der Unteranzeige hinzugefügt. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Klasse	Der Name der Regelklasse, zu der das Attribut gehört. Sie wird auf der Registerkarte "Technisch" angezeigt.
Attribut	Der Name des Attributs. Sie wird auf der Registerkarte "Technisch" angezeigt.
Zugehöriger Folgesatz	Bei Auswahl dieser Eigenschaft wird die Anmerkung für den zugehörigen Folgesatz hinzugefügt. Der Benutzer muss in der Dropdown-Liste einen der Folgesätze auswählen (zur Verfügung stehen die Einträge "Keine", "Übergeordnetes Element" und "Untergeordnetes Element"). Sie wird auf der Registerkarte "Technisch" angezeigt.
Typ von zugehöriger Angabe	Bei Auswahl dieser Eigenschaft wird die Anmerkung für den Typ der zugehörigen Angabe hinzugefügt. Der Benutzer muss in der Dropdown-Liste einen der Folgesätze auswählen (zur Verfügung stehen die Einträge "Keine", "Übergeordnetes Element" und "Untergeordnetes Element"). Sie wird auf der Registerkarte "Technisch" angezeigt.
Abstrakt	Bei Auswahl dieser Eigenschaft wird das Attribut als abstraktes Regelelement und die Klasse als abstrakte Klasse definiert. Sie wird auf der Registerkarte "Technisch" angezeigt.

Assistenten für Regelelemente

Die Assistenten zum Ändern von Regelwerken und Regelklassen werden von einigen Regelelementen (z. B. dem Regelelement "Regel" oder "Erstellen") verwendet,

um eine Verknüpfung mit einer anderen Regelklasse aus entweder dem aktuellen Regelwerk oder einem externen Regelwerk herzustellen. In diesem Assistenten sind drei Optionen verfügbar:

Tabelle 25. Tabelle zum Assistenten für Regelelemente

Name	Beschreibung
Leerer Verweis	Hiermit kann der Regelautor einen Platzhalter für ein Regelelement erstellen, der später beim Entwickeln der Regel bearbeitet werden kann.
Neue Regel erstellen	Hiermit kann der Regelautor eine neue Regel erstellen, indem er im Dropdown-Kombinationsfeld den Namen der Regel und den Ergebnistyp der neuen Regel angibt.
Vorhandene Regel verwenden	Hiermit kann der Regelautor eine bereits erstellte Regel im Dropdown-Kombinationsfeld auswählen. Falls die Regel im vorhandenen Regelwerk nicht enthalten ist, kann der Regelautor ein anderes Regelwerk auswählen, indem er den Namen des Regelwerks eingibt und die Suchschaltfläche auswählt. Daraufhin wird ein weiteres Dialogfeld geöffnet, in dem der Regelautor das gewünschte Regelwerk auswählen kann.

Referenzinformationen zu Regelelementen in Paletten des CER-Editors

Dieser Abschnitt enthält Definitionen für alle Regelelemente, die im CER-Editor enthalten sind. Die Regelelemente sind anhand der unterschiedlichen Typen der Regelelementpaletten gruppiert. Hilfreiche Kategorisierungen dieser Regelelemente finden Sie in den bereits gelieferten Informationen.

Einführung

Dieser Abschnitt enthält Definitionen für alle Regelelemente, die im CER-Editor enthalten sind. Die Regelelemente sind nachfolgend anhand der unterschiedlichen Typen der Regelelementpaletten gruppiert. Hilfreiche Kategorisierungen dieser Regelelemente finden Sie im vorangegangenen Abschnitt.

Paletten

Sie können die Paletten abkoppeln, indem Sie eine Palette aus der ursprünglichen Position auf die Diagrammgrafik ziehen. Um die Palette wieder anzukoppeln, ziehen Sie sie einfach wieder auf die rechte Seite der Anzeige.

Reduzierbare Paletten

Den Palettencontainer können Sie erweitern oder reduzieren, indem Sie auf die Umschaltfläche klicken, die sich auf dem Rand des Palettencontainers befindet. Beim Erweitern des Palettencontainers werden der Name der gegenwärtig ausgewählten Palette und Beschreibungen der einzelnen Regelelemente angezeigt. Wenn Sie den Palettencontainer reduzieren, werden die Beschreibungen ausgeblendet und der für den Container im Diagramm benötigte Platz verringert.

Referenzinformationen zu Regelelementen in den Paletten "Geschäftlich (Standard)" und "Geschäftlich (Erweitert)"

Regel

Das Element "Regel" stellt eine grafische Darstellung des Ausdrucks "rule" bereit und verneint einen booleschen Wert. Zum Element "Verweis" können keine anderen Palettenelemente hinzugefügt werden. Das Element "Regel" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zu den Elementen "Regel-

gruppe UND", "Regelgruppe ODER" oder "Wiederholungsregel". Für Regelelemente gibt es verschiedene Typen von Einsatzszenarios. Ein Verweis auf eine Regel kann hinzugefügt werden, wenn ein Benutzer in der Geschäftsansicht (siehe Abschnitt 2.3.1 des Handbuchs *Working With CER*) und in der technischen Ansicht (siehe Abschnitt 2.3.2 des Handbuchs *Working With CER*) arbeitet. In der Geschäftsansicht sind die folgenden Optionen verfügbar:

- Leerer Verweis: Mit dieser Option können Benutzer einen leeren Verweis erstellen.
- Neue Regel erstellen: Mit dieser Option können Benutzer eine neue Regel erstellen.
- Vorhandene Regel verwenden: Mit dieser Option können Benutzer eine Regel im aktuellen Regelwerk auswählen oder nach externen Regelwerken suchen und eine Regel in einem externen Regelwerk auswählen.

In der technischen Ansicht sind die folgenden Optionen verfügbar:

- Vorhandene Regel verwenden: Mit dieser Option können Benutzer eine Regel im aktuellen Regelwerk auswählen oder nach externen Regelwerken suchen und eine Regel in einem externen Regelwerk auswählen.
- Neue Regel erstellen: Mit dieser Option können Benutzer eine neue Regel erstellen.

Falls ein Benutzer gerade in der Geschäftsansicht arbeitet und einen Verweis auf eine bestimmte Regelklasse oder ein bestimmtes Attribut hinzufügen will, muss er in die technische Ansicht wechseln.

Tabelle 26. Typen von Einsatzszenarios des Elements "Regel"

Name	Beschreibung
Verschachtelter Verweis	Für einen Verweis kann ein verschachtelter Verweis erstellt werden, der auf ein Attribut zeigt, das einen anderen Regelklassentyp besitzt, weil es beispielsweise nicht in der vorhandenen Klasse enthalten ist. Das Attribut des äußeren Verweises ist ein Objekt der Klasse für das Attribut des verschachtelten Verweises. Diese Struktur kann nur dann erstellt werden, wenn sich das Attribut des verschachtelten Verweises, das einen anderen Regelklassentyp besitzt, in der Klasse befindet, in der der verschachtelte Verweis erstellt wird.
Verschachtelter Verweis mit Erstellung	Für einen Verweis kann ein verschachtelter Verweis erstellt werden, der auf ein Attribut zeigt, das einen anderen Regelklassentyp besitzt, weil es beispielsweise nicht in der vorhandenen Klasse enthalten ist, sich das Attribut jedoch nicht in der aktuellen Klasse befindet. Das Attribut des äußeren Verweises ist ein Objekt der Klasse für das Attribut des verschachtelten Verweises. Diese Struktur kann nur dann erstellt werden, wenn sich das Attribut des verschachtelten Verweises (das einen anderen Regelklassentyp besitzt) in der Klasse befindet, bei der es sich nicht um die Klasse handelt, in der der verschachtelte Verweis erstellt wird.
CurrentUnitMemeber	Dient zum Verweis auf das Mitglied in der aktuellen Einheit, das die außerordentliche Testbedingung erfüllt.
FARelationship	Dient zum Verweis auf die Klasse, die als Beziehungsdatensatz für das Mitglied der Verpflegungsgruppe verwendet wird.
FAException	Dient zum Verweis auf eine Klasse, mit deren Hilfe überprüft wird, ob andere Mitglieder in einer Einheit die außerordentliche Bedingung erfüllen.

Tabelle 26. Typen von Einsatzszenarios des Elements "Regel" (Forts.)

Name	Beschreibung
HCCurrent	Zum Verweis auf einen aktuellen Listeneintrag (z. B. Mitglied des Haushalts usw.) in der Zusammensetzung des Haushalts wird "HCCurrent" verwendet.
Current	Zum Verweis auf einen aktuellen Listeneintrag in Listen wie beispielsweise "dynamiclist" wird das Element "Current" verwendet. Falls auf ein Attribut der Klasse für den aktuellen Listeneintrag verwiesen werden muss, wird das aktuelle Element in einen Verweis eingeschlossen, der auf dieses Attribut zeigt.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 27. Eigenschaften des Elements "Verweis"

Name	Beschreibung
Klasse	Der Name der Regelklasse. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Attribut	Der Name eines Attributs. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben. Sie wird auf der Registerkarte "Technisch" angezeigt.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.
Verhalten, wenn mehrere Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler, Rückgabe von Null, Rückgabe des ersten Elements, Rückgabe des letzten Elements) zurückgegeben, wenn mehrere Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 28. Popup-Menüoptionen für Element "Regel"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Regel" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Regel" in das Element "Regelgruppe UND" eingeschlossen.
Regel bearbeiten	Mit dieser Option können Sie das Element "Regel" bearbeiten, indem Sie die Regel auswählen, auf die verwiesen werden soll. Weitere Informationen erhalten Sie im Assistenten "Regel bearbeiten".
Diagramm für diese Regel öffnen	Diese Option öffnet das Diagramm für die Regel, auf die verwiesen wird, in einer neuen Diagrammregisterkarte.
Regellogik hier einschließen	Falls das Element "Regel" auf eine andere Regel (in demselben Regelwerk oder in einem anderen, externen Regelwerk) verweist, kann die Logik für diese Regel in die gegenwärtig im Editor angezeigte Regel eingeschlossen werden.

Regelgruppe UND

Das Element "Regelgruppe UND" stellt eine grafische Darstellung des Ausdrucks "all" bereit und gibt einen booleschen Wert zurück. Es wird für eine Liste mit booleschen Werten verwendet, um zu ermitteln, ob alle Listenwerte "true" lauten. Die Liste der booleschen Werte wird normalerweise durch einen Ausdruck "fixedlist" bereitgestellt. Andere Palettenelemente (z. B. "Verweis", "Wiederholungsregel" oder "Auswählen"), die eine Liste mit booleschen Werten bereitstellen, können zur Berechnung zum leeren Mitglied des Elements "Regelgruppe UND" hinzugefügt werden. Das Element "Regelgruppe UND" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wann").

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 29. Popup-Menüoptionen für Element "Regelgruppe UND"

Name	Beschreibung
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe UND" in ein anderes Element "Regelgruppe UND" eingeschlossen.
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe UND" in das Element "Regelgruppe ODER" eingeschlossen.
Ändern in ODER (OR)	Bei Auswahl dieser Option wird das Element "Regelgruppe UND" in das Element "Regelgruppe ODER" geändert.

Regelgruppe ODER

Das Element "Regelgruppe ODER" stellt eine grafische Darstellung des Ausdrucks "any" für boolesche Werte zurück, die zur Berechnung zum leeren Mitglied des Elements "Regelgruppe ODER" hinzugefügt werden können. Das Element "Regelgruppe ODER" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wann").

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 30. Popup-Menüoptionen für Element "Regelgruppe ODER"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe ODER" in ein anderes Element "Regelgruppe ODER" eingeschlossen.
Mit AND (UND) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe ODER" in das Element "Regelgruppe AND" eingeschlossen.
Ändern in UND (AND)	Bei Auswahl dieser Option wird das Element "Regelgruppe ODER" in das Element "Regelgruppe UND" geändert.

Nicht

Das Element "Nicht" stellt eine grafische Darstellung des Ausdrucks "not" bereit und verneint einen booleschen Wert. Andere Palettenelemente (z. B. "Verweis", "Regelgruppe ODER" oder "Regelgruppe UND"), die einen booleschen Wert zurückgeben, können zum Element "Nicht" hinzugefügt werden. Das Element "Nicht" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wiederholungsregel").

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 31. Popup-Menüoptionen für Element "Nicht"

Name	Beschreibung
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Nicht" in das Element "Regelgruppe UND" eingeschlossen.
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Nicht" in das Element "Regelgruppe ODER" eingeschlossen.

Auswählen

Das Element "Auswählen" stellt eine grafische Darstellung des Ausdrucks "choose" bereit und wählt einen Wert auf der Grundlage einer erfüllten Bedingung aus. Der Assistent "Auswahl bearbeiten" stellt neun Datentypen bereit (Zeichenfolge, Nummer, Boolesch, Datum, Datetime, Codetabelle, Nachricht, Zeitlinie, Codetabelle, Regelkasse, Java-Klasse, Nachricht). Zum Element "Auswählen" können nur die Elemente "Wann" und "Andernfalls" hinzugefügt werden. Das Element "Auswählen" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Regelgruppe UND", zum Element "Regelgruppe ODER" oder zur komplexen Ansicht des Elements "Vergleichen".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 32. Popup-Menüoptionen für Element "Auswählen"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Auswählen" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Auswählen" in das Element "Regelgruppe UND" eingeschlossen.
Auswahl bearbeiten	Diese Option stellt eine Liste von Datentypen für das Element "Auswählen" bereit. Weitere Informationen erhalten Sie nachfolgend im Dialogfeld "Auswahl bearbeiten".

Vergleichen

Das Element "Vergleichen" stellt eine grafische Darstellung des Ausdrucks "compare" bereit und vergleicht einen Wert auf der linken Seite mit einem Wert auf der rechten Seite gemäß dem angegebenen Vergleich. Wenn ein Element "Vergleichen" zu einem Diagramm hinzugefügt wird, erstellt es leere Einträge für die Argumente auf der linken Seite und auf der rechten Seite. Das Element "Vergleichen" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zu "Wann", "Regelgruppe UND" oder "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 33. Popup-Menüoptionen für Element "Vergleichen"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Vergleichen" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Vergleichen" in das Element "Regelgruppe UND" eingeschlossen.

Wann

Das Element "Wann" ist Teil des Elements "Auswählen". Es enthält eine zu testende Bedingung und einen Wert, der zurückgegeben wird, wenn die Bedingung erfüllt ist. Zur leeren Bedingung des Elements "Wann" können andere Palettenelemente hinzugefügt werden (z. B. "Code" oder "Beliebig"). Zum leeren Wert des Elements "Wann" können andere Palettenelemente hinzugefügt werden (z. B. "Nummer" oder "Verweis").

Arithmetisch

Das Element "Arithmetisch" stellt eine grafische Darstellung des Ausdrucks "arithmetic" bereit und führt eine arithmetische Berechnung für zwei Zahlen (eine Zahl auf der linken Seite und eine Zahl auf der rechten Seite) aus. Das Ergebnis wird optional auf die angegebene Anzahl von Dezimalstellen gerundet. Zum Element "Arithmetisch" können andere Palettenelemente (z. B. "Max.", "Min." oder "Verweis") hinzugefügt werden. Das Element "Arithmetisch" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 34. Eigenschaften des Elements "Arithmetisch"

Name	Beschreibung
Dezimalstellen	Hier kann ein Benutzer eingeben, wie viele Dezimalstellen verwendet werden sollen. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Runden	Hier können unterschiedliche Rundungstypen (Aufrundungsfunktion, Abrundung, Gaußklammer, Abrundung bei 5, Rundung bei 5 auf nächstliegende gerade Zahl, Aufrundung bei 5, Aufrundung) angegeben werden. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Min.

Das Element "Min." stellt eine grafische Darstellung des Ausdrucks "min" bereit und ermittelt den kleinsten Wert in einer Liste (bzw. Null, falls die Liste leer ist). Das Element "Min." enthält eine feste Liste, die jeden beliebigen Typ eines vergleichbaren Objekts enthält. Zum leeren Eintrag (feste Liste) des Elements "Min." können andere Palettenelemente hinzugefügt werden (z. B. "Nummer" oder "Verweis"). Das Element "Min." kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 35. Popup-Menüoptionen für Element "Min."

Name	Beschreibung
Ändern in MAX. (MAX)	Diese Option ändert das Element "Min." in das Element "Max.".

Max.

Das Element "Max." stellt eine grafische Darstellung des Ausdrucks "max" bereit und ermittelt den größten Wert in einer Liste (bzw. Null, falls die Liste leer ist). Das Element "Max." enthält eine feste Liste, die jeden beliebigen Typ eines ver-

gleichbaren Objekts enthält. Zum leeren Eintrag (feste Liste) des Elements "Max." können andere Palettenelemente hinzugefügt werden (z. B. "Nummer" oder "Verweis"). Das Element "Max." kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 36. Popup-Menüoptionen für Element "Max."

Name	Beschreibung
Ändern in MIN. (MIN)	Diese Option ändert das Element "Max." in das Element "Min.".

Summe

Das Element "Summe" stellt eine grafische Darstellung des Ausdrucks "sum" bereit und berechnet die numerische Summe einer Liste von Zahlenwerten. Das Element "Summe" enthält eine feste Liste, die jeden beliebigen Typ eines numerischen Objekts enthält. Zum leeren Eintrag (feste Liste) des Elements "Summe" können andere Palettenelemente hinzugefügt werden (z. B. "Nummer" oder "Verweis"). Das Element "Summe" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

Wiederholungsregel

Das Element "Wiederholungsregel" stellt eine grafische Darstellung des Ausdrucks "dynamiclist" bereit und erstellt eine neue Liste, indem ein Ausdruck für jeden Eintrag einer vorhandenen Liste ausgewertet wird. Zur leeren Liste des Elements "Wiederholungsregel" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis"). Zum leeren Eintrag (listitemexpression) des Elements "Wiederholungsregel" können andere Palettenelemente hinzugefügt werden (z. B. "Summe" oder "Auswählen"). Das Element "Wiederholungsregel" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wann").

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 37. Eigenschaften des Elements "Wiederholungsregel"

Name	Beschreibung
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben. Sie wird auf der Registerkarte "Technisch" angezeigt.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 38. Popup-Menüoptionen für Element "Wiederholungsregel"

Name	Beschreibung
Duplikate entfernen	Bei Auswahl dieser Option werden doppelte Einträge im Element "Wiederholungsregel" entfernt.
Ergebnisse verketten	Bei Auswahl dieser Option werden die Einträge im Element "Wiederholungsregel" verkettet.
Innere Listen zusammenführen	Bei Auswahl dieser Option werden die Listen in einer einzigen Liste zusammengeführt.
Erfolgreich bei beliebigem	Bei Auswahl dieser Option wird das Element "Wiederholungsregel" in das Element "Beliebig" eingeschlossen.
Erfolgreich bei allen	Bei Auswahl dieser Option wird das Element "Wiederholungsregel" in das Element "Alle" eingeschlossen.
Summierungselement	Bei Auswahl dieser Option wird eine Liste von Zahlen summiert.

Filter

Das Element "Filter" stellt eine grafische Darstellung des Ausdrucks "filter" bereit und erstellt eine neue Liste mit allen Einträgen aus einer vorhandenen Liste, die die Filterbedingung erfüllen. Zur leeren Liste des Elements "Filter" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis"). Zum leeren Eintrag (listitemexpression) des Elements "Filter" können andere Palettenelemente hinzugefügt werden (z. B. "Summe" oder "Auswählen"). Das Element "Filter" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zu den Elementen "Regelgruppe UND", "Regelgruppe ODER" und "Wann".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 39. Eigenschaften des Elements "Filter"

Name	Beschreibung
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben. Sie wird auf der Registerkarte "Technisch" angezeigt.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.
Verhalten, wenn mehrere Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler, Rückgabe von Null, Rückgabe des ersten Elements, Rückgabe des letzten Elements) zurückgegeben, wenn mehrere Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 40. Popup-Menüoptionen für Element "Filter"

Name	Beschreibung
Duplikate entfernen	Bei Auswahl dieser Option werden doppelte Einträge im Element "Filter" entfernt.
Ergebnisse verketten	Bei Auswahl dieser Option werden die Einträge im Element "Filter" verkettet.

Tabelle 40. Popup-Menüoptionen für Element "Filter" (Forts.)

Name	Beschreibung
Innere Listen zusammenführen	Bei Auswahl dieser Option werden die Listen in einer einzigen Liste zusammengeführt.
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Filter" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Filter" in das Element "Regelgruppe UND" eingeschlossen.

Größe

Das Element "Größe" stellt eine grafische Darstellung des Ausdrucks "property" bereit. Der Name dieses Elements wird auf "size" gesetzt. Das Element "Größe" erhält eine Eigenschaft eines Java-Objekts. Zum Element "Größe" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis" oder "Wiederholungsregel"). Das Element "Größe" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 41. Eigenschaften des Elements "Größe"

Name	Beschreibung
Wert	Der Name des Eigenschaftselements.

Andernfalls

Das Element "Andernfalls" ist Teil des Elements "Auswählen". Es enthält einen zurückzugebenden Wert. Zum leeren Wert des Elements "Andernfalls" können andere Palettenelemente hinzugefügt werden (z. B. "Nummer" oder "Verweis").

Gesetzgebungsänderung

Das Element "Gesetzgebungsänderung" stellt eine grafische Darstellung des Ausdrucks "legislationchange" bereit und kann eines oder mehrere Elemente "Ära" enthalten. Der Assistent "Gesetzgebungsänderung" stellt zehn Datentypen bereit (Zeichenfolge, Nummer, Boolesch, Datum, Datetime, Liste, Nachricht, Codetabelle, Regelkasse, Java-Klasse). Der ausgewählte Datentyp wird verwendet, um einen Rückgabewert des Elements "Ära" zu definieren.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 42. Popup-Menüoptionen für Element "Gesetzgebungsänderung"

Name	Beschreibung
Gesetzgebungsänderung bearbeiten	Nach Auswahl dieser Option können Sie das Element "Gesetzgebungsänderung" bearbeiten, indem Sie einen Typ für die neue Regel auswählen. Weitere Informationen erhalten Sie nachfolgend im Dialogfeld "Gesetzgebungsänderung bearbeiten".

Ära

Das Element "Ära" ist Teil des Elements "Gesetzgebungsänderung" und enthält einen Datumseintrag (Leer ab) sowie einen Wert, der an das Element "Gesetzgebungsänderung" zurückgegeben wird. Zum leeren Datumseintrag des Elements

"Ära" können andere Palettenelemente hinzugefügt werden (z. B. "Datum" oder "Verweis"). Zum leeren Wert des Elements "Ära" können andere Palettenelemente hinzugefügt werden (z. B. "Auswählen" oder "Verweis").

Referenzinformationen zu Regelelementen in der Palette "Datentypen"

Boolean

Das Element "Boolesch" stellt eine grafische Darstellung des Ausdrucks "true" und "false" bereit. Es ist standardmäßig auf "true" gesetzt. Das Element "Boolesch" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wiederholungsregel"). Zum Element "Boolesch" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 43. Popup-Menüoptionen für Element "Boolesch"

Name	Beschreibung
Ändern in "Falsch" (False)	Bei Auswahl dieser Otpion wird das Regelelement "Boolesch" in "Falsch" geändert, falls sein Wert "Wahr" lautet.
Ändern in "Wahr" (True)	Bei Auswahl dieser Otpion wird das Regelelement "Boolesch" in "Wahr" geändert, falls sein Wert "Falsch" lautet.

String (Zeichenfolge)

Das Element "Zeichenfolge" stellt eine grafische Darstellung des Ausdrucks "String" bereit, die ein literaler konstanter Zeichenfolgewert ist. Das Element "Zeichenfolge" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel". Zum Element "Zeichenfolge" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 44. Eigenschaften des Elements "Zeichenfolge"

Name	Beschreibung
Value	Der Wert des Elements "Zeichenfolge". Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Zahl

Das Element "Nummer" stellt eine grafische Darstellung des Ausdrucks "Number" bereit, die ein literaler konstanter Zahlenwert ist. Das Element "Nummer" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel". Zum Element "Nummer" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 45. Eigenschaften des Elements "Nummer"

Name	Beschreibung
Value	Der Wert des Elements "Nummer". Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Date (Datum)

Das Element "Datum" stellt eine grafische Darstellung des Ausdrucks "Date" bereit, die ein literaler konstanter Datumswert ist. Das Element "Datum" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Dauer des Zeitraums" oder zum Element "Ära". Zum Element "Datum" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 46. Eigenschaften des Elements "Datum"

Name	Beschreibung
Value	Der Wert des Elements "Datum". Die Standardeinstellung ist das aktuelle Datum. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Nulldatum	Nach Auswahl des Kontrollkästchens "Nulldatum" kann der Benutzer das Nulldatum in Cúram verwenden. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Codetabelle

Das Element "Codetabelle" stellt eine grafische Darstellung des Ausdrucks "Code" bereit, die ein literaler konstanter Wert ist, der einen Code aus einer Cúram-Codetabelle repräsentiert. Das Element "Codetabelle" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann". Zum Element "Codetabelle" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 47. Eigenschaften des Elements "Codetabelle"

Name	Beschreibung
Name der Codetabelle	Der Name der Codetabelle. Wenn Sie diese Eigenschaft leer lassen, können Sie nach allen verfügbaren Codetabellen suchen. Wenn Sie einen Wert eingeben, wird nach einer Codetabelle gesucht, deren Name mit dem Eingabewert beginnt. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Codetabellenwert	In diesem Dropdown-Feld sind alle Einträge aufgeführt, die in der ausgewählten Codetabelle enthalten sind. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Satz

Das Element "Satz" stellt eine grafische Darstellung des Ausdrucks "rate" bereit, die ein literaler konstanter Wert ist, der einen Satz aus einer Curám-Satztabelle repräsentiert. Das Element "Satz" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann". Zum Element "Satz" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 48. Eigenschaften des Elements "Satz"

Name	Beschreibung
Tabellenname	Der Name der Satztabelle. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Zeile	Der Zeilenwert der Satztabelle. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Spalte	Der Spaltenwert der Satztabelle. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Intervallmuster

Das Element "Intervallmuster" stellt eine grafische Darstellung des Ausdrucks "FrequencyPattern" bereit, die ein literaler konstanter Wert des Typs "FrequencyPattern" ist. Das Element "Intervallmuster" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Erstellen". Zum Element "Intervallmuster" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 49. Eigenschaften des Elements "Intervallmuster"

Name	Beschreibung
Muster	Das Intervallmuster. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 50. Popup-Menüoptionen für Element "Filter"

Name	Beschreibung
Intervallmuster bearbeiten	Nach Auswahl dieser Option können Sie das ausgewählte Intervallmuster bearbeiten.

Ressourcennachricht

Das Element "Ressourcennachricht" stellt eine grafische Darstellung des Ausdrucks "ResourceMessage" bereit und erstellt aus einer Eigenschaftenressource eine lokalisierbare Nachricht. Das Element "Ressourcennachricht" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Erstellen". Zum Element "Ressourcennachricht" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 51. Eigenschaften des Elements "Ressourcennachricht"

Name	Beschreibung
Schlüssel	Ressourcenpaketobjekte enthalten einen Bereich von Schlüssel/Wert-Paaren. Sie geben den Schlüssel an, der eine Zeichenfolge sein muss, wenn Sie den Wert aus dem Ressourcenpaket abrufen wollen. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Tabelle 51. Eigenschaften des Elements "Ressourcennachricht" (Forts.)

Name	Beschreibung
Ressourcenpaket	Der Name des Ressourcenpakets. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 52. Popup-Menüoptionen für Element "Ressourcennachricht"

Name	Beschreibung
Neues Argument	Nach Auswahl dieser Option können Sie ein neues Argument zum Regelelement "Ressourcennachricht" hinzufügen.
Argument entfernen	Nach Auswahl dieser Option können Sie ein Argument aus dem Regelelement "Ressourcennachricht" entfernen.

XML-Nachricht

Das Element "XML-Nachricht" stellt eine grafische Darstellung des Ausdrucks "XmlMessage" bereit und erstellt aus dem unformatierten XML-Inhalt eine Nachricht. Das Element "XML-Nachricht" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Erstellen". Zum Element "XML-Nachricht" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 53. Eigenschaften des Elements "XML-Nachricht"

Name	Beschreibung
Wert	Der Wert für den Inhalt der XML-Ressource.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 54. Popup-Menüoptionen für Element "XML-Nachricht"

Name	Beschreibung
Nachricht bearbeiten	Diese Option ruft ein Dialogfeld auf, in dem ein Benutzer den Inhalt der Nachricht eingeben kann.

Null

Das Element "Null" stellt eine grafische Darstellung des Ausdrucks "null" bereit. Das Element "Null" kann in Elementen wie "Auswählen", "Wann", "Vergleichen" usw. verwendet werden, um einen beliebigen Wert mit Null zu vergleichen.

Referenzinformationen zu Regelelementen in der Palette "Technisch"

Erstellen

Das Element "Erstellen" stellt eine grafische Darstellung des Ausdrucks "create" bereit und ruft eine neue Instanz einer Regelklasse im Hauptspeicher der Sitzung ab. Es unterstützt die beiden Parametertypen "Standard" und "Obligatorisch". Zu den Parametern des Elements "Erstellen" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis", "Wiederholungsregel" oder "Auswählen"). Das Element

"Erstellen" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wiederholungsregel" oder zum Element "Wann".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 55. Eigenschaften des Elements "Erstellen"

Name	Beschreibung
Klasse	Der Name der Regelklasse, die als Typ ausgewählt ist. Sie wird auf der Registerkarte "Technisch" angezeigt.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 56. Popup-Menüoptionen für Element "Erstellen"

Name	Beschreibung
Erstellung bearbeiten	Mit dieser Option können Sie das Element "Erstellen" bearbeiten, indem Sie die Regelklasse und das Regelwerk auswählen, auf die verwiesen werden soll.
Neuer Parameter	Mit dieser Option können Sie einen neuen Parameter erstellen, indem Sie das Attribut auswählen, für das Sie einen Parameter hinzufügen wollen.
Neuer obligatorischer Parameter	Mit dieser Option können Sie einen neuen obligatorischen Parameter erstellen.

Suchen

Das Element "Suchen" stellt eine grafische Darstellung des Ausdrucks "readall" bereit und ruft alle Regelobjektinstanzen einer Regelklasse ab, die durch Client-Code erstellt wurden. Es kann ein einzelnes Element aus einer Liste abrufen, wenn der Ausdruck "singleitem" ausgewählt ist. Das Element "Suchen" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Filter" oder zum Element "Erstellen". Zum Element "Suchen" können keine Elemente hinzugefügt werden.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 57. Eigenschaften des Elements "Suchen"

Name	Beschreibung
Klasse	Der Name der Regelklasse, die als Typ ausgewählt ist. Sie wird auf der Registerkarte "Technisch" angezeigt.
Regelwerk	Der Name des Regelwerks, das die ausgewählte Regelklasse enthält. Sie wird auf der Registerkarte "Technisch" angezeigt.
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben. Sie wird auf der Registerkarte "Technisch" angezeigt.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Kästchen "Einzelnes Element" ausgewählt ist.

Tabelle 57. Eigenschaften des Elements "Suchen" (Forts.)

Name	Beschreibung
Verhalten, wenn mehrere Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler, Rückgabe von Null, Rückgabe des ersten Elements, Rückgabe des letzten Elements) zurückgegeben, wenn mehrere Elemente gefunden wurden. Diese Option ist aktiv, wenn das Kästchen "Einzelnes Element" ausgewählt ist.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 58. Popup-Menüoptionen für Element "Suchen"

Name	Beschreibung
Suche bearbeiten	Mit dieser Option können Sie das Element "Suchen" bearbeiten, indem Sie die Regelklasse und das Regelwerk auswählen, auf die verwiesen werden soll.

Feste Liste

Das Element "Feste Liste" stellt eine grafische Darstellung des Ausdrucks "fixedlist" bereit und erstellt eine neue Liste aus Elementen, die zum Zeitpunkt des Regelwerkentwurfs bekannt sind. Der Assistent "Feste Liste" stellt neun Datentypen bereit (Zeichenfolge, Nummer, Boolesch, Datum, Datetime, Codetabelleneintrag, Regelkasse, Java-Klasse, Nachricht). Die Funktion "Unterliste" wird unterstützt. Zum leeren Eintrag des Elements "Feste Liste" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis", "Wiederholungsregel" oder "Auswählen"). Das Element "Feste Liste" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wann"). Abhängig vom ausgewählten Datentyp wird das Element "Feste Liste" in ein anderes Regelelement eingeschlossen, beispielsweise beim Typ "Boolesch" in das Regelelement "Regelgruppe UND" oder "Regelgruppe ODER", beim Typ "Zeichenfolge" in das Regelelement "Verketten" oder beim Typ "Nummer" in das Regelelement "Max.", "Min." oder "Summe".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 59. Eigenschaften des Elements "Feste Liste"

Name	Beschreibung
Datentyp	Der Datentyp des Elements "Feste Liste". Dieser Typ sollte dem Datentyp des Attributs entsprechen Falls der Datentyp in eine Zeitlinie geändert werden soll, wählen Sie das Kästchen "Zeitlinie" aus. Soll der Datentyp in eine Liste geändert werden, wählen Sie das Kästchen "Liste" aus. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Eigenschaft

Das Element "Eigenschaft" stellt eine grafische Darstellung des Ausdrucks "property" bereit. Es enthält eine Eigenschaft eines Java-Objekts. Zum Element "Eigenschaft" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis" oder "Wiederholungsregel"). Das Element "Eigenschaft" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 60. Eigenschaften des Elements "Eigenschaft"

Name	Beschreibung
Wert	Der Name des Eigenschaftselements.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 61. Popup-Menüoptionen für Element "Eigenschaft"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Eigenschaft" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Eigenschaft" in das Element "Regelgruppe UND" eingeschlossen.
Neues Argument	Nach Auswahl dieser Option können Sie ein neues Argument zum Regelelement "Eigenschaft" hinzufügen.
Argument entfernen	Nach Auswahl dieser Option können Sie ein Argument aus dem Regelelement "Eigenschaft" entfernen.

Wichtig: Seit Cúram Version 6 unterstützen CER und der Abhängigkeitsmanager die automatische Neuberechnung von durch CER berechneten Werten, falls sich ihre Abhängigkeiten ändern.

Wenn Sie die Implementierung der Methode für eine Eigenschaft ändern, wissen CER und der Abhängigkeitsmanager *nicht* automatisch, dass Attributwerte neu berechnet werden müssen, die unter Verwendung der alten Version der Methode für die Eigenschaft berechnet wurden.

Sobald eine Methode für eine Eigenschaft in einer Produktionsumgebung für gespeicherte Attributwerte verwendet wurde, sollten Sie anstelle einer Änderung der Implementierung eine neue Methode für die Eigenschaft erstellen (mit der erforderlichen neuen Implementierung) und Ihre Regelwerke so ändern, dass die neue Methode für die Eigenschaft verwendet wird. Wenn Sie Ihre Regelwerkänderungen, die auf die neue Methode für die Eigenschaft verweisen, veröffentlichen, berechnen CER und der Abhängigkeitsmanager automatisch alle Instanzen des betroffenen Attributwerts neu.

Benutzerdefinierter Ausdruck

Das Element "Benutzerdefinierter Ausdruck" stellt eine grafische Darstellung für einen beliebigen benutzerdefinierten und gültigen XML-Knoten bereit. Das Element "Benutzerdefinierter Ausdruck" kann zu jedem beliebigen Element in CER hinzugefügt werden. Der Benutzer muss sicherstellen, dass die Knotennamen des Elements "Benutzerdefinierter Ausdruck" nicht mit Knotennamen der CER-Sprache übereinstimmen.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 62. Popup-Menüoptionen für Element "Benutzerdefinierter Ausdruck"

Name	Beschreibung
Benutzerdefinierten Ausdruck bearbeiten	Mit dieser Option rufen Sie ein Popup-Feld auf, in dem Sie den benutzerdefinierten Ausdruck bearbeiten können.

Existenzzeitlinie

Das Element "Existenzzeitlinie" stellt eine grafische Darstellung des Ausdrucks "existencetimeline" bereit. Zu den Einträgen des Elements "Existenzzeitlinie" für das Startdatum und das Enddatum können andere Palettenelemente hinzugefügt werden (z. B. "Datum" oder "Verweis"). Zu den Einträgen des Elements "Existenzzeitlinie" für den Wert vor, während und nach der Existenzkönnen andere Palettenelemente hinzugefügt werden (z. B. "Datum" oder "Verweis"). Das Element "Existenzzeitlinie" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 63. Eigenschaften des Elements "Zeitlinie"

Name	Beschreibung
Datentyp	Der Datentyp des Elements "Existenzzeitlinie". Falls der Datentyp in eine Zeitlinie geändert werden soll, wählen Sie das Kästchen "Zeitlinie" aus. Soll der Datentyp in eine Liste geändert werden, wählen Sie das Kästchen "Liste" aus. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 64. Popup-Menüoptionen für Element "Existenzzeitlinie"

Name	Beschreibung
Existenzzeitlinie bearbeiten	Diese Option stellt zehn Datentypen (Zeichenfolge, Nummer, Boolesch, Datum, Datetime, Codetabelleneintrag, Regelklasse, Java-Klasse, Liste, Nachricht) für einen Intervalltyp des Regelelements "Existenzzeitlinie" bereit.

Timeline

Das Element "Zeitlinie" stellt eine grafische Darstellung des Ausdrucks "Timeline" bereit. Mithilfe der entsprechenden Menüoption kann für ein Element "Zeitlinie" ein Anfangswert festgelegt werden. Zum Element "Zeitlinie" können andere Palettenelemente hinzugefügt werden (z. B. "Intervall", "Feste Liste", "Wiederholungsregel" usw.).

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 65. Eigenschaften des Elements "Zeitlinie"

Name	Beschreibung
Datentyp	Der Datentyp des Elements "Zeitlinie". Falls der Datentyp in eine Zeitlinie geändert werden soll, wählen Sie das Kästchen "Zeitlinie" aus. Soll der Datentyp in eine Liste geändert werden, wählen Sie das Kästchen "Liste" aus. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 66. Popup-Menüoptionen für Element "Zeitlinie"

Name	Beschreibung
Intervalle hinzufügen	Nach Auswahl dieser Option können Sie Intervalle zur Zeitlinie hinzufügen.
Anfangswert hinzufügen	Nach Auswahl dieser Option können Sie einen Anfangswert zur Zeitlinie hinzufügen.
Intervalle entfernen	Nach Auswahl dieser Option können Sie Intervalle aus der Zeitlinie entfernen.
Anfangswert entfernen	Nach Auswahl dieser Option können Sie den Anfangswert aus der Zeitlinie entfernen.
Zeitlinie bearbeiten	Diese Option ruft den Assistenten "Zeitlinie" auf, in dem Sie die Zeitlinie bearbeiten können.

Interval

Das Element "Intervall" stellt eine grafische Darstellung des Ausdrucks "Interval" bereit. Zu den Einträgen des Elements "Intervall" für das Startdatum und das Enddatum können andere Palettenelemente hinzugefügt werden (z. B. "Datum" oder "Verweis"). Das Element "Intervall" kann nur zum Element "Intervalle" eines Elements "Zeitlinie" hinzugefügt werden. Der Intervalltyp kann mit dem Intervallassistenten festgelegt werden. Das Element "Intervall" enthält die untergeordneten Elemente "Start" und "Wert".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 67. Popup-Menüoptionen für Element "Intervall"

Name	Beschreibung
Intervall bearbeiten	Diese Option ruft den Assistenten für die Bearbeitung des Intervalls auf.

Kombinierter Folgesatz

Das Element "Kombinierter Folgesatz" stellt eine grafische Darstellung des Ausdrucks "combineSuccessionSet" bereit. Zum Element "Kombinierter Folgesatz" können andere Palettenelemente hinzugefügt werden (z. B. "Filter" oder "Feste Liste"). Das Element "Kombinierter Folgesatz" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Erstellen".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 68. Popup-Menüoptionen für Element "Kombinierter Folgesatz"

Name	Beschreibung
Kombinierten Folgesatz bearbeiten	Mit dieser Option können Sie das Element "Kombinierter Folgesatz" bearbeiten, indem Sie die Regelklasse und das Regelwerk auswählen, auf die verwiesen werden soll.

Aufruf

Das Element "Aufruf" stellt eine grafische Darstellung des Ausdrucks "call" bereit und ruft eine statische Java-Methode auf, um eine komplexe Berechnung durchzuführen. Zum Element "Aufruf" kann ein neues Argument hinzugefügt werden. Zum Argument des Elements "Aufruf" können andere Palettenelemente hinzugefügt werden (z. B. "Datum", "Dauer des Zeitraums" oder "Verweis"). Das Element "Dauer des Zeitraums" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Erstellen".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 69. Eigenschaften des Elements "Aufruf"

Name	Beschreibung
Klasse	Der Name der Klasse, die die aufrufende Methode enthält. Sie wird auf der Registerkarte "Technisch" angezeigt.
Methodenname	Der Name der Methode, die aufgerufen wird. Sie wird auf der Registerkarte "Technisch" angezeigt.
Datentyp	Der Rückgabetyp des Aufrufs. Dieser Typ sollte mit dem Datentyp des Attributs identisch sein. Falls der Datentyp in eine Zeitlinie geändert werden soll, wählen Sie das Kästchen "Zeitlinie" aus. Soll der Datentyp in eine Liste geändert werden, wählen Sie das Kästchen "Liste" aus. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 70. Popup-Menüoptionen für Element "Aufruf"

Name	Beschreibung
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Aufruf" in das Element "Regelgruppe UND" eingeschlossen.
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Aufruf" in das Element "Regelgruppe ODER" eingeschlossen.
Neues Argument	Nach Auswahl dieser Option können Sie ein neues Argument zum Regelelement "Aufruf" hinzufügen.
Argument entfernen	Nach Auswahl dieser Option können Sie ein Argument aus dem Regelelement "Aufruf" entfernen.

Wichtig: Seit Cúram Version 6 unterstützen CER und der Abhängigkeitsmanager die automatische Neuberechnung von durch CER berechneten Werten, falls sich ihre Abhängigkeiten ändern.

Wenn Sie die Implementierung einer statischen Methode ändern, wissen CER und der Abhängigkeitsmanager *nicht* automatisch, dass Attributwerte neu berechnet werden müssen, die unter Verwendung der alten Version der statischen Methode berechnet wurden.

Sobald eine statische Methode in einer Produktionsumgebung für gespeicherte Attributwerte verwendet wurde, sollten Sie anstelle einer Änderung der Implementierung eine neue statische Methode erstellen (mit der erforderlichen neuen Implementierung) und Ihre Regelwerke so ändern, dass die neue statische Methode verwendet wird. Wenn Sie Ihre Regelwerkänderungen, die auf die neue statische Methode verweisen, veröffentlichen, berechnen CER und der Abhängigkeitsmanager automatisch alle Instanzen des betroffenen Attributwerts neu.

Dauer des Zeitraums

Das Element "Dauer des Zeitraums" stellt eine grafische Darstellung des Ausdrucks "periodlength" bereit und berechnet die Menge der Zeiteinheiten zwischen zwei Datumsangaben. Zum Element "Dauer des Zeitraums" können andere Palettenelemente hinzugefügt werden (z. B. "Datum", "Aufruf" oder "Verweis"). Das Element "Dauer des Zeitraums" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Erstellen".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 71. Eigenschaften des Elements "Dauer des Zeitraums"

Name	Beschreibung
Einheit	Diese Eigenschaft bietet vier Typen von Einheiten (Tage, Wochen, Monate, Jahre) für das Regelelement "Dauer des Zeitraums". Sie wird auf der Registerkarte "Technisch" angezeigt.
Einschluss von Enddatum	Diese Eigenschaft stellt zwei Möglichkeiten für Datumseinschlüsse bereit (inklusive oder exklusive). Sie wird auf der Registerkarte "Technisch" angezeigt.

ALLE

Das Element "ALLE" stellt eine grafische Darstellung des Ausdrucks "all" bereit und gibt einen booleschen Wert zurück. Es wird für eine Liste mit booleschen Werten verwendet, um zu ermitteln, ob alle Listenwerte "true" lauten. Das Element "ALLE" enthält kein Element "Feste Liste" (fixedlist). Zum Element "ALLE" können andere Palettenelemente hinzugefügt werden (z. B. "Wiederholungsregel" oder "Feste Liste"). Das Element "ALLE" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 72. Popup-Menüoptionen für Element "ALLE"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe UND" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe UND" in ein anderes Element "Regelgruppe UND" eingeschlossen.
Ändern in ODER (OR)	Bei Auswahl dieser Option wird das Element "Regelgruppe UND" in das Element "Regelgruppe ODER" geändert.

BELIEBIG

Das Element "BELIEBIG" stellt eine grafische Darstellung des Ausdrucks "any" bereit und gibt einen booleschen Wert zurück. Es wird für eine Liste mit booleschen Werten verwendet, um zu ermitteln, ob ein beliebiger der Listenwerte "true" lautet. Das Element "BELIEBIG" enthält kein Element "Feste Liste" (fixedlist). Zum Element "BELIEBIG" können andere Palettenelemente hinzugefügt werden (z. B. "Wiederholungsregel" oder "Feste Liste"). Das Element "BELIEBIG" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 73. Popup-Menüoptionen für Element "Sortieren"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe ODER" in ein anderes Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Regelgruppe ODER" in das Element "Regelgruppe AND" eingeschlossen.
Ändern in UND (AND)	Bei Auswahl dieser Option wird das Element "Regelgruppe ODER" in das Element "Regelgruppe UND" geändert.

Diese/s/r

Das Element "Diese/s/r" stellt eine grafische Darstellung des Ausdrucks "this" bereit, die ein Verweis auf das aktuelle Regelobjekt ist. Das Element "Diese/s/r" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "BELIEBIG". Zum Element "Diese/s/r" können keine Elemente hinzugefügt werden.

Sortiert

Das Element "Sortieren" stellt eine grafische Darstellung des Ausdrucks "sort" bereit. Es verwendet eine Liste als untergeordnetes Element und sortiert sie.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 74. Popup-Menüoptionen für Element "Sortieren"

Name	Beschreibung
Neue Position für aufsteigende Sortierung	Nach Auswahl dieser Option können Sie eine neue Position für die aufsteigende Sortierung hinzufügen.
Neue Position für absteigende Sortierung	Nach Auswahl dieser Option können Sie eine neue Position für die absteigende Sortierung hinzufügen.

Verweis auf gemeinsam verwendete Regel

Das Element "Verweis auf gemeinsam verwendete Regel" ist im Grunde genommen ein normaler Verweis, der ein Element "Erstellen" enthält. Das Element "Verweis auf gemeinsam verwendete Regel" ruft einen Assistenten auf, in dem die Namen der Regelklassen angezeigt werden, für die das primäre Attribut im aktuellen Regelwerk festgelegt ist. Das Element "Verweis auf gemeinsam verwendete Regel" kann nach Auswahl der Menüoption "Verweis auf gemeinsam verwendete Regel bearbeiten" für das Diagramm bearbeitet werden. Eine gemeinsam verwendete Regel ist eine Klasse, die ein primäres Attribut enthält. Mit dem Assistenten für ge-

meinsam verwendete Regeln kann ein Benutzer eine gemeinsam verwendete Regel erstellen, die zum Erstellen von Verweisen auf gemeinsam verwendete Regeln verwendet werden kann.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 75. Eigenschaften des Elements "Verweis auf gemeinsam verwendete Regel"

Name	Beschreibung
Klasse	Der Name der Regelklasse. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Attribut	Der Name eines Attributs. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben. Sie wird auf der Registerkarte "Technisch" angezeigt.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.
Verhalten, wenn mehrere Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler, Rückgabe von Null, Rückgabe des ersten Elements, Rückgabe des letzten Elements) zurückgegeben, wenn mehrere Elemente gefunden wurden. Diese Option ist aktiv, wenn das Feld "Einzelnes Element" ausgewählt ist.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 76. Popup-Menüoptionen für Element "Verweis auf gemeinsam verwendete Regel"

Name	Beschreibung
Mit ODER (OR) umschließen	Bei Auswahl dieser Option wird das Element "Verweis auf gemeinsam verwendete Regel" in das Element "Regelgruppe ODER" eingeschlossen.
Mit UND (AND) umschließen	Bei Auswahl dieser Option wird das Element "Verweis auf gemeinsam verwendete Regel" in das Element "Regelgruppe UND" eingeschlossen.
Verweis bearbeiten	Mit dieser Option können Sie das Element "Verweis" bearbeiten, indem Sie die Regel auswählen, auf die verwiesen werden soll.
Verweis auf gemeinsam verwendete Regel bearbeiten	Mit dieser Option können Sie das Element "Verweis auf gemeinsam verwendete Regel" bearbeiten, indem Sie die gemeinsam verwendete Regel auswählen, auf die verwiesen werden soll. Weitere Informationen erhalten Sie nachfolgend im Dialogfeld "Verweis auf gemeinsam verwendete Regel bearbeiten".
Neuer Parameter	Mit dieser Option können Sie einen neuen Parameter erstellen, indem Sie das Attribut auswählen, für das Sie einen Parameter hinzufügen wollen. Weitere Informationen erhalten Sie nachfolgend im Dialogfeld "Neuer Parameter".
Neuer obligatorischer Parameter	Mit dieser Option können Sie einen neuen obligatorischen Parameter erstellen.

Verketten

Das Element "Verketten" stellt eine grafische Darstellung des Ausdrucks "concat" bereit und erstellt durch das Verketten einer Liste von Werten eine lokalisierbare Nachricht. Das Element "Verketten" enthält eine feste Liste (fixedlist) mit Zeichenfolgeobjekten. Zum leeren Eintrag (feste Liste) des Elements "Verketten" können andere Palettenelemente hinzugefügt werden (z. B. "Zeichenfolge" oder "Verweis"). Das Element "Verketten" kann zu anderen Palettenelementen hinzugefügt werden, beispielsweise zum Element "Wann" oder zum Element "Wiederholungsregel".

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 77. Eigenschaften des Elements "Verketten"

Name	Beschreibung
Datentyp	Der Datentyp des Elements "Verketten". Dieser Typ muss dem Datentyp des Attributs entsprechen. Falls der Datentyp in eine Zeitlinie geändert werden soll, wählen Sie das Kästchen "Zeitlinie" aus. Soll der Datentyp in eine Liste geändert werden, wählen Sie das Kästchen "Liste" aus. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.

Listen zusammenführen

Das Element "Listen zusammenführen" stellt eine grafische Darstellung des Ausdrucks "joinlists" bereit und erstellt eine neue Liste, indem einige vorhandene Listen zusammengeführt werden. Zum leeren Eintrag (fixedlist) des Elements "Listen zusammenführen" können andere Palettenelemente hinzugefügt werden (z. B. "Verweis" oder "Auswählen"). Das Element "Listen zusammenführen" kann zu anderen Palettenelementen hinzugefügt werden (z. B. "Regelgruppe UND", "Regelgruppe ODER" oder "Wann").

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 78. Eigenschaften des Elements "Listen zusammenführen"

Name	Beschreibung
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden.
Verhalten, wenn mehrere Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler, Rückgabe von Null, Rückgabe des ersten Elements, Rückgabe des letzten Elements) zurückgegeben, wenn mehrere Elemente gefunden wurden.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 79. Popup-Menüoptionen für Lists "Listen zusammenführen"

Name	Beschreibung
Duplikate entfernen	Bei Auswahl dieser Option werden doppelte Einträge im Element "Listen zusammenführen" entfernt.

Tabelle 79. Popup-Menüoptionen für Lists "Listen zusammenführen" (Forts.)

Name	Beschreibung
Ergebnisse verketten	Bei Auswahl dieser Option werden die Einträge im Element "Listen zusammenführen" verkettet.
Innere Listen zusammenführen	Bei Auswahl dieser Option werden die Listen in einer einzigen Liste zusammengeführt.

Referenzinformationen zu Regelelementen in den Vorlagen "Haushaltseinheiten"

Zusammensetzung des Haushalts

Diese Zusammensetzungsvorlage wird in Regeln von CGIS (Curam Global Income Support) verwendet und stellt eine Zusammensetzung des Haushalts dar.

Kategorie "Haushalt"

Diese Vorlage stellt eine neue Kategorie "Haushalt" für einen Haushalt in CGIS-Regeln dar.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 80. Popup-Menüoptionen für Kategorie "Haushalt"

Name	Beschreibung
Obligatorische Mitglieder hinzufügen	Diese Option fügt ein neues obligatorisches Mitglied zum Regelelement **Kategorie "Haushalt"** hinzu.
Obligatorische Mitglieder entfernen	Diese Option entfernt ein obligatorisches Mitglied aus dem Regelelement **Kategorie "Haushalt"**.
Optionale Mitglieder hinzufügen	Diese Option fügt ein neues optionales Mitglied zum Regelelement **Kategorie "Haushalt"** hinzu.
Optionale Mitglieder entfernen	Diese Option entfernt ein optionales Mitglied aus dem Regelelement **Kategorie "Haushalt"**.

Referenzinformationen zu Regelelementen in den Vorlagen "Finanzeinheiten"

Finanzeinheit

Diese Vorlage wird in Regeln von CGIS (Curam Global Income Support) verwendet und stellt eine Finanzeinheit dar.

Kategorie 'Finanzeinheit'

Diese Vorlage stellt ein neue Kategorie einer Finanzeinheit für eine Finanzeinheit in CGIS-Regeln dar.

Mitglied von Finanzeinheit

Diese Vorlage stellt ein neues Mitglied einer Finanzeinheit für eine Finanzeinheit in CGIS-Regeln dar.

Referenzinformationen zu Regelelementen in den Vorlagen "Unterstützung bei Nahrung"

Einheit für Unterstützung bei Nahrung

Diese Vorlage stellt eine neue Einheit für die Unterstützung bei Nahrung in CGIS-Regeln dar.

Ein Diagramm für die Unterstützung bei Nahrung muss zunächst konfiguriert werden, bevor es vollständig bearbeitet werden kann. Bestimmte Assistenten des Editors machen es erforderlich, dass die Konfiguration festgelegt wird, bevor sie spezielle Optionen für die Unterstützung bei Nahrung bereitstellen (z. B. Assistent für Verweise). Die Konfiguration eines Diagramms für die Unterstützung bei Nahrung kann jederzeit geändert werden.

Kategorie "Unterstützung bei Nahrung" für alleinstehende Person

Diese Vorlage stellt eine neue Kategorie "Unterstützung bei Nahrung" für eine alleinstehende Person in CGIS-Regeln dar.

Kategorie 'Unterstützung bei Nahrung' für mehrere Personen

Diese Vorlage stellt eine neue Kategorie "Unterstützung bei Nahrung" für mehrere Personen in CGIS-Regeln dar.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 81. Popup-Menüoptionen für Kategorie "Unterstützung bei Nahrung" für mehrere Personen

Name	Beschreibung
Verpflegungsgruppe entfernen	Mit dieser Option können Sie eine Verpflegungsgruppe aus dem Regelelement **Kategorie "Unterstützung bei Nahrung" für mehrere Personen** entfernen.
Verwandte entfernen	Mit dieser Option können Sie Verwandte aus dem Regelelement **Kategorie "Unterstützung bei Nahrung" für mehrere Personen** entfernen.
Verwerfen entfernen	Mit dieser Option können Sie "Verwerfen" aus dem Regelelement **Kategorie "Unterstützung bei Nahrung" für mehrere Personen** entfernen.
Haushaltsvorstand entfernen	Mit dieser Option können Sie einen Haushaltsvorstand aus dem Regelelement **Kategorie "Unterstützung bei Nahrung" für mehrere Personen** entfernen.
Optionale Mitglieder entfernen	Mit dieser Option können Sie ein optionals Mitglied aus dem Regelelement **Kategorie "Unterstützung bei Nahrung" für mehrere Personen** entfernen.

Mitglieder der Verpflegungsgruppe

Diese Vorlage stellt neue Mitglieder für Verpflegungsgruppen bei der Unterstützung bei Nahrung in CGIS-Regeln dar.

Verwandte

Diese Vorlage stellt neue Verwandte bei der Unterstützung bei Nahrung in CGIS-Regeln dar.

Verwerfen
Diese Vorlage stellt einen neuen Ausschluss bei der Unterstützung bei Nahrung in CGIS-Regeln dar.

Mitglieder der Haushaltseinheit
Diese Vorlage stellt ein neues Mitglied der Haushaltseinheit für die Unterstützung bei Nahrung in CGIS-Regeln dar.

Optionale Mitglieder
Diese Vorlage stellt neue optionale Mitglieder für die Unterstützung bei Nahrung in CGIS-Regeln dar.

Ausnahmen
Diese Vorlage stellt neue Ausnahmen für die Unterstützung bei Nahrung in CGIS-Regeln dar.

Referenzinformationen zu Regelelementen in den Vorlagen "Entscheidungstabelle"

Entscheidungstabelle
Das Element "Entscheidungstabelle" stellt eine grafische Darstellung des Ausdrucks "decision table" bereit. Wenn ein Element "Entscheidungstabelle" auf eine Regel oder ein Attribut gezogen wird, wird der Assistent **Entscheidungstabelle erstellen** angezeigt. In diesem Assistenten muss der Benutzer die folgenden Optionen festlegen:

- Anzahl der Zeilen: Diese Option legt die Anzahl der Zeilen fest, die in der Entscheidungstabelle enthalten sind. Die maximale Anzahl der Zeilen beträgt 99.
- Ergebnistyp: Diese Option legt den Rückgabetyp der Entscheidungstabelle fest. Der Ergebnistyp muss mit dem Ergebnistyp der Regel oder des Attributs identisch sein, in der/dem die Entscheidungstabelle enthalten ist.
- Regelklasse: Mit dieser Option kann der Benutzer die aktuelle Regelklasse auswählen oder Regelklassen ändern.

Nachdem der Benutzer die Schaltfläche *Weiter* ausgewählt hat, kann er eine vorhandene Regel verwenden oder die Erstellung einer neuen Regel auswählen.

In der folgenden Tabelle sind spezielle Eigenschaften für dieses Element aufgeführt:

Tabelle 82. Eigenschaften des Elements "Entscheidungstabelle"

Name	Beschreibung
Klasse	Der Name der Regelklasse, die als Typ ausgewählt ist. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Attribut	Der Name des Attributs, das die Entscheidungstabelle enthält. Diese Eigenschaft wird auf der Registerkarte "Geschäftlich" angezeigt.
Einzelnes Element	Aus dem Element wird nur ein einziges Element zurückgegeben. Sie wird auf der Registerkarte "Technisch" angezeigt.
Verhalten, wenn keine Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler oder Rückgabe von Null) zurückgegeben, wenn keine Elemente gefunden wurden. Diese Option ist aktiv, wenn das Kästchen "Einzelnes Element" ausgewählt ist.

Tabelle 82. Eigenschaften des Elements "Entscheidungstabelle" (Forts.)

Name	Beschreibung
Verhalten, wenn mehrere Elemente gefunden wurden	Es wird eines der hier angegebenen Ergebnisse (Fehler, Rückgabe von Null, Rückgabe des ersten Elements, Rückgabe des letzten Elements) zurückgegeben, wenn mehrere Elemente gefunden wurden. Diese Option ist aktiv, wenn das Kästchen "Einzelnes Element" ausgewählt ist.

In der folgenden Tabelle sind spezielle Popup-Menüoptionen für dieses Element aufgeführt:

Tabelle 83. Popup-Menüoptionen für "Entscheidungstabelle"

Name	Beschreibung
Entscheidungstabelle bearbeiten	Nach Auswahl dieser Option können Sie den Ergebnistyp oder das zugehörige Attribut ändern.
Neue Zeile hinzufügen	Diese Option fügt eine neue Zeile zur Entscheidungstabelle hinzu.

Best Practices für CER

Die Berücksichtigung dieser Best Practices erleichtert Ihnen das Entwickeln, Testen und Verwalten von Regelwerken.

Regelattribut `description`

Jede Regelklasse übernimmt letzlich aus einer so genannten "Stammregelklasse" von CER. Diese Stammklasse enthält ein Regelattribut `description`, für das es eine Standardimplementierung gibt, die jedoch nicht besonders hilfreich ist.

Der Wert des Attributs `description` eines Regelobjekts wird im RuleDoc und ebenfalls von der Methode `toString` für ein Element `RuleObject` ausgegeben (diese Methode wird in vielen integrierten Entwicklungsumgebungen für Java verwendet, wenn Sie auf eine Variable "klicken"). Ein aussagekräftiger Wert für das Attribut `description` ist für das Verständnis des Verhaltens einer Regelgruppe möglicherweise unabdingbar.

Sie sollten die Standardberechnung für das Attribut `description` überschreiben, indem Sie für jede Regelklasse explizit ein Attribut `description` erstellen. Im CER-Editor können Sie ein Attribut `description` genauso wie ein normales Attribut für eine Regelklasse erstellen. Der CER-Regelwerkvalidierer gibt eine Warnung aus, wenn eine Regelklasse vorhanden ist, die kein Regelattribut `description` definiert (oder aus einer anderen definierten Regelklasse übernimmt).

Das Attribut `description` ist eine lokalisierbare Nachricht und seine Berechnung kann (wie bei anderen Regelattributen) so einfach oder so komplex wie benötigt sein.

Das folgende Beispiel zeigt ein Regelwerk, bei dem einige Regelklassen eine Implementierung des Attributs `description` angeben:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_description"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
```

```
<Class name="Person">

  <Attribute name="firstName">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="lastName">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- Override the default description -->
  <Attribute name="description">
    <type>
      <javaclass name="curam.creole.value.Message"/>
    </type>
    <derivation>
      <!-- Concatenate the person's first and last names -->
      <concat>
        <fixedlist>
          <listof>
            <javaclass name="Object"/>
          </listof>
          <members>

            <reference attribute="firstName"/>
            <String value=" "/>
            <reference attribute="lastName"/>
          </members>
        </fixedlist>
      </concat>
    </derivation>
  </Attribute>

</Class>

<Class name="Income">
  <!-- The person to which this
       income record relates. -->
  <Attribute name="person">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
  <Attribute name="startDate">
    <type>
      <javaclass name="curam.util.type.Date"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
  <Attribute name="amount">
    <type>
      <javaclass name="Number"/>
```

```
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Override the default description -->
    <Attribute name="description">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Concatenate the person's description and the start
 date-->
        <concat>
          <fixedlist>
            <listof>
              <javaclass name="Object"/>
            </listof>
            <members>

              <reference attribute="description">
                <reference attribute="person"/>
              </reference>
              <!-- In a real rule set, this description would use
                   a <ResourceMessage> to avoid hard-coded
                   single-language Strings.  -->
              <String value="'s income, starting on "/>
              <reference attribute="startDate"/>
            </members>
          </fixedlist>
        </concat>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Benefit">
    <!-- NB no override of <description>; the CER rule set validator
         will issue a warning, and rule objects of this class will
         be more difficult to understand in RuleDoc or a Java
         integrated development environment. -->
    <!-- The person to which this
         benefit record relates. -->
    <Attribute name="person">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>
```

Die nachfolgende Testklasse erstellt einige Regelobjekte (`Person`, `Income` und `Benefit`):

```
package curam.creole.example;

import java.io.File;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.SessionDoc;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.ruleclass.Example_description.impl.Benefit;
import
 curam.creole.ruleclass.Example_description.impl.Benefit_Factory;
import curam.creole.ruleclass.Example_description.impl.Income;
import
 curam.creole.ruleclass.Example_description.impl.Income_Factory;
import curam.creole.ruleclass.Example_description.impl.Person;
import
 curam.creole.ruleclass.Example_description.impl.Person_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.util.type.Date;

/**
 * Tests the description rule attribute.
 */
public class TestDescription extends TestCase {

  /**
   * Tests the description rule attribute.
   */
  public void testDescriptions() {

    /**
     * Create a new session.
     */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    /**
     * Create a SessionDoc to report on rule objects.
     */
    final SessionDoc sessionDoc = new SessionDoc(session);

    /*
     * Create a Person rule object.
     */
    final Person person =
        Person_Factory.getFactory().newInstance(session);
    person.firstName().specifyValue("John");
    person.lastName().specifyValue("Smith");

    /*
     * Create an Income rule object.
     */
    final Income income =
        Income_Factory.getFactory().newInstance(session);
    income.person().specifyValue(person);
    income.amount().specifyValue(123);
    income.startDate().specifyValue(Date.fromISO8601("20070101"));

    /*
     * Create a Benefit rule object.
     */
```

```
    final Benefit benefit =
        Benefit_Factory.getFactory().newInstance(session);
    benefit.person().specifyValue(person);
    benefit.amount().specifyValue(234);

    /*
     * The .toString method evaluates the description rule
     * attribute
     */
    System.out.println(person.toString());

    /*
     * println calls an object's toString method to print it.
     */
    System.out.println(income);

    /*
     * The benefit rule class does not provide an implementation of
     * the description rule attribute, so we'll get a default
     * description here
     */
    System.out.println(benefit);

    /*
     * Write out SessionDoc for this session.
     */
    sessionDoc.write(new File("./gen/sessiondoc"));
>
  }

}
```

Bei der Ausführung des Tests wird die folgende Ausgabe erzeugt, die die Beschreibungen der Regelobjekte enthält:

```
John Smith
John Smith's income, starting on 01/01/07 00:00
Undescribed instance of rule class 'Benefit', id '3'
```

Am Ende des Tests werden die Regelobjekte der Sitzung als SessionDoc ausgegeben. Die allgemeine SessionDoc-Zusammenfassung zeigt die erstellten Regelobjekte und listet die Beschreibungen der einzelnen Regelobjekte auf:

Example_description

Generated: 13-Jul-2012 11:52:43

External rule objects

Details	Type	Description	Action
details	Example_description.Benefit	Undescribed instance of rule class 'Benefit', id '3'	Created during this session
details	Example_description.Income	John Smith's income, starting on 01/01/07 00:00	Created during this session
details	Example_description.Person	John Smith	Created during this session

Internal rule objects

Details	Type	Description	Action

Abbildung 16. SessionDoc mit Werten des Attributs `description` von Regelobjekten

Die Beschreibung für das Regelobjekt `Benefit` ist die Standardbeschreibung. Falls keine sinnvolle Implementierung des Attributs `description` gegeben ist, muss ein Benutzer, der das SessionDoc liest, möglicherweise zum SessionDoc für das Regelobjekt `Benefit` navigieren, damit die Angaben für ihn einen Sinn ergeben:

Rule Object

Generated: 13-Jul-2012 11:52:43

Type

Example_description.Benefit

Description

Undescribed instance of rule class 'Benefit', id '3'

Creation

Created externally

Action during this session

Created during this session

Attributes

Name	Declared type	State	Value	Derivation	Depends on	Used by
amount	Number	SPECIFIED	234	• Specified externally.	*None*	*None*
description	Message	CALCULATED	Undescribed instance of rule class 'Benefit', id '3'	• Default rule object description.	*None*	*None*
person	Person	SPECIFIED	John Smith	• Specified externally.	*None*	*None*

Abbildung 17. SessionDoc für ein Regelobjekt ohne überschriebenes Attribut `description`

Der abschließende Screenshot zeigt, wie eine integrierte Entwicklungsumgebung (z. B. das im gezeigten Beispiel verwendete Eclipse) die Methode `toString` eines Objekts beim Debugging verwendet, die (für Regelobjekte) den Wert des Attributs `description` berechnet:

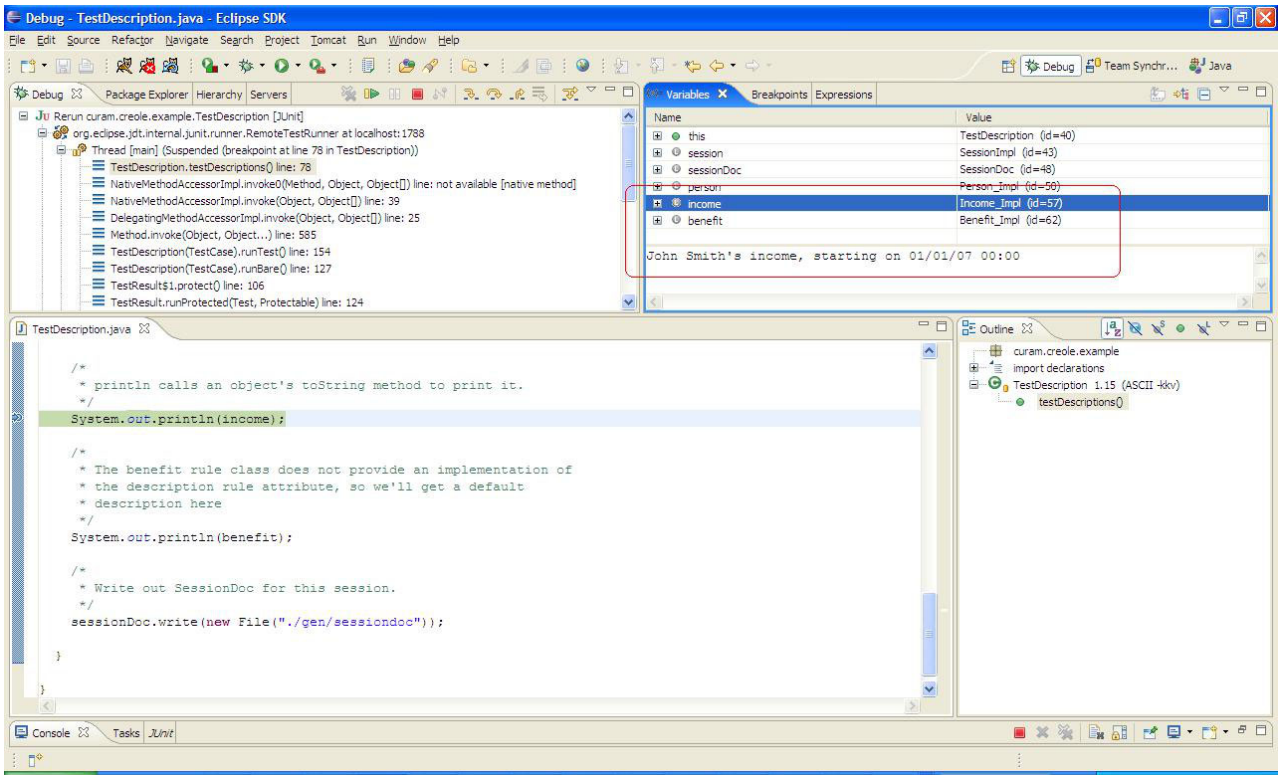

Abbildung 18. Verwendung des Attributs `description` in einer integrierten Entwicklungsumgebung

Tipp: Denken Sie daran, dass ein Attribut `description` die Aufgabe hat, eine Regelobjektinstanz und nicht die Regelklasse selbst zu beschreiben.

Insbesondere sollte die Berechnung des Regelattributs `description` Daten einschließen, die eine einfache Unterscheidung der verschiedenen Regelobjektinstanzen einer Regelklasse ermöglichen.

Regelwerk schnell einsatzbereit machen

Es kann hilfreich sein, ein CER-Regelwerk schnell einsatzbereit zu machen, indem bestimmte Tasks für die Regelentwicklung zurückgestellt werden.

In diesem Zusammenhang sind eines oder mehrere der folgenden Abkürzungsverfahren denkbar:

- Erstellen Sie für jedes Geschäftskonzept leere Regelklassen (also Klassen ohne Regelattribute). Die Regelattribute können Sie später hinzufügen. Sie können beispielsweise mit dem CER-Editor eine leere Regelklasse erstellen und später ein Attribut zu dieser Klasse hinzufügen.
- Erstellen Sie fest codierte Ableitungen für Regelattribute. Die Geschäftsregeln können Sie später hinzufügen. Indem Sie beispielsweise die Berechnung eines Regelattributs `isEligible` mit `<true>` deklarieren, können Sie Tests schreiben und/oder CER in Ihre eigene Anwendung integrieren. Die fest codierte Ableitung für "immer auswählbar" können Sie später durch die realen Geschäftsregeln ersetzen. Sie können beispielsweise im CER-Editor das Regelelement "Boolesch" (Standardwert ist "Wahr", also "true") auf ein Regelattribut `isEligible` ziehen.
- Erstellen Sie Nachrichten als einfache fest codierte Zeichenfolgen in einer einzelnen Ländereinstellung. Sie können die Zeichenfolgen später in lokalisierbare

Nachrichten konvertieren. Sie können beispielsweise im CER-Editor das Regelelement "Ressourcennachricht" oder "XML-Nachricht" auf ein Regelattribut ziehen.

- Verwenden Sie Zeichenfolgewerte anstelle von Anwendungscodetabellenwerten. Sie können die Zeichenfolgen später in Codes konvertieren (und die Regeln für ihren Test aktualisieren). Sie können beispielsweise im CER-Editor das Regelelement "Zeichenfolge" auf ein Regelattribut ziehen.

Wichtig: Die Verwendung dieser Abkürzungsverfahren bedeutet *nicht*, dass Sie später auf eine exaktere Ausarbeitung verzichten können, sondern stellt lediglich einen Teil der Bearbeitung zurück, bis das Regelwerk "einsatzbereit" ist.

Sie sollten *nicht* mit der Erstellung von Tests für Ihre Regeln warten. Insbesondere bei Verwendung dieser Abkürzungsverfahren sorgt ein gutes Gerüst von Regelwerktests dafür, dass bestimmte Fehlerarten vermieden werden, sobald Sie mit der Nachbearbeitung der zurückgestellten Elemente beginnen. Erstellen Sie die Tests direkt beim Schreiben der Regeln.

Ebenso kann es eine große Versuchung sein, mit der Erstellung von Regelattributen `description` zu warten. In den frühen Phasen des Regelwerksentwurfs kann eine solche Zurückstellung jedoch ein Sparen am falschen Ende sein, da die Regelattribute `description` eine wertvolle Unterstützung beim Debug von Regeln darstellen, die mit einem relativ geringen Aufwand verbunden ist.

Benennung von Regelelementen

CER-Regelattribute sollten so benannt werden, dass ihre geschäftliche Bedeutung ersichtlich wird. Orientieren Sie sich bei der Benennung eines Regelattributs an dem von ihm bereitgestellten Ergebnis und nicht an dem Verfahren, mit dem dieses Ergebnis erzielt wird.

Ein Regelwerkname kann eine Kombination aus Buchstaben (Standardzeichenbereich A bis Z ohne Zeichen mit Akzent), Zahlen und Unterstreichungszeichen sein. Ein Regelwerkname sollte mit einem Großbuchstaben beginnen.

Ein Klassenname kann eine Kombination aus Buchstaben (Standardzeichenbereich A bis Z ohne Zeichen mit Akzent), Zahlen und Unterstreichungszeichen sein. Ein Klassenname sollte mit einem Großbuchstaben beginnen.

Ein Attributname kann eine Kombination aus Buchstaben (Standardzeichenbereich A bis Z ohne Zeichen mit Akzent), Zahlen und Unterstreichungszeichen sein. Ein Attributname sollte mit einem Kleinbuchstaben beginnen.

Tipp: Verwenden Sie bei der Benennung von Regelelementen die so genannte "Kamelschreibweise" (CamelCase).

Bei dieser Schreibweise werden die einzelnen Bestandteile von Komposita oder Ausdrücken ohne Leerzeichen miteinander verbunden, wobei als erstes Zeichen der einzelnen Bestandteile im Kompositum jeweils ein Großbuchstabe verwendet wird und die gesamte Angabe entweder mit einem Groß- oder einem Kleinbuchstaben beginnt.

Anmerkung: Jedes Regelelement besitzt einen einzigen Namen, der nicht lokalisiert werden kann. Verwenden Sie für lokalisierbare Beschreibungen die Anmerkung "Label" (Beschriftung). Eine entsprechende Erläuterung finden Sie im Abschnitt „Lokalisierung der Beschreibungen von CER-Regelartefakten” auf Seite 12.

Situationen für die Verwendung des Ausdrucks 'reference'

Die korrekte Verwendung des Ausdrucks `reference` ist für die Struktur eines guten CER-Regelwerks von wesentlicher Bedeutung. Mit der Verwendung des Ausdrucks `reference` geht die Erstellung der geeigneten Anzahl von Regelattributen einher. Der CER-Editor bietet unterschiedliche Typen von Szenarios für die Erstellung und Verwendung des Regelelements "Verweis". Weitere Informationen enthält der Abschnitt „Regel" auf Seite 119, für das Element "Regel".

Das Erreichen der richtigen Balance (zwischen zu wenigen und zu vielen Verwendungen von `reference`) ist möglicherweise eher eine Kunst als eine Wissenschaft, kann jedoch durch einige allgemeine Richtlinien unterstützt werden:

- Falls Sie feststellen, dass einige Ausdrücke sehr tief verschachtelt oder auf andere Weise komplex sind, werden möglicherweise zu wenig Verweise verwendet. In einer solchen Situation kann es sinnvoll sein, komplexe Ausdrücke durch die Erstellung von Regelattributen für einen sinnvollen Block von Ausdrücken aufzuteilen und einen Ausdruck `reference` für das neue Regelattribut zu verwenden.
- Falls Ihre Anforderungen ein striktes Konzept oder eine strikte Berechnung vorsehen, das/die durch ein Regelattribut nicht vollständig umgesetzt wird, kann es sinnvoll sein, ein entsprechendes Regelattribut zu erstellen.
- Falls verschiedene Ausdrücke dieselben Arten von Berechnungen wiederholen, könnte das Regelwerk von der Erstellung eines Regelattributs profitieren, das die allgemeine Logik implementiert.
- Falls sich die Benennung eines Regelattributs als kompliziert herausstellt, ist das Regelattribut unter Umständen eine unnötige Kapselung von Logik und im Regelwerk werden möglicherweise zu viele Ausdrücke `reference` verwendet. In einer solchen Situation kann es sinnvoll sein, das Regelattribut zu entfernen und seine Ableitung an den Stellen zu integrieren, an denen es verwendet wird. Dies gilt insbesondere dann, wenn nur eine einzige andere Berechnung auf das Regelattribut verweist.

RuleDoc verwenden

Ein relativ kompaktes CER-Regelwerk kann direkt im CER-Editor geöffnet und dort problemlos in seiner Gänze verstanden und nachvollzogen werden.

Bei komplexeren Regelwerken können Sie das CER-Tool für RuleDocs verwenden, um Einblicke in deren Verhalten und Struktur zu erhalten.

Informationen zum Generieren von RuleDocs aus CER-Regelwerken finden Sie im Abschnitt „RuleDoc" auf Seite 20.

Allgemeine Regeln normalisieren

Während der Entwicklung eines Regelwerks stellen Sie unter Umständen fest, dass bestimmte Regeln in unterschiedlichen Teilen der Regelwerkfunktionen ähnlich sind.

Es kann sinnvoll sein, solche allgemeinen Regeln zu ermitteln und zu zentralisieren.

Grob gesagt gibt es bei der Zentralisierung allgemeiner Regeln zwei Möglichkeiten:

- **Übernahme**

  Mithilfe der CER-Unterstützung für die Implementierungsübernahme können Sie eine Regelklasse durch eine andere erweitern lassen. Der CER-Editor stellt

den Übernahmemechanismus für das Entwerfen der Regel bereit. Weitere Informationen enthält der Abschnitt „Eigenschaften für Regelklassen” auf Seite 117, für das Element "Erweitert".

- **Einschluss**

  Mithilfe der CER-Unterstützung für die Erstellung neuer Regelobjekte aus Regeln können Sie von einer Regelklasse neue Instanzen einer anderen Regelklasse erstellen lassen, wenn diese benötigt werden. Der CER-Editor stellt den Einschlussmechanismus für das Entwerfen der Regel bereit. Weitere Informationen finden Sie unter "Assistent zum Ändern von Regelwerken und Regelklassen" im Abschnitt „Assistenten für Regelelemente” auf Seite 118.

Bisweilen ist es nicht ganz einfach, den Mechanismus zu bestimmen, der beim Zentralisieren von allgemeinen Regeln verwendet werden sollte. Generell sollte die Übernahme nur mit großer Sorgfalt und nur dann eingesetzt werden, wenn die Unterregelklasse ein Geschäftskonzept darstellt, das genuin eine Instanz des von der Oberklasse dargestellten Geschäftskonzepts *ist*. CER unterstützt insbesondere keine Mehrfachübernahme.

Ein Beispiel für eine Übernahme ist eine Situation, in der eine Person Ressourcen besitzt und jede Ressource ein Gebäude oder ein Fahrzeug sein kann. Die Regelklassen `Building` (= Gebäude) und `Vehicle` (Fahrzeug) erweitern jeweils eine abstrakte Regelklasse `Resource`. Ziehen Sie in diesem Zusammenhang auch die Liste im Abschnitt „Regelklassen” auf Seite 42, hinzu.

Ein Einschluss sollte verwendet werden, wenn das durch eine Regelklasse dargestellte Geschäftskonzept eine Instanz des durch die eingeschlossene Regelklasse dargestellten Geschäftskonzepts *enthält*.

Ein Beispiel für einen Einschluss ist eine Situation, in der auf eine Person viele verschiedene Altersbereichstests angewendet werden. Die Regelklasse `Person` erstellt viele Instanzen von `AgeRangeTest`.

Falls Sie feststellen, dass verschiedene Regelwerke ähnliche Regelklassen enthalten, können Sie (seit Cúram Version 6) mithilfe der CER-Funktionalität ein Regelwerk auf Artefakte in einem anderen Regelwerk verweisen lassen. Ordnen Sie die allgemeinen Regelklassen in einem oder mehreren allgemeinen Regelwerken an und verwenden Sie für nicht allgemeine Regelklassen andere Regelwerke.

Nicht verwendete Regeln entfernen

Beim Zentralisieren von allgemeinen Regeln stellen Sie möglicherweise fest, dass auf einige Regelattribute nicht mehr aus anderen Berechnungen verwiesen wird und diese Attribute (im Rahmen von "Aufräumarbeiten") aus dem Regelwerk entfernt werden können.

CER unterstützt das Melden von Regelattributen, auf die nicht von anderen Berechnungen im Regelwerk verwiesen wird und die daher potenziell entfernt werden können. Zum Löschen von Regelklassen und Regelattributen können Sie auch den CER-Editor verwenden. Weitere Informationen finden Sie im Abschnitt „Technische Ansicht” auf Seite 108.

Angaben über die Ausführung des CER-Berichts für nicht verwendete Attribute enthält der Abschnitt „Nicht verwendete Regelattribute” auf Seite 30.

Warnung: Es ist durchaus möglich, dass es sich bei einem Regelattribut um ein "übergeordnetes" Regelattribut handelt, auf das nur durch Client-Code verwiesen

wird. Solche Attribute werden in diesem Bericht möglicherweise als nicht verwendet gemeldet. Sie sollten jedoch nur dann scheinbar nicht verwendete Regelattribute aus Ihrem Regelwerk entfernen, wenn Sie sicher sind, dass kein Client-Code oder Test von diesen Attributen abhängig ist.

Reihenfolge von Deklarationen

Die Reihenfolge der Deklarationen in Ihren Regelwerken wirkt sich grundsätzlich nicht auf das Verhalten aus.

Reihenfolge von Regelklassen in einem Regelwerk

Sie können die Regelklassen in Ihrem Regelwerk so anordnen, wie es für Sie am sinnvollsten ist. Eine Umordnung der Regelklassen hat keinen Einfluss auf das Verhalten von Regelwerken.

Reihenfolge von berechneten Regelattributen in einer Regelklasse

Auch die *berechneten* Regelattribute können Sie in Ihren Regelklassen in jeder gewünschten Reihenfolge anordnen. Eine Umordnung von *berechneten* Regelattributen hat keinen Einfluss auf das Verhalten von Regelwerken.

Reihenfolge von initialisierten Attributen in einer Regelklasse

Bei jeder Erstellung einer Regelobjektinstanz für eine Klasse (ob nun innerhalb von Regeln mit dem Ausdruck `create` oder mittels Java-Code unter Verwendung der generierten Regelklassen oder der dynamischen Regel-API) müssen die Werte aller initialisierten Attribute *in der Reihenfolge angegeben werden, in der sie in der Regelklasse definiert sind*.

Warnung: Infolgedessen sollten Sie die Umordnung der Attribute in einem Block `Initialization` vermeiden, sofern Sie nicht auch alle Stellen (in Regeln oder Java-Code) überarbeiten wollen, an denen Regelobjektinstanzen der Regelklasse erstellt werden.

Reihenfolge von booleschen Bedingungen

Die Reihenfolge von booleschen Bedingungen in einem Ausdruck `all` oder `any` hat keinen Einfluss auf den logischen Wert des Ergebnisses.

Zur Laufzeit wird jedoch die Verarbeitung von booleschen Bedingungen gestoppt, sobald ein Ergebnis bestätigt wird. Sie sollten daher boolesche Bedingungen so anordnen, dass der Ausdruck `all` oder `any` sein Ergebnis möglichst schnell erzielt.

Dies bedeutet Folgendes:

- Bei Ausdrücken `all` sollten boolesche Bedingungen, die wahrscheinlich mit dem Ergebnis *false* ausgewertet werden, weiter vorne in der Liste stehen.
- Bei Ausdrücken `any` sollten boolesche Bedingungen, die wahrscheinlich mit dem Ergebnis *true* ausgewertet werden, weiter vorne in der Liste stehen.

Im CER-Editor können Sie die Reihenfolge der Regelelemente "Boolesch" im Regelelement "Beliebig" ändern. Ordnen Sie beispielsweise alle Regelelemente "Boolesch" mit dem Wert "true" zuerst an.

Regelobjekte erstellen

Das Verfahren für die Erstellung von Regelobjekten muss sorgfältig gewählt werden.

Insbesondere sollten Sie bei Verwendung von CER in Ihren eigenen Anwendungen bereits früh festlegen, welche Regelobjekte vom Anwendungscode und welche Regelobjekte von den Regeln erstellt werden sollen.

Weitere Details enthält der Abschnitt „Externe und interne Regelobjekte" auf Seite 35.

Regelobjekte anstelle von IDs übergeben

Wenn Sie ein internes Regelobjekt mit dem Ausdruck "create" (siehe „create" auf Seite 191) erstellen, können Sie unter Verwendung des Initialisierungsblocks (Initialize) und/oder von Elementen `specify` Daten an das neue Regelobjekt übergeben.

Falls die übergebenen Daten einen Verweis auf externe Daten enthalten, für die es eine ID gibt (z. B. einen durch eine Fall-ID (caseID) gekennzeichneten Fall), kann es sinnvoll sein, die Regelklasse für das erstellte Regelobjekt so zu entwerfen, dass sie nicht mit einem ID-Wert, sondern mit einem Regelobjekt initialisiert wird, von dem die Daten dargestellt werden.

Die Verwendung eines solchen Regelobjekts für die Initialisierung kann die Typsicherheit Ihrer Daten erhöhen, weil sie verhindern kann, dass andere Regelentwickler eigene interne Regelobjekte für dieselbe Regelklasse erstellen, jedoch versehentlich eine ID übergeben wird, die eine andere Art von externen Daten darstellt.

Die Übergabe eines falschen ID-Typs verursacht wahrscheinlich, dass Regeln während der *Laufzeit* fehlschlagen (weil beispielsweise bei dem Versuch, ein Regelobjekt für diese ID zu konvertieren, keine zugrunde liegenden Daten gefunden werden). Die Übergabe eines Regelobjekts für die Typsicherheit ermöglicht es hingegen, dass der CER-Regelwerkvalidierer das Problem bereits zur *Entwurfszeit* erkennt.

Statische Methoden entwickeln

CER unterstützt eine Vielzahl von Ausdrücken, mit denen wahrscheinlich die benötigten Berechnungen bereitgestellt werden können.

Für Fälle, in denen eine Geschäftsberechnung nicht unter Verwendung der CER-Ausdrücke implementiert werden kann, unterstützt CER den Ausdruck `call`, damit Sie aus Ihrem Regelwerk heraus eine statische Methode für eine angepasste Java-Klasse aufrufen können. Der CER-Editor enthält einige Regelelemente (z. B. "Aufrufen"), damit Benutzer eine statische Methode für eine angepasste Java-Klasse definieren können. Weitere Informationen enthält der Abschnitt „Aufruf" auf Seite 137, für das Element "Aufruf".

Das folgende Beispiel zeigt ein Regelwerk, das eine Java-Methode aufruft:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_StaticMethodDevelopment"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Income">

    <Attribute name="paymentReceivedDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
```

```
    </Attribute>

    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Attribute name="incomes">
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="mostRecentIncome">
      <type>
        <ruleclass name="Income"/>
      </type>
      <derivation>
        <!-- In early development, the method
             identifyMostRecentIncome_StronglyTyped would be used.

             In practice, you would not maintain two versions of
             each method; you would simply weaken the argument
             and return types, and keep a single method.
        -->

        <call class="curam.creole.example.BestPracticeDevelopment"
          method="identifyMostRecentIncome_WeaklyTyped">
          <type>
            <ruleclass name="Income"/>
          </type>
          <arguments>
            <this/>
          </arguments>
        </call>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>
```

Beim Entwickeln einer statischen Methode können Sie die von CER generierten Java-Klassen als Ausgangspunkt für die Argumenttypen und/oder Rückgabetypen der Methode verwenden:

```
package curam.creole.example;

import java.util.List;

import curam.creole.execution.session.Session;
import
 curam.creole.ruleclass.Example_StaticMethodDevelopment.impl.Income;
import
 curam.creole.ruleclass.Example_StaticMethodDevelopment.impl.Person;
```

```
public class BestPracticeDevelopment {

  /**
   * Identifies a person's most recent income.
   *
   * Note that this calculation can be performed using CER
   * expressions, but is shown here in Java just to illustrate the
   * use of generated types when developing static Java methods for
   * use with CER.
   *
   * This method is suitable only for use in development; for
   * production, see the
   * {@linkplain #identifyMostRecentIncome_WeaklyTyped} below.
   *
   * In practice, you would not maintain two versions of each
   * method; you would simply weaken the argument and return types,
   * and keep a single method.
   */
  public static Income identifyMostRecentIncome_StronglyTyped(
      final Session session, final Person person) {

    Income mostRecentIncome = null;
    final List<? extends Income> incomes =
        person.incomes().getValue();
    for (final Income current : incomes) {
      if (mostRecentIncome == null
          || mostRecentIncome.paymentReceivedDate().getValue()
              .before(current.paymentReceivedDate().getValue())) {
        mostRecentIncome = current;
      }
    }

    return mostRecentIncome;
  }
}
```

Sobald die Java-Methode wie gewünscht funktioniert, müssen Sie die Typen so abschwächen, dass anstelle der generierten Java-Klassen (die nicht in einer Produktionsumgebung eingesetzt werden sollten) dynamische Regelobjekte verwendet werden:

```
/**
   * Identifies a person's most recent income.
   *
   * Note that this calculation can be performed using CER
   * expressions, but is shown here in Java just to illustrate the
   * use of generated types when developing static Java methods for
   * use with CER.
   *
   * This method is suitable for use in production; for initial
   * development, see
   * {@linkplain #identifyMostRecentIncome_StronglyTyped} above.
   *
   * In practice, you would not maintain two versions of each
   * method; you would simply weaken the argument and return types,
   * and keep a single method.
   */
  public static RuleObject identifyMostRecentIncome_WeaklyTyped(
      final Session session, final RuleObject person) {

    RuleObject mostRecentIncome = null;
    final List<? extends RuleObject> incomes =
        (List<? extends RuleObject>) person.getAttributeValue(
            "incomes").getValue();
    for (final RuleObject current : incomes) {
      if (mostRecentIncome == null
```

```
            || ((Date) mostRecentIncome.getAttributeValue(
                "paymentReceivedDate").getValue())
                .before((Date) current.getAttributeValue(
                    "paymentReceivedDate").getValue())) {
          mostRecentIncome = current;
        }
      }

      return mostRecentIncome;

    }
```

Wichtig: Wenn Sie die Implementierung einer statischen Methode ändern, weiß CER *nicht* automatisch, dass Attributwerte neu berechnet werden müssen, die unter Verwendung der alten Version der statischen Methode berechnet wurden.

Sobald eine statische Methode in einer Produktionsumgebung für gespeicherte Attributwerte verwendet wurde, sollten Sie anstelle einer Änderung der Implementierung eine neue statische Methode erstellen (mit der erforderlichen neuen Implementierung) und Ihre Regelwerke so ändern, dass die neue statische Methode verwendet wird. Wenn Sie Ihre Regelwerkänderungen, die auf die neue statische Methode verweisen, veröffentlichen, berechnet CER automatisch alle Instanzen des betroffenen Attributwerts neu.

Für jede aus CER heraus aufgerufene statische Methode (oder Methode für Eigenschaften) gilt Folgendes:

- Sie darf *nur* von den an sie übergebenen Daten abhängig sein; das Verhalten muss also deterministisch sein und auf den Eingabeparametern basieren. Sie darf keine Daten aus anderen Quellen (z. B. Datenbank, Umgebungsvariablen oder globale Variablen) abrufen, da CER nicht feststellen kann, dass solche Daten gegebenenfalls geändert wurden und Neuberechnungen somit nicht zuverlässig wären.
- Sie darf *keine* Nebeneffekte erzeugen (z. B. Schreiben von Daten in die Datenbank), da CER weder die Reihenfolge, in der die Verarbeitung stattfindet, noch die Häufigkeit der Aufrufe von statischen Methoden oder von Methoden für Eigenschaften garantieren kann.

Häufig auftretende Probleme bei Tests vermeiden

Das Schreiben von JUnit-Tests für Ihre CER-Regelwerke ist im Großen und Ganzen ein unkomplizierter Prozess. Es gibt jedoch eine Reihe von Problemen, die Ihnen bewusst sein sollten.

Der vorliegende Abschnitt zeigt einige Beispiele, die auf den ersten Blick unproblematisch erscheinen, jedoch nicht das erforderliche Verhalten erzeugen.

JUnit-Methoden `assertEquals` vermeiden

JUnit-Tests übernehmen zum Testen Zusicherungsmethoden. Typischerweise sichern Sie zu, dass ein *erwartetes* Ergebnis mit einem *tatsächlichen* Ergebnis übereinstimmt.

Ein häufig auftretendes Problem ist die Verwendung der JUnit-Methode `assertEquals`, um lediglich festzustellen, dass sie bei numerischen Vergleichen nicht ordnungsgemäß funktioniert (und eine etwas verwirrende Fehlernachricht erzeugt).

CER konvertiert vor der Vearbeitung alle Instanzen von `Number` in ein eigenes numerisches Format (gestützt auf `java.math.BigDecimal`), um sicherzustellen, dass

kein Genauigkeitsverlust stattfindet. Diese Konvertierung kann problematisch und nicht intuitiv sein, falls Sie die JUnit-Methode `assertEquals` verwenden.

CER enthält einen Ersatz in einer Helper-Klasse. Verwenden Sie `CREOLETestHelper.assertEquals`. Hiermit werden Zahlen beliebigen Typs ordnungsgemäß verglichen.

Bei anderen Datentypen verhält sich `CREOLETestHelper.assertEquals` genauso wie die JUnit-Methode `assertEquals`. Es ist daher im Allgemeinen ein gutes Verfahren, in allen Tests `CREOLETestHelper.assertEquals` zu verwenden, um mögliche Unklarheiten zu vermeiden (auch dann, wenn diese Verwendung technisch gesehen nicht nötig ist).

```
public void creoleTestHelperNotUsed() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Will not work - getValue returns CER's own numerical handler,
     * but 65 is an integer.
     *
     * JUnit will report the somewhat confusing message:
     * junit.framework.AssertionFailedError: expected:<65> but
     * was:<65>
     *
     * Use CREOLETestHelper.assertEquals instead.
     */
    assertEquals(65, flexibleRetirementYear.ageAtRetirement()
        .getValue());

  }
```

Verwendung von `.getValue()` bedenken

Der CER-Testcodegenerator erstellt für jede Regelklasse eine Java-Schnittstelle sowie eine Zugriffsmethode für die Schnittstelle jedes Regelattributs.

Diese generierte Zugriffsmethode gibt ein CER-Element `AttributeValue` und *nicht* den Wert des Attributs direkt zurück. Um den eigentlichen Wert abzurufen, müssen Sie die Methode `.getValue()` für das Element `AttributeValue` aufrufen.

Falls Sie die Verwendung von `.getValue()` in einem Test vergessen, wird der Test zwar möglicherweise fehlerfrei kompiliert, zeigt jedoch bei der Ausführung nicht das ordnungsgemäße Verhalten.

```
public void getValueNotUsed() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Will not work - ageAtRetirement() is a calculator, not a
     * value.
     *
     * JUnit will report the message:
     * junit.framework.AssertionFailedError: expected:<65> but
```

```
     * was: <Value: 65>
     *
     * Remember to use .getValue() on each attribute calculator!
     */
    assertEquals(65, flexibleRetirementYear.ageAtRetirement());

  }
```

Bitte beachten Sie, dass in diesem Beispiel der Wert von `AttributeValue` als Zeichenfolge "Value: 65" und nicht mit der Zahl 65 (dies wäre die Rückgabe von `.getValue()`) angezeigt wird.

Angabe aller von getesteten Berechnungen benötigten Werte bedenken

In Ihren Tests müssen Sie nur die Werte angeben, auf die während der Regelausführung zugegriffen wird.

Es kann jedoch leicht passieren, dass Sie vergessen, einen Wert anzugeben. Falls CER dann versucht, eine Berechnung auszuführen, findet CER ein Attribut, dessen Ableitung mit `<specified>` angegeben ist, für das jedoch im Testcode kein Wert angegeben war, und meldet eine Reihe von Fehlern:

```
public void valueNotSpecified() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    /**
     * Will not work - a value required for calculation was marked
     * as <specified> but no value was specified for it.
     *
     * CER will report a stack of messages:
     * <ul>
     *
     * <li> Error calculating attribute 'ageAtRetirement' on rule
     * class 'FlexibleRetirementYear' (instance id '1', description
     * 'Undescribed instance of rule class
     * 'FlexibleRetirementYear', id '1'). </li>
     *
     * <li>Error calculating attribute 'retirementCause' on rule
     * class 'FlexibleRetirementYear' (instance id '1', description
     * 'Undescribed instance of rule class
     * 'FlexibleRetirementYear', id '1'). </li>
     *
     * <li>Value must be specified before it is used (it cannot be
     * calculated).</li>
     *
     * </ul>
     *
     *
     * Remember to specify all values required by calculations!
     */
    CREOLETestHelper.assertEquals(65, flexibleRetirementYear
        .ageAtRetirement().getValue());

  }
```

Identischen Wert nicht mehrfach angeben

CER lässt die Angabe eines Wertes zu, der andernfalls berechnet werden würde.

Bei der Verwendung der Strategie `RecalculationsProhibited` gibt CER jedoch einen Laufzeitfehler aus, wenn Sie versuchen, den Wert eines Attributs (für ein bestimmtes Regelobjekt) mehrfach anzugeben. Sobald der Wert einmal angegeben

wurde, kann er nicht mehr geändert werden (ein solches Vorgehen würde dazu führen, dass zuvor ausgeführte Berechnungen nunmehr "falsch" sind).

```
public void valueSpecifiedTwice() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Will not work - the same attribute value cannot be specified
     * a second time.
     *
     * CER will report the message: A value cannot be specified,
     * as the current state of this calculator is 'SPECIFIED'.
     *
     * Do not attempt to specify the same value twice!
     */
    flexibleRetirementYear.retirementCause().specifyValue(
        "Lottery winner");

  }
```

Korrekten Typ für Wert eines Attributs angeben

Für jedes CER-Element `AttributeValue` gibt es eine Methode `specifyValue`, damit der Wert des Attributs angegeben (statt berechnet) werden kann. Die Methode verwendet einen beliebigen Wert als Eingabe, aber wenn Sie den falschen Typ für den Wert angeben, gibt CER einen Laufzeitfehler aus:

```
public void incorrectValueType() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    /**
     * Will not work - retirementCause() expects a String, not a
     * Number.
     *
     * CER will report the message: Attempt to set the value '123'
     * (of type 'java.lang.Integer') on attribute 'retirementCause'
     * of rule class 'FlexibleRetirementYear' (which expects a
     * 'java.lang.String').
     */
    flexibleRetirementYear.retirementCause().specifyValue(123);

  }
```

Anmerkung:

Technische Benutzer fragen sich an dieser Stelle möglicherweise, warum `AttributeValue.specifyValue` nicht die generischen Angaben von Java 5 verwendet, um den Wertetyp einzuschränken, der empfangen werden kann.

Falls Regelklasse A die Regelklasse B erweitert, kann die Klasse A jederzeit die Ableitung jedes Attributs der Klasse B überschreiben. Die Klasse A kann ebenfalls jedes Attribut der Klasse B mit einem restriktiveren Typ neu deklarieren (die Deklaration der Klasse A gibt also ihren Typ als Untertyp des von der Klasse B deklarierten Typs an).

Die generierte Java-Schnittstelle für die Klasse A erweitert die generierte Java-Schnittstelle für die Klasse B. Da die Accessoren Berechnungsfunktionen und nicht den Wertetyp direkt zurückgeben, müssen alle Schnittstellen Platzhalterausdrücke verwenden, damit der Compiler zulässt, dass die Deklaration der Klasse A für den Accessor des Attributs die Deklaration der Klasse B erweitert. Aufgrund der Verwendung einer Platzhaltererweiterung kann `specifyValue` nicht auf einen Typ beschränkt werden und muss daher für den Empfang eines beliebigen Elements `Object` deklariert werden.

Falls in einem anderen Fall die Java-Klasse C zur Erweiterung der Java-Klasse D eingesetzt werden soll, kann die Klasse C einen restriktiveren Typ für einen der Getter der Klasse D definieren, jedoch nicht die Setter der Klasse D auf einen Subtyp beschränken. Die Klasse C muss den Setter von D implementieren und alle nicht erwünschten Werte während der Laufzeit feststellen (auch wenn dies wohl das Liskovsche Substitutionsprinzip verletzen könnte).

Eine weitere Begründung ist die Tatsache, dass beim Einsatz von rein dynamischen Regelobjekten (d. h. in einer interpretierten Sitzung) keine Beschränkung von Werten zur Kompilierzeit möglich ist.

CER verwendt daher seine Kenntnisse über deklarierte Attributtypen, um falsche Werte zur Laufzeit und nicht zur Kompilierzeit festzustellen.

Alle Regelobjekte einer Sitzung vor der Ausführung von Berechnungen mit `getValue` erstellen

In Regelwerktests kann jede beliebige Anzahl von Regelobjekten in einer CER-Sitzung definiert werden, bevor mit der Überprüfung einer beliebigen Anzahl von berechneten Werten für diese Regelobjekte fortgefahren wird.

Sobald Berechnungen gestartet wurden, verhindert jedoch die Strategie `RecalculationsProhibited`, dass Regelobjekte erstellt werden, die den Wert einer vorherigen Berechnung mit dem Ausdruck "readall" (siehe „`readall`” auf Seite 224) ungültig machen.

Sie sollten Ihre Tests so strukturieren, dass die Erstellung aller Testregelobjekte *vor* den Berechnungen (also vor jeder Ausführung einer Methode `getValue`) stattfindet. In der Praxis stellt dies keine übermäßige Einschränkung dar.

Falls Ihr Test versucht, in einer Sitzung ein neues Regelobjekt zu erstellen, nachdem eine Berechnung stattgefunden hat, löst (falls zuvor ausgeführte Berechnungen mit `readall` betroffen sind), die Strategie `RecalculationsProhibited` einen Laufzeitfehler aus:

```
public void newObjectsAddedAfterCalculationsStarted() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Reached statutory retirement age.");

    /**
     * Calculate the age at retirement and test its value
     */
    CREOLETestHelper.assertEquals(65, flexibleRetirementYear
        .ageAtRetirement().getValue());

    /**
     * Create another rule object.
```

```
     */

    /**
     * May not work - new rule objects added to the session once
     * calculations have started could invalidate earlier
     * <code><readall></code> calculations.
     *
     * {@linkplain RecalculationsProhibited} may report the
     * message: "Cannot create new rule objects for this session,
     * because this session has already accepted a calculation
     * request."
     *
     * To avoid this problem, create all your rule objects before
     * attempting any calculations!
     */
    final FlexibleRetirementYear flexibleRetirementYear2 =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

  }
```

Warnung: Falls Ihr Regelwerk *gegenwärtig* keine Ausdrücke des Typs `readall` enthält, können Sie möglicherweise auf eine Umstrukturierung Ihrer Tests verzichten, damit alle Regelobjekte vor der Ausführung von Berechnungen erstellt werden.

Ändern Sie später Ihren Test jedoch so, dass er Ausdrücke `readall` enthält, müssen Sie ihn ab diesem Zeitpunkt umstrukturieren.

Strukturieren Sie zur Vermeidung von Nachbesserungen Ihre Tests immer so, dass alle Regelobjekterstellungen vor dem Beginn von Berechnungen ausgeführt werden.

CER-XML-Wörterverzeichnis

Nachfolgend finden Sie eine Beschreibung der Elemente, aus denen die CER-Sprache für Regelwerke besteht.

Regelwerk

Ein Regelwerk gibt seinen *Namen* an und enthält eine beliebige Anzahl von Elementen `Class` für Regelklassen und/oder Anweisungen `Include`. Ein Regelwerk kann optional Elemente `Annotation` für Anmerkungen enthalten.

Die XML-Struktur eines Regelwerks und seiner Elemente wird durch das CER-Schema `RuleSet.xsd` eingeschränkt. Dieses Schema ist dynamisch aufgebaut, so dass Erweiterungen für CER Ausdrücke und Anmerkungen zum Schema beitragen können.

Das folgende Beispiel zeigt die Gliederung eines Regelwerks:

```
RuleSet
  Anmerkungen (optional)
    ...
    ...
  Include
    ...
  Include
    ...
  ... weitere Anweisungen "Include"
  Klasse
    Anmerkungen (optional)
      ...
```

```
    ...
Initialisierung (optional)
    Attribut
      Anmerkungen (optional)
        ...
        ...
    type
        ...
    Attribut
      type
        ...
    ... weitere initialisierte Attribute
  Attribut
    Anmerkungen (optional)
      ...
      ...
  type
      ...
    derivation
      (Ausdruck)
        Anmerkungen (optional)
          ...
          ...
        (Unterausdruck)
        ...
  Attribut
    type
      ...
    derivation
      ...
    ... weitere berechnete Attribute
... weitere Regelklassen
```

Anweisung `Include`

Es kann hilfreich sein, ein umfangreiches Regelwerk in kleinere Bestandteile zu untergliedern, um die parallele Entwicklung oder die Wiederverwendung zu vereinfachen. Jedes Regelwerk kann Anweisungen `Include` enthalten, um andere Regelwerke und Klassen zu integrieren. Ein mit "Include" eingeschlossenes Element muss eines der folgenden Stammelemente enthalten:

- **Class**

  Einfache Regelklasse

- **RuleSet**

  Vollständiges Regelwerk, das wiederum selbst eigene Anweisungen `Include` enthalten kann, die rekursiv verarbeitet werden

Es werden verschiedene Typen von Anweisungen `Include` unterstützt:

- **RelativePath**

  Enthält eine XML-Datei mit einem relativen Pfad für die einschließende Datei. Dieses Verfahren kann bei der eigenständigen Entwicklung des Regelwerks durch Entwickler in einer dateibasierten Entwicklungsumgebung sinnvoll sein.

- **Classpath**

  Enthält eine XML-Datei, die an der benannten Position im Laufzeitklassenpfad vorhanden ist. Dieses Verfahren ist geeignet, um auf allgemeine Regelwerke zu verweisen, die sich selten ändern und in die Anwendung integriert sind.

Tipp: Innerhalb eines Regelwerks ist die Reihenfolge, in der Anweisungen `Include` angegeben sind, ohne Bedeutung. Sie können Anweisungen `Include` in einem Regelwerk ganz nach Bedarf umordnen, ohne dass dies das Verhalten des Regelwerks beeinflusst.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Include"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- This rule class is defined directly in this rule set -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Include a rule set defined in another file.

       When assembled into a single rule set, the
       names of all the rule classes must be unique. -->
  <Include>
    <RelativePath value="./HelloWorld.xml"/>
  </Include>

</RuleSet>
```

Der Abschnitt „CER-Konsolidierungskomponente für Regelwerke” auf Seite 30, enthält Informationen dazu, wie ein Regelwerk, das Einschlüsse von `RelativePath` enthält, in einer einzigen Regelwerkdatei komprimiert werden kann.

Regelklasse

Eine Regelklasse definiert das Verhalten ihrer Regelobjektinstanzen.

Eine Regelklasse gibt ihren *Namen* an (der unter allen Regelklassen im Regelwerk eindeutig sein muss), definiert, ob sie *abstrakt* ist, und enthält die folgenden Elemente:

- **Initialization**

  Dieser optionale Block enthält Attribute, deren Werte angegeben werden müssen, wann immer eine Regelobjektinstanz der Klasse erstellt wird.

- **Attribute**

  Dies sind 0 oder mehr berechnete Anweisungen `Attribute`, die jeweils einen Wert beschreiben, der von der Regelklasse berechnet werden kann.

Eine Regelklasse kann in ihrer eigenen XML-Datei (bei der das XML-Stammelement also `Class` lautet) definiert sein und mit einer Anweisung "Include" (siehe „Anweisung `Include`” auf Seite 166) in ein übergeordnetes Regelwerk aufgenommen werden.

Initialisierte Attribute

Der Block `Initialization` enthält eine oder mehrere Anweisungen `Attribute`, die jeweils den Typ (`type`) eines Attributs, jedoch *keine* Ableitung (`derivation`) angeben.

Bei jeder Erstellung einer Regelobjektinstanz für eine Klasse (ob nun innerhalb von Regeln mit dem Ausdruck `create` oder mittels Java-Code unter Verwendung der

generierten Regelklassen oder der dynamischen Regel-API) müssen die Werte aller initialisierten Attribute *in der Reihenfolge angegeben werden, in der sie in der Regelklasse definiert sind*.

Warnung: Infolgedessen sollten Sie die Umordnung der Attribute in einem Block `Initialization` vermeiden, sofern Sie nicht auch alle Stellen (in Regeln oder Java-Code) überarbeiten wollen, an denen Regelobjektinstanzen der Regelklasse erstellt werden.

Berechnete Attribute

Berechnete Attribute werden in der Regelklasse direkt aufgelistet.

Tipp: Innerhalb einer Regelklasse ist die Reihenfolge, in der berechnete Attribute angegeben sind, ohne Bedeutung. Sie können berechnete Attribute in einer Regelklasse ganz nach Bedarf umordnen, ohne dass dies das Verhalten der Regelklasse beeinflusst.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_RuleClass"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Initialization>
      <!-- Initialized attributes each contain a type
           but no derivation.

           You should NOT arbitrarily reorder
           initialized attributes. -->
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="age">
        <type>
          <javaclass name="Number"/>
        </type>
      </Attribute>
    </Initialization>

    <!-- Each calculated attribute specifies both
         a type and a derivation.

         You are free to arbitrarily reorder
         calculated attributes. -->
    <Attribute name="isAdult">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="18"/>
        </compare>
      </derivation>
    </Attribute>

    <Attribute name="isSeniorCitizen">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
```

```
          <reference attribute="age"/>
          <Number value="65"/>
        </compare>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Attribut

Jedes Attribut gibt seinen *Namen* an (der unter den von der Regelklasse definierten oder übernommenen Regelattributen eindeutig sein muss) und enthält die folgenden Elemente.

Jedes Element `Attribute` enthält Folgendes:

- **type**

  Definiert den Typ des Wertes, der von diesem Attribut bereitgestellt wird (siehe Abschnitt „Unterstützte Datentypen" auf Seite 42).
- **derivation** (nur bei berechneten Attributen)

  Definiert, wie das Attribut seinen Wert berechnet. Jedes Element `derivation` enthält einen einzigen CER-Ausdruck (siehe Abschnitt „Vollständige alphabetische Liste der Ausdrücke" auf Seite 171).

  Wichtig: Es gibt besondere Markierungsausdrücke, die die Semantik des Regelattributs ändern können. Weitere Informationen hierzu enthält der Abschnitt „Markierungen" auf Seite 171.

Ausdrücke

CER unterstützt eine Vielzahl von Ausdrücken. Die Ausdrücke sind später in alphabetischer Reihenfolge aufgeführt, zunächst jedoch aus Referenzgründen auch logisch gruppiert (bitte beachten Sie, dass einige Ausdrücke absichtlich in mehreren logischen Gruppen angegeben sind).

Boolesche Logik

- „true" auf Seite 254
- „false" auf Seite 204
- „all" auf Seite 175
- „any" auf Seite 178
- „not" auf Seite 215

Wertevergleich

- „equals" auf Seite 200
- „compare" auf Seite 189
- „sort" auf Seite 239

Konstanten

Die folgenden Ausdrücke stellen Literalkonstantenwerte bereit.

- „true" auf Seite 254
- „false" auf Seite 204
- „null" auf Seite 216
- „String" auf Seite 241
- „Number" auf Seite 217

- „Date" auf Seite 196
- „Code" auf Seite 188
- „FrequencyPattern" auf Seite 209

Bedingungslogik
- „choose" auf Seite 185

Listenkumulierungen
Die folgenden Ausdrücke kumulieren eine Liste von Werten in einem abgeleiteten Wert.
- „all" auf Seite 175
- „any" auf Seite 178
- „sum" auf Seite 243
- „min" auf Seite 214
- „max" auf Seite 212
- „concat" auf Seite 190
- „singleitem" auf Seite 237

Weitere Operationen, die direkt von der Java-Schnittstelle `java.util.List` bereitgestellt werden, sind im Abschnitt „Nützliche Listenoperationen" auf Seite 261, aufgeführt.

Listentransformationen
Die folgenden Ausdrücke wandeln eine Liste um und erstellen auf diese Weise eine neue Liste.
- „dynamiclist" auf Seite 197
- „fixedlist" auf Seite 206
- „filter" auf Seite 204
- „joinlists" auf Seite 211
- „removeduplicates" auf Seite 232
- „sort" auf Seite 239
- „sublists" auf Seite 241

Lokalisierbare Nachrichten
Die folgenden Ausdrücke ermöglichen die Erstellung von Nachrichten, die in der Sprache/Ländereinstellung des Benutzers angezeigt werden können.
- „concat" auf Seite 190
- „ResourceMessage" auf Seite 234
- „XmlMessage" auf Seite 255

Numerische Berechnungen
Die folgenden Ausdrücke unterstützen numerische Berechnungen.
- „Number" auf Seite 217
- „arithmetic" auf Seite 180
- „periodlength" auf Seite 219
- „sum" auf Seite 243
- „max" auf Seite 212
- „min" auf Seite 214

Verweise

Die folgenden Ausdrücke ermöglichen eine Berechnung, um auf ein anderes Element zu verweisen.

- „reference" auf Seite 229
- „current" auf Seite 194
- „this" auf Seite 244

Erstellung

Der folgende Ausdruck ermöglicht die Erstellung eines neuen Regelobjekts.

- „create" auf Seite 191

Abruf

Der folgende Ausdruck ermöglicht dem Abruf von Regelobjekten.

- „readall" auf Seite 224

Java-Aufrufe

Die folgenden Ausdrücke ermöglichen den Aufruf von Java-Code zur Durchführung einer Berechnung.

- „property" auf Seite 221
- „call" auf Seite 183

Markierungen

Die folgenden Ausdrücke sind besondere Markierungen (und keine eigentlichen Berechnungen).

- „abstract" auf Seite 172
- „specified" auf Seite 240

Zeitlinien

Die folgenden Ausdrücke verarbeiten CER-Zeitlinien.

Weitere Details zu CER-Zeitlinien enthält der Abschnitt „Verarbeitung von Daten, die sich mit der Zeit ändern" auf Seite 52 des vorliegenden Handbuchs.

- „Interval" auf Seite 210
- „Timeline" auf Seite 245
- „existencetimeline" auf Seite 202
- „intervalvalue" auf Seite 211
- „timelineoperation" auf Seite 248

Anspruchsberechtigung und Leistungshöhe für Produktbereitstellung

Die folgenden Ausdrücke stellen geschäftsspezifische Berechnungen für die Anspruchsberechtigung und Leistungshöhe bei Produktbereitstellungsfällen bereit.

Beschreibungen dieser Ausdrücke könenn Sie dem Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules entnehmen.

- „combineSuccessionSets" auf Seite 189
- „legislationChange" auf Seite 212
- „rate" auf Seite 224

Vollständige alphabetische Liste der Ausdrücke

Dieser Abschnitt enthält Definitionen für alle Ausdrücke, die in CER und der Anwendung enthalten sind.

Die Ausdrücke sind nachfolgend alphabetisch aufgeführt. Hilfreiche Kategorisierungen dieser Ausdrücke können Sie den vorherigen Abschnitten entnehmen.

Anmerkung: Einige Ausdrücke dienen geschäftsspezifischen Ableitungen in der Anwendung. Solche Ausdrücke sind hier zwar aufgeführt, jedoch unter Verweis auf andere Cúram-Handbücher, in denen diese Ausdrücke in ihrem Geschäftskontext beschrieben sind.

Zur Verkürzung sind die Beispielregelwerke ohne Anmerkungen dargestellt. In der Praxis enthalten Regelwerke, die mit dem CER-Editor gespeichert werden, Anmerkungen für Diagramm- und Beschreibungsinformationen.

abstract:
Dieser Markierungsausdruck gibt an, dass die Ableitung des Attributs für konkrete Unterklassen (oder eine deren Superklassen) angegeben werden muss.

Falls eines oder mehrere Attribute in einer Regelklasse mit `abstract` gekennzeichnet sind, erfordert es der CER-Regelwerkvalidierer, dass auch die Klasse selbst mit *abstract="true"* gekennzeichnet ist, und verhindert die Verwendung der Regelklasse in Ausdrücken "create" (siehe Abschnitt „create" auf Seite 191).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_abstract"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Base class for all types of benefit.
       Every concrete subclass has its own
       calculation of "name" and "isEligible". -->
  <Class name="Benefit" abstract="true">
    <Initialization>
      <!-- The person for which benefit eligibility
        is being determined. -->
      <Attribute name="person">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
    </Initialization>


    <!-- The name of this type of benefit -->
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <abstract/>
      </derivation>
    </Attribute>

    <!-- Whether the person is eligible for this benefit. -->
    <Attribute name="isEligible">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <abstract/>
      </derivation>
    </Attribute>

  </Class>
```

```
<!-- A concrete subclass of Benefit.
     Contains concrete derivations for the inherited
     abstract attributes. -->
<Class name="MedicalBenefit" extends="Benefit">
  <Attribute name="name">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <String value="Medical Benefit"/>
    </derivation>
  </Attribute>
  <Attribute name="isEligible">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <all>
        <fixedlist>
          <listof>
            <javaclass name="Boolean"/>
          </listof>
          <members>
            <!-- NB the person attribute is inherited from Benefit
-->
            <reference attribute="isPoor">
              <reference attribute="person"/>
            </reference>
            <reference attribute="isSick">
              <reference attribute="person"/>
            </reference>
          </members>
        </fixedlist>

      </all>

    </derivation>
  </Attribute>
</Class>


<!-- Another concrete subclass of Benefit,
  with different concrete derivations for the inherited
  abstract attributes. -->
<Class name="NeedyBenefit" extends="Benefit">
  <Attribute name="name">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <String value="Medical Benefit"/>
    </derivation>
  </Attribute>
  <Attribute name="isEligible">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <all>
        <fixedlist>
          <listof>
            <javaclass name="Boolean"/>
          </listof>
          <members>
            <reference attribute="isPoor">
              <reference attribute="person"/>
            </reference>
```

```
            <any>
              <fixedlist>
                <listof>
                  <javaclass name="Boolean"/>
                </listof>
                <members>
                  <reference attribute="isHungry">
                    <reference attribute="person"/>
                  </reference>
                  <reference attribute="isDeprived">
                    <reference attribute="person"/>
                  </reference>
                </members>
              </fixedlist>
            </any>
          </members>
        </fixedlist>

      </all>

    </derivation>
  </Attribute>
</Class>

<Class name="Person">

  <Attribute name="isPoor">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isSick">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isHungry">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isDeprived">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>


  <!-- A list of all the benefits for
       which the person is being assessed.  -->
  <Attribute name="allBenefits">
    <type>
```

```
        <javaclass name="List">
          <ruleclass name="Benefit"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Benefit"/>
          </listof>
          <members>
            <!-- Create instances of the concrete rule classes -->
            <create ruleclass="MedicalBenefit">
              <this/>
            </create>
            <create ruleclass="NeedyBenefit">
              <this/>
            </create>
          </members>
        </fixedlist>

      </derivation>
    </Attribute>

    <!-- The benefits for which this person
         is eligible.

         Note that the list is of the abstract
         rule class "Benefit", but that each
         concrete instance determines its
         eligibility in its own way.    -->
    <Attribute name="eligibleBenefits">
      <type>
        <javaclass name="List">
          <ruleclass name="Benefit"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <reference attribute="allBenefits"/>
          </list>
          <listitemexpression>
            <reference attribute="isEligible">
              <current/>
            </reference>
          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

all:
Dieser Ausdruck ermittelt für eine Liste von booleschen Werten, ob alle Listenwerte *true* lauten.

Die Berechnung wird beim ersten in der Liste gefundenen Wert *false* gestoppt. Falls die Liste leer ist, gibt dieser Ausdruck das Ergebnis *true* zurück.

Die Liste der booleschen Werte wird normalerweise durch einen Ausdruck "fixedlist" (siehe „fixedlist” auf Seite 206) oder "dynamiclist" (siehe „dynamiclist” auf Seite 197) bereitgestellt.

Tipp: Die Reihenfolge der Einträge in dieser Liste hat keinen Einfluss auf den Wert dieses Ausdrucks. Im Hinblick auf das Leistungsverhalten kann es jedoch sinnvoll sein, einen Ausdruck "fixedlist" (siehe „fixedlist" auf Seite 206) so zu strukturieren, dass sich schnell als nicht übereinstimmend herausstellende Werte eher am Anfang der Liste und alle Werte, deren Berechnung möglicherweise aufwendiger ist, eher am Ende der Liste angegeben sind.

Anmerkung: Seit Cúram Version 6 meldet CER Fehler in untergeordneten Ausdrücken nicht mehr, wenn sich ein Fehler nicht auf das Gesamtergebnis auswirkt.

Beispiel: Eine feste Liste von drei booleschen Attributen enthält die folgenden Werte:

- `true`
- <fehler_während_berechnung>
- `false`

Die Berechnung des Wertes von `all` für diese Werte gibt das Ergebnis `false` zurück, weil mindestens einer der Einträge "false" ist (nämlich der dritte Eintrag in der Liste). Dies ist unabhängig davon, dass der zweite Eintrag einen Fehler zurückgibt.

Beispiel: Eine andere feste Liste von drei booleschen Attributen enthält hingegen die folgenden Werte:

- `true`
- <fehler_während_berechnung>
- `true`

Die Berechnung des Wertes von `all` für diese Werte gibt den vom zweiten Eintrag in der Liste gemeldeten Fehler zurück, da dieser Fehler die Feststellung verhindert, ob alle Einträge den Wert `true` aufweisen.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_all"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="isLoneParent">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Example of <all> operating on a <fixedlist> -->
        <!-- To be considered a "lone parent", a person must
             be both unmarried and have at least one child -->
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- We happen to know that most people on our
 database
                   are married, so we test this condition first.

                   If it so happens that the isMarried value is not
                   specified for a Person, then if that Person has
                   no children then the <all> will return false;
                   otherwise it will return an error indicating that
```

```
                the value of isMarried was not specified.
                -->
            <not>
              <reference attribute="isMarried"/>
            </not>
            <not>
              <property name="isEmpty">
                <object>
                  <reference attribute="children"/>
                </object>
              </property>
            </not>
          </members>
        </fixedlist>
      </all>
    </derivation>
  </Attribute>

  <Attribute name="hasNoYoungChildren">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <!-- Example of <all> operating on a <dynamiclist>.

           If it so happens that one child's age cannot be
           calculated, and there is at least one child under 5,
           then the <all> will return false; otherwise, it
           will return the error showing why the child's age could
           not be calculated.
      -->

      <!-- Check whether the children are all over 5 years of age
-->
      <all>
        <dynamiclist>
          <list>
            <reference attribute="children"/>
          </list>
          <listitemexpression>
            <compare comparison="&gt;">
              <reference attribute="age">
                <current/>
              </reference>
              <Number value="5"/>
            </compare>
          </listitemexpression>
        </dynamiclist>
      </all>
    </derivation>
  </Attribute>

  <!-- The children of this person - each child is a person too!
-->
  <Attribute name="children">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isMarried">
    <type>
```

```
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

any:
Dieser Ausdruck ermittelt für eine Liste von booleschen Werten, ob einer der Listenwerte *true* lautet.

Die Berechnung wird beim ersten in der Liste gefundenen Wert *true* gestoppt. Falls die Liste leer ist, gibt dieser Ausdruck das Ergebnis *false* zurück.

Die Liste der booleschen Werte wird normalerweise durch einen Ausdruck "fixedlist" (siehe „fixedlist" auf Seite 206) oder "dynamiclist" (siehe „dynamiclist" auf Seite 197) bereitgestellt.

Tipp: Die Reihenfolge der Einträge in dieser Liste hat keinen Einfluss auf den Wert dieses Ausdrucks. Im Hinblick auf das Leistungsverhalten kann es jedoch sinnvoll sein, einen Ausdruck "fixedlist" (siehe „fixedlist" auf Seite 206) so zu strukturieren, dass sich schnell als zutreffend herausstellende Werte eher am Anfang der Liste und alle Werte, deren Berechnung möglicherweise aufwendiger ist, eher am Ende der Liste angegeben sind.

Anmerkung: Seit Cúram Version 6 meldet CER Fehler in untergeordneten Ausdrücken nicht mehr, wenn sich ein Fehler nicht auf das Gesamtergebnis auswirkt.

Beispiel: Eine feste Liste von drei booleschen Attributen enthält die folgenden Werte:

- `false`
- <fehler_während_berechnung>
- `true`

Die Berechnung des Wertes von `any` für diese Werte gibt das Ergebnis `true` zurück, weil mindestens einer der Einträge "true" lautet (nämlich der dritte Eintrag in der Liste). Dies ist unabhängig davon, dass der zweite Eintrag einen Fehler zurückgibt.

Beispiel: Eine andere feste Liste von drei booleschen Attributen enthält hingegen die folgenden Werte:

- `false`
- <fehler_während_berechnung>
- `false`

Die Berechnung des Wertes von any für diese Werte gibt den vom zweiten Eintrag in der Liste gemeldeten Fehler zurück, da dieser Fehler die Feststellung verhindert, ob einer der Einträge den Wert true aufweist.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_any"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="qualifiesForFreeTravelPass">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Example of <any> operating on a <fixedlist> -->
        <!-- To qualify for a free travel pass, the person
             must be aged, blind or disabled -->
        <any>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- We happen to know that most people on our
                   database are senior citizens, so we test
                   this condition first.

                   If it so happens that the isBlind value is not
                   specified for a Person, then if that Person is
                   disabled then the <any> will return false;
                   otherwise it will return an error indicating that
                   the value of isBlind was not specified.
                   -->

              <compare comparison="&gt;=">
                <reference attribute="age"/>
                <Number value="65"/>
              </compare>
              <reference attribute="isBlind"/>
              <reference attribute="isDisabled"/>
            </members>
          </fixedlist>
        </any>
      </derivation>
    </Attribute>

    <Attribute name="qualifiesForChildBenefit">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Example of <any> operating on a <dynamiclist>.

             If it so happens that one child's age cannot be
             calculated, and there is at least one child under 16,
             then the <any> will return true; otherwise, it
             will return the error showing why the child's age could
             not be calculated.

          -->
        <!-- To qualify for child benefit, this person must
             have one or more children aged under 16. -->
        <any>
          <dynamiclist>
            <list>
```

```
            <reference attribute="children"/>
          </list>
          <listitemexpression>
            <compare comparison="&lt;">
              <reference attribute="age">
                <current/>
              </reference>
              <Number value="16"/>
            </compare>
          </listitemexpression>
        </dynamiclist>
      </any>
    </derivation>
  </Attribute>

  <!-- The children of this person - each child is a person too!
-->
  <Attribute name="children">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isBlind">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isDisabled">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="age">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

 </Class>

</RuleSet>
```

arithmetic:
Dieser Ausdruck führt eine arithmetische Berechnung für zwei Zahlen (eine Zahl auf der linken Seite und eine Zahl auf der rechten Seite) aus. Das Ergebnis wird optional auf die angegebene Anzahl von Dezimalstellen gerundet.

Die folgenden Operationen werden unterstützt:

- **Addition**
  Linke Seite + rechte Seite
- **Subtraktion**
  Linke Seite - rechte Seite
- **Multiplikation**
  Linke Seite * rechte Seite
- **Division**
  Linke Seite / rechte Seite

Falls ein Auf-/Abrunden erforderlich ist, müssen Sie Folgendes angeben:

- Anzahl der Dezimalstellen für die Rundung
- Rundungsmodus, also die Richtung, in der die Rundung ausgeführt wird (eine Liste der unterstützten Rundungsmodi und eine ausführliche Erläuterung ihres jeweiligen Verhaltens enthält das JavaDoc für `RoundingMode`)

Warnung: Bei Divisionsoperation sollten Sie generell einen Rundungsmodus und die Anzahl der Dezimalstellen angeben. Andernfalls kann zur Laufzeit kein exaktes Ergebnis berechnet werden und es tritt ein Laufzeitfehler auf.

Der Regelwerkvalidierer gibt eine Warnung aus, wenn er feststellt, dass für eine Divisionsoperation in einem Regelwerk keine Rundung angegeben ist.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_arithmetic"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="ArithmeticExampleRuleClass">

    <!-- 3 + 2 = 5 -->
    <Attribute name="addANumberToAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <Number value="3"/>
          <Number value="2"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- 3 - 2 = 1 -->
    <Attribute name="subtractANumberFromAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="-">
          <Number value="3"/>
          <Number value="2"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- 3 * 2 = 6 -->
    <Attribute name="multiplyANumberByAnother">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
```

```
    <arithmetic operation="*">
      <Number value="3"/>
      <Number value="2"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- 3 / 2 = 1.5 -->
<!-- Because the division is by 2,
     we can get away without rounding.
     A warning will still be issued by the
     CER rule set validator, though.   -->
<Attribute name="divideANumbersByAnother">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="/">
      <Number value="3"/>
      <Number value="2"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- (3 + 2) * 4 = 20 -->
<Attribute name="chainedArithmetic">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="*">
      <arithmetic operation="+">
        <Number value="3"/>
        <Number value="2"/>
      </arithmetic>
      <Number value="4"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- 1.23 + 3.45 = 4.68,
     = 4.7 when rounded to the nearest 1 decimal place-->
<Attribute name="roundedAddition">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic decimalPlaces="1" operation="+"
      rounding="half_up">
      <Number value="1.23"/>
      <Number value="3.45"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- 2 / 3, = 0.667 to 3 decimal places -->
<!-- If no rounding is specified,
     then a runtime error will occur -->
<Attribute name="roundedDivision">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic decimalPlaces="3" operation="/"
      rounding="half_up">
      <Number value="2"/>
      <Number value="3"/>
```

```
        </arithmetic>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

call:
Dieser Ausdruck ruft eine statische Java-Methode auf, um eine komplexe Berechnung durchzuführen.

Der Ausdruck `call` deklariert Folgendes:

- **type**
  Der Datentyp des zurückgegebenen Wertes (siehe „Unterstützte Datentypen" auf Seite 42).
- **arguments** (optional)
  Eine Liste von Werten, die als Argumente übergeben werden sollen.

Die Java-Methode muss für eine Klasse definiert sein, die sich zum Zeitpunkt der Validierung des Regelwerks im Klassenpfad befindet. Das erste Argument der Methode muss ein Objekt `Session` sein. Die übrigen Argumente müssen mit den Angaben im Regelwerk übereinstimmen.

Warnung: Sie müssen sicherstellen, dass jeder Java-Code, der von einem Ausdruck `call` aufgerufen wird, *nicht* versucht, Werte von Regelobjektattributen zu ändern.

CER-Regelwerke verwenden generell unveränderliche Datentypen. Es besteht jedoch die Möglichkeit, eigene veränderliche Java-Klassen als Datentypen zu verwenden. Wenn Sie dies tun, müssen Sie sicherstellen, dass kein aufgerufener Code die Änderung eines angepassten Java-Datentypes zu ändern, da eine solche Änderung dazu führen könnte, das zuvor ausgeführte Berechnungen nun "falsch" sind.

```
package curam.creole.example;

import curam.creole.execution.RuleObject;
import curam.creole.execution.session.Session;

public class Statics {

  /**
   * Calculates a person's favorite color.
   *
   * This calculation is too complex for rules and so has been
   * coded in java.
   *
   * @param session
   *          The rule session
   * @param person
   *          the person
   * @return the calculated favorite color of the specified person
   */
  public static String calculateFavoriteColor(
      final Session session, final RuleObject person) {

    // Note that the retrieval of the attribute value must be
    // cast to the correct type
    final String name =
        (String) person.getAttributeValue("name").getValue();
    final Number age =
```

```
        (Number) person.getAttributeValue("age").getValue();

    final String ageString = age.toString();
    // Calculate the person's favorite color according
    // to the digits in their age and their name
    if (ageString.contains("5") || ageString.contains("7")) {
      return "Blue";
    } else if (name.contains("z")) {
      return "Purple";
    } else {
      return "Green";
    }

  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_call"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="favoriteColor">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- Call a java static method
             to perform the calculation -->
        <call class="curam.creole.example.Statics"
          method="calculateFavoriteColor">
          <type>
            <javaclass name="String"/>
          </type>
          <arguments>
            <!-- Pass in this person
                 as an argument to the
                 static method -->
            <this/>
          </arguments>
        </call>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Wichtig: Seit Cúram Version 6 unterstützen CER und der Abhängigkeitsmanager die automatische Neuberechnung von durch CER berechneten Werten, falls sich ihre Abhängigkeiten ändern.

Wenn Sie die Implementierung einer statischen Methode ändern, wissen CER und der Abhängigkeitsmanager *nicht* automatisch, dass Attributwerte neu berechnet werden müssen, die unter Verwendung der alten Version der statischen Methode berechnet wurden.

Sobald eine statische Methode in einer Produktionsumgebung für gespeicherte Attributwerte verwendet wurde, sollten Sie anstelle einer Änderung der Implementierung eine neue statische Methode erstellen (mit der erforderlichen neuen Implementierung) und Ihre Regelwerke so ändern, dass die neue statische Methode verwendet wird. Wenn Sie Ihre Regelwerkänderungen, die auf die neue statische Methode verweisen, veröffentlichen, berechnen CER und der Abhängigkeitsmanager automatisch alle Instanzen des betroffenen Attributwerts neu.

choose:
Dieser Ausdruck wählt einen Wert auf der Grundlage einer erfüllten Bedingung aus.

Der Ausdruck `choose` enthält Folgendes:

- **type**

  Eine Datentypkennung (siehe „Unterstützte Datentypen" auf Seite 42), die den Typ des auszuwählenden Werts angibt.
- **test** (optional)

  Ein Ausdruck, der wiederum den für das Element `condition` in jedem Ausdruck `when` zu testenden Wert angibt. Falls kein Ausdruck `test` angegeben ist, wird das Element `condition` in jedem Ausdruck `when` gestestet, um zu überprüfen, ob der Wert *true* zurückgegeben wird.
- **when** (1 oder mehr)

  Jeder Ausdruck enthält eine Bedingung (`condition`) für den Wert und einen Wert (`value`), der beim Erfüllen der Bedingung zurückgegeben wird.
- **otherwise**

  Ein Ausdruck, der einen zurückzugebenden Wert (`value`) enthält (damit auf alle Fälle immer ein Wert ausgewählt wird).

Die Bedingungen werden in der Reihenfolge der Ausdrücke `when` ausgewertet. Die Auswertung stoppt bei der ersten Bedingung, die den Test erfüllt. Nachfolgende Bedingungen werden nicht ausgewertet.

Der Ausdruck `choose` entspricht den Anweisungen `if / else if /.../ else`, die für die meisten Programmiersprachen typisch sind. Mit seiner Hilfe kann effektiv eine Entscheidungstabelle implementiert werden.

Es kann sinnvoll sein, die Bedingungen so anzuordnen, dass die am ehesten erfolgreichen Bedingungen am Beginn der Liste angegeben sind (was unnötige Berechnungen verhindert).

Warnung: Bei einfachen Bedingungen (z. B. zum Testen der Gleichheit mit einem einzelnen Wert) können Sie die Bedingungen normalerweise umordnen, ohne das Verhalten des Regelwerks zu beeinflussen.

Bei komplexeren Berechnungen (und somit im Allgemeinen) müssen Sie jedoch sorgfältig prüfen, ob eine andere Anordnung der Bedingungen zu unerwünschten Verhaltensänderungen führt.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_choose"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="ageCategory">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <choose>
          <!-- There's no explicit <test> clause, so this
               <choose> statement will test each condition to
               see if it is TRUE. -->
          <type>
            <javaclass name="String"/>
          </type>

          <!-- Note that the order of these conditions is
               important; if we were to swap the positions of the
               "Newborn" and "Infant" tests, then all children
               under 5 (including those under 1) would be
               identified as Infants; no children would be
               identified as Newborns.  -->
          <when>
            <condition>
              <compare comparison="&lt;">
                <reference attribute="age"/>
                <Number value="1"/>
              </compare>
            </condition>
            <value>
              <String value="Newborn"/>
            </value>
          </when>
          <when>
            <condition>
              <compare comparison="&lt;">
                <reference attribute="age"/>
                <Number value="5"/>
              </compare>
            </condition>
            <value>
              <String value="Infant"/>
            </value>
          </when>
          <when>
            <condition>
              <compare comparison="&lt;">
                <reference attribute="age"/>
                <Number value="18"/>
              </compare>
```

```
          </condition>
          <value>
            <String value="Child"/>
          </value>
        </when>
        <otherwise>
          <value>
            <String value="Adult"/>
          </value>
        </otherwise>
      </choose>
    </derivation>
  </Attribute>

  <Attribute name="numberOfSpouses">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="maritalStatus">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <choose>
        <type>
          <javaclass name="String"/>
        </type>
        <!-- Test the number of spouses -->
        <test>
          <reference attribute="numberOfSpouses"/>
        </test>
        <!-- Note that the order of the "0" and "1" tests do not
matter -
             so you might want to order these according to whether
most
             Person instances being tested have 0 or 1 spouses.
     -->
        <when>
          <condition>
            <Number value="0"/>
          </condition>
          <value>
            <String value="Unmarried"/>
          </value>
        </when>
        <when>
          <condition>
            <Number value="1"/>
          </condition>
          <value>
            <String value="Married - single spouse"/>
          </value>
        </when>
        <otherwise>
          <value>
            <String value="Married - multiple spouses"/>
          </value>
        </otherwise>
      </choose>

    </derivation>
  </Attribute>
```

```
  </Class>

</RuleSet>
```

Code:
Dies ist ein Literalkonstantenwert, der einen Code aus einer Anwendungscodetabelle darstellt.

Der Ausdruck `Code` gibt einen Codetabellennamen an und verwendet als Eingabe ein einziges Argument, das den Wert des erforderlichen Codes aus der Tabelle angibt.

Anmerkung: Sie müssen den Zeichenfolgewert für den Code angegen. Von Codetabellen generierte Konstanten können nicht verwendet werden, da CER eine vollständig dynamische Sprache ist und nicht von Buildzeitkonstrukten abhängig sein kann.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Code"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Boolean representation of gender -->
    <Attribute name="isMale">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Code representation of gender -->
    <Attribute name="gender">
      <type>
        <codetableentry table="Gender"/>
      </type>
      <derivation>
        <Code table="Gender">
          <choose>
            <type>
              <javaclass name="String"/>
            </type>
            <when>
              <condition>
                <reference attribute="isMale"/>
              </condition>
              <value>
                <!-- use the "MALE" code from the codetable -->
                <String value="MALE"/>
              </value>
            </when>
            <otherwise>
              <value>
                <!-- use the "FEMALE" code from the codetable -->
                <String value="FEMALE"/>
              </value>
            </otherwise>
          </choose>
        </Code>
      </derivation>
    </Attribute>
```

```
  </Class>

</RuleSet>
```

combineSuccessionSets:
Weitere Informationen finden Sie im Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

compare:
Dieser Ausdruck vergleicht einen Wert auf der linken Seite mit einem Wert auf der rechten Seite gemäß dem angegebenen Vergleich.

Die folgenden Vergleiche werden unterstützt:

- <
  Linke Seite "ist kleiner als" rechte Seite.
- <=
  Linke Seite "ist kleiner-gleich" rechte Seite.
- >
  Linke Seite "ist größer als" rechte Seite.
- >=
  Linke Seite "ist größer-gleich" rechte Seite.

Die Werte für die linke Seite und die rechte Seite können ein beliebiger Typ eines vergleichbaren Objekts sein. Hierzu zählen (unter anderem) die folgenden Typen:

- Number
- String
- curam.util.type.Date.

Anmerkung: Alle Instanzen von Number werden vor dem Vergleich in das eigene numerische Format von CER (gestützt auf java.math.BigDecimal) konvertiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_compare"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="CompareExampleRuleClass">

    <!-- 3 >= 2 - TRUE-->
    <Attribute name="compareTwoNumbers">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <Number value="3"/>
          <Number value="2"/>
        </compare>
      </derivation>
    </Attribute>

    <!-- New Year earlier than Christmas - TRUE -->
    <Attribute name="compareTwoDates">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison="&lt;">
```

```
          <Date value="2007-01-01"/>
          <Date value="2007-12-25"/>
        </compare>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

concat:
Dieser Ausdruck erstellt eine lokalisierbare Nachricht (siehe „Lokalisierungsunterstützung” auf Seite 9), indem eine Liste von Werten verkettet wird.

Der Ausdruck concat erstellt eine Zeichenfolge der Werte ohne zusätzliche Leerzeichen oder weiteren Text. Falls Sie eine komplexere Formatierung oder einen lokalisierbaren Text benötigen, verwenden Sie stattdessen den Ausdruck "ResourceMessage" (siehe „ResourceMessage” auf Seite 234).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_concat"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="surname">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- An identifier for a person, including
     first name, surname and date of birth, e.g.
     John Smith (03 Oct 1970).

     First name and surname are plain Strings,
     but date of birth will be localized
     according to the user's locale.
     -->
    <Attribute name="personIdentifier">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
```

```
      <derivation>
        <concat>
          <fixedlist>
            <listof>
              <!-- Note we use Object, as we have a
                   mixture of String and Date items
                   in the list. -->
              <javaclass name="Object"/>
            </listof>
            <members>
              <reference attribute="firstName"/>
              <!-- space separator between names -->
              <String value=" "/>
              <reference attribute="surname"/>
              <String value=" ("/>
              <reference attribute="dateOfBirth"/>
              <String value=")"/>
            </members>
          </fixedlist>
        </concat>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

create:
Dieser Ausdruck ruft eine neue Instanz einer Regelklasse im Hauptspeicher der Sitzung ab. Alle vom Regelobjekt benötigten Initialisierungswerte müssen als untergeordnete Elemente des Ausdrucks `create` angegeben werden.

Seit Cúram Version 6 kann der Ausdruck `create` verwendet werden, um eine neue Instanz einer Regelklasse aus einem *anderen* Regelwerk zu erstellen, indem für das optionale XML-Attribut `ruleset` ein Wert definiert wird.

Anmerkung: Mit dem Ausdruck `create` erstellte Regelobjekte können während der Regelausführung nicht abgerufen werden, da dies das Ordnungsprinzip von CER verletzen würde.

Seit Cúram Version 6 besteht eine Auswahlmöglichkeit für die Syntax, die bei der Übergabe von Werten an ein erstelltes Regelobjekt verwendet wird:

- **Initialisierungsblock**

  CER unterstützt weiterhin einen Block von Attributen, die in einem Element `Initialization` definiert sind. Diese Syntax ist gut für Attribute geeignet, die immer festgelegt sein *müssen* und keine Standardimplementierung besitzen.
- **Angabe von Elementen**

  Seit Cúram Version 6 unterstützt CER auch willkürliche Attribute, deren Wert mit einem Element `specify` überschrieben werden kann, in dem die festzulegenden Attribute angegeben sind und das den zu verwendenden Wert enthält. Diese Syntax kann bei Attributen hilfreich sein, die nur manchmal festgelegt werden und/oder eine Standardimplementierung besitzen.

Seit Cúram Version 6 werden Regelobjekte innerhalb der Sitzung "in einem Pool zusammengefasst". Durch diesen Pool können identische Anforderungen für die Erstellung eines Regelobjekts mit einem einzigen Regelobjekt erfüllt werden, was die Hauptspeicherbelegung und (durch die Verhinderung identischer Berechnungen) auch die CPU-Belastung verringert. Zwei Anforderungen für die Erstellung ei-

nes Regelobjekts sind dann identisch, wenn sie dieselbe Regelklasse anfordern und die Werte aller initialisierten und angegebenen Attribute gleich sind.

Falls im folgenden Beispiel die geschäftliche Telefonnummer einer Person mit der privaten Telefonnummer einer Person identisch ist, wird für beide Nummern ein einziges Regelobjekt verwendet und der abgeleitete Wert für `isOutOfThisArea` folglich nur ein einziges Mal berechnet. Bei unterschiedlichen geschäftlichen und privaten Nummern werden zwei Regelobjekte erstellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_create"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">

    <!-- Phone number details as gathered in evidence -->
    <Attribute name="homePhoneAreaCode">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="homePhoneNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="workPhoneAreaCode">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="workPhoneNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Create PhoneNumber rule objects
         and place them in a list -->
    <Attribute name="phoneNumbers">
      <type>
        <javaclass name="List">
          <ruleclass name="PhoneNumber"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="PhoneNumber"/>
```

```
        </listof>

        <members>

          <!-- Phone Number for the home details. -->
          <create ruleclass="PhoneNumber">
            <!-- The value for PhoneNumber.owner -->
            <this/>
            <!-- The value for PhoneNumber.number -->
            <reference attribute="homePhoneNumber"/>
            <specify attribute="areaCode">
              <!-- The value for PhoneNumber.areaCode -->
              <reference attribute="homePhoneAreaCode"/>
            </specify>
          </create>

          <!-- Phone Number for the work details.

               If a person's work phone number is identical to the
               person's home phone number (i.e. the area code and
               number are the same), then this <create> expression
               will return the same rule object as the rule object
               returned by the <create> expression above.  If the
               phone numbers are not identical, then two different
               rule objects will be returned.-->
          <create ruleclass="PhoneNumber">
            <this/>
            <reference attribute="workPhoneNumber"/>
            <specify attribute="areaCode">
              <reference attribute="workPhoneAreaCode"/>
            </specify>
          </create>

        </members>
      </fixedlist>
    </derivation>
  </Attribute>

</Class>

<Class name="PhoneNumber">

  <Initialization>
    <!-- The values for these attributes must be passed, in order,
         by any <create> expression. -->
    <Attribute name="owner">
      <type>
        <ruleclass name="Person"/>
      </type>
    </Attribute>
    <Attribute name="number">
      <type>
        <javaclass name="Number"/>
      </type>
    </Attribute>
  </Initialization>


  <!-- The value for this attribute may be passed by a <specify>
       element within a <create> expression, which will override
       the default derivation here. -->
  <Attribute name="areaCode">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <!-- Default implementation, used if the <create> expression
```

```
            does not <specify> a value for this attribute. -->
          <Number value="123"/>
        </derivation>
      </Attribute>

      <!-- For a pooled rule object, this derived value will only be
           calculated once.

           For example, if a person's work phone number is identical
           to the person's home phone number, then the same rule
           object will be used for both home and work phone numbers,
           and the "isOutOfThisArea" value for this single rule
           object will be calculated only once.
        -->
      <Attribute name="isOutOfThisArea">
        <type>
          <javaclass name="Boolean"/>
        </type>
        <derivation>
          <not>
            <equals>
              <reference attribute="areaCode"/>
              <!-- The area code for the agency's area -->
              <Number value="123"/>
            </equals>
          </not>
        </derivation>
      </Attribute>
    </Class>
</RuleSet>
```

current:
Dieser Ausdruck verweist auf einen Eintrag in einer Liste, der verarbeitet wird.

Der Ausdruck `current` kann nur innerhalb eines Ausdrucks verwendet werden, der Einträge in einer Liste verarbeitet. Beispiele:

- Ausdruck `listitemexpression` in einem Ausdruck "filter" (siehe „filter" auf Seite 204) oder "dynamiclist" (siehe „dynamiclist" auf Seite 197)
- Ausdruck `sortorder` in einem Ausdruck "sort" (siehe „sort" auf Seite 239)

Zur Verdeutlichung können Sie dem Ausdruck `current` einen Aliasnamen zuweisen, der mit dem Aliasnamen für den Ausdruck `list`, auf den verwiesen wird, übereinstimmen muss. Aliasnamen sind erforderlich, wenn im Gültigkeitsbereich derselben Berechnung mehrere Ausdrücke `current` verwendet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_listitem"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="adults">
```

```
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <filter>
        <list>
          <reference attribute="members"/>
        </list>
        <listitemexpression>
          <!-- The reference uses current to refer
                 to an item in the list of Person
                 rule objects. -->
          <reference attribute="isAdult">
            <current/>
          </reference>
        </listitemexpression>
      </filter>
    </derivation>
  </Attribute>

</Class>

<Class name="Person">

  <Attribute name="children">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="age">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isAdult">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <compare comparison=">=">
        <reference attribute="age"/>
        <Number value="18"/>
      </compare>

    </derivation>
  </Attribute>

  <!-- The children of this person who
       are not yet adults. -->
  <Attribute name="dependentChildren">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
```

```
      <derivation>
        <filter>
          <!-- Use an alias to avoid confusion (for human
               readers of the rule set!) between the parent
               Person and the child Person.  -->
          <list alias="child">
            <reference attribute="children"/>
          </list>
          <listitemexpression>
            <not>
              <reference attribute="isAdult">
                <!-- The alias on the current must match
                   that on the list. -->
                <current alias="child"/>
              </reference>
            </not>
          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Date:
Dieser Ausdruck ist ein literaler Datumskonstantenwert des Typs `curam.util.type.Date`.

Der Wert des Ausdrucks `Date` wird im Format *jjjj-mm-tt* angegeben.

Anmerkung: Es gibt absichtlicherweise in CER keine Funktion, mit der das aktuelle Datum abgerufen werden kann, da eine solche Funktion insofern flüchtig ist, als sie heute einen anderen Wert als morgen zurückgeben würde.

Flüchtige Funktionen sind in CER nicht zulässig. Falls sich das Ergebnis einer Funktion ändern könnte, würde dies bedeuten, dass zuvor ausgeführte Berechnungen nun möglicherweise "falsch" sind.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Date"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="DateExampleRuleClass">

    <Attribute name="nullDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <!-- A null Date -->
        <null/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <!-- The Date 3rd October, 1970 -->
        <Date value="1970-10-03"/>
      </derivation>
    </Attribute>
```

```
  </Class>

</RuleSet>
```

dynamiclist:
Dieser Ausdruck erstellt eine neue Liste, indem ein Ausdruck für jeden Eintrag einer vorhandenen Liste ausgewertet wird.

Die neue Liste enthält pro Eintrag in der vorhandenen Liste einen korrespondierenden Eintrag, wobei die Reihenfolge erhalten bleibt.

Ein Ausdruck `dynamiclist` gibt Folgendes an:

- **list**

  Die vorhandene Liste.
- **listitemexpression**

  Der Ausdruck, der für jeden Eintrag der vorhandenen Liste auszuwerten ist.

Ein Ausdruck `dynamiclist` kann verwendet werden, wenn die Anzahl der Einträge in der gewünschten Liste zur Entwurfszeit nicht bekannt ist (also je nach dem Wert anderer Attribute von Ausführung zu Ausführung variieren kann). Falls die Anzahl der Einträge festgelegt ist (also zur Entwurfszeit bekannt ist), sollte stattdessen der Ausdruck "fixedlist" (siehe „fixedlist" auf Seite 206) verwendet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_dynamiclist"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isDisabled">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="totalIncome">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
```

```
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Pet">
  <Initialization>
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
  </Initialization>

</Class>

<Class name="Household">

  <Attribute name="members">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="containsDisabledPerson">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <any>
        <!-- gets a list of Booleans, corresponding
             to the isDisabled attribute on each
             Person members of this Household   -->
        <dynamiclist>
          <list>
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <reference attribute="isDisabled">
              <current/>
            </reference>
          </listitemexpression>
        </dynamiclist>
      </any>
    </derivation>
  </Attribute>

  <Attribute name="totalIncomeOfAdultMembers">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <sum>
        <dynamiclist>
          <list>
            <!-- filter the members down to
                 just the adults -->
            <filter>
              <list>
```

```
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <compare comparison=">=">
              <reference attribute="age">
                <current/>
              </reference>
              <Number value="18"/>
            </compare>
          </listitemexpression>
        </filter>
      </list>
      <listitemexpression>
        <reference attribute="totalIncome">
          <current/>
        </reference>
      </listitemexpression>
    </dynamiclist>
  </sum>
 </derivation>
</Attribute>

<Attribute name="memberAges">
 <type>
  <javaclass name="List">
   <javaclass name="Number"/>
  </javaclass>
 </type>
 <derivation>
  <dynamiclist>
   <list>
    <reference attribute="members"/>
   </list>
   <listitemexpression>
    <reference attribute="age">
     <current/>
    </reference>
   </listitemexpression>
  </dynamiclist>
 </derivation>
</Attribute>

<Attribute name="youngestAge">
 <type>
  <javaclass name="Number"/>
 </type>
 <derivation>
  <min>
   <reference attribute="memberAges"/>
  </min>
 </derivation>
</Attribute>

<!-- get all the pets in the household,
     by joining together each person's
     list of pets -->
<Attribute name="allPets">
 <type>
  <javaclass name="List">
   <ruleclass name="Pet"/>
  </javaclass>
 </type>
 <derivation>
  <joinlists>
   <!-- a list of list of pets, one
        list for each household
        member -->
```

```
          <dynamiclist>
            <list>
              <reference attribute="members"/>
            </list>
            <listitemexpression>
              <reference attribute="pets">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>

        </joinlists>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

defaultDescription:
Dieser Ausdruck stellte eine Standardimplementierung des Attributs `description` bereit, die alle Regelklassen aus der Stammregelklasse übernehmen. Weitere Informationen finden Sie im Abschnitt „Unterstützte Datentypen" auf Seite 42.

Jede Regelklasse sollte das Attribut `description` aus der Stammregelklasse überschreiben, um eine aussagekräftigere Beschreibung bereitzustellen. Falls keine Überschreibung angegeben ist (oder übernommen wird), wird bei der Validierung des Regelwerks eine Warnung ausgegeben.

Wichtig: Der Ausdruck `defaultDescription` darf *nur* durch die Stammregelklasse verwendet werden. Sie dürfen ihn nicht in Ihren eigenen Regelklassen einsetzen.

```
<Class name="RootRuleClass" abstract="true">
    <Attribute name="description">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- For use ONLY in the RootRuleClass -->
        <defaultDescription/>
      </derivation>
    </Attribute>
  </Class>
```

equals:
Dieser Ausdruck ermittelt, ob zwei Objekte (ein Objekt auf der linken Seite und ein Objekt auf der rechten Seite) identisch sind.

Werte des Typs `Number` werden vor dem Vergleich von CER in das eigene numerische Format (gestützt auf `java.math.BigDecimal`) konvertiert. Unterschiede bei führenden oder abschließenden Nullen werden ignoriert.

Werte `null` werden problemlos verglichen. Falls der Wert auf der linken Seite und der Wert auf der rechten Seite `null` ist, gibt der Ausdruck `equals` das Ergebnis `true` zurück. Ist nur einer der Werte auf der linken Seite und der rechten Seite `null`, gibt der Ausdruck `equals` das Ergebnis `false` zurück.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_equals"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="EqualsExampleRuleClass">
```

```
<!-- TRUE -->
<Attribute name="identicalStrings">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <equals>
      <String value="A String"/>
      <String value="A String"/>
    </equals>
  </derivation>
</Attribute>

<!-- FALSE -->
<Attribute name="differentStrings">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <equals>
      <String value="A String"/>
      <String value="A different String"/>
    </equals>
  </derivation>
</Attribute>

<!-- TRUE -->
<Attribute name="identicalNumbers">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <equals>
      <!-- These numbers are the same,
        disregarding trivial
        differences in leading/trailing
        zeroes -->
      <Number value="123"/>
      <Number value="000123.000"/>
    </equals>
  </derivation>
</Attribute>

<!-- FALSE -->
<Attribute name="differentTypes">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <equals>
      <!-- These objects are of
        different types, so are
        not equal even if they
        "look" the same.-->
      <String value="123"/>
      <Number value="123"/>
    </equals>
  </derivation>
</Attribute>

<!-- FALSE -->
<Attribute name="oneNull">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
```

```
        <equals>
          <null/>
          <Number value="456"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- TRUE -->
    <Attribute name="twoNulls">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <null/>
          <null/>
        </equals>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

existencetimeline:
Dieser Ausdruck erstellt aus einem Paar von inklusivem Startdatum und inklusivem Endatum (beide optional) eine Zeitlinie eines angegebenen Typs.

Weitere Informationen finden Sie im Abschnitt „Zeitlinien erstellen" auf Seite 61.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_existencetimeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>


    <!-- will be null if the person is still alive -->
    <Attribute name="dateOfDeath">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Creates a timeline which is false before the Person is
         born, true while the Person is alive, and false after the
         Person dies.   If the Person has no date-of-death recorded,
         there will be no trailing "false" interval. -->
    <Attribute name="isAliveTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Boolean"/>
        </javaclass>
```

```
      </type>
      <derivation>
        <existencetimeline>
          <intervaltype>
            <javaclass name="Boolean"/>
          </intervaltype>
          <intervalfromdate>
            <reference attribute="dateOfBirth"/>
          </intervalfromdate>
          <intervaltodate>
            <reference attribute="dateOfDeath"/>
          </intervaltodate>
          <preExistenceValue>
            <false/>
          </preExistenceValue>
          <existenceValue>
            <true/>
          </existenceValue>
          <postExistenceValue>
            <false/>
          </postExistenceValue>
        </existencetimeline>

      </derivation>
    </Attribute>

   <!-- Creates a timeline which is "Before Birth" before the Person
        is born, "During Lifetime" while the Person is alive, and
        "After Death" after the Person dies.   If the Person has no
        date-of-death recorded, there will be no trailing "After
        Death" interval. -->
    <Attribute name="lifeStatus">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
        <existencetimeline>
          <intervaltype>
            <javaclass name="String"/>
          </intervaltype>
          <intervalfromdate>
            <reference attribute="dateOfBirth"/>
          </intervalfromdate>
          <intervaltodate>
            <reference attribute="dateOfDeath"/>
          </intervaltodate>
          <preExistenceValue>
            <String value="Before Birth"/>
          </preExistenceValue>
          <existenceValue>
            <String value="During Lifetime"/>
          </existenceValue>
          <postExistenceValue>
            <String value="After Death"/>
          </postExistenceValue>
        </existencetimeline>

      </derivation>
    </Attribute>



  </Class>
</RuleSet>
```

false:
Dies ist der boolesche konstante Wert "false".

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_false"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="FalseExampleRuleClass">

    <Attribute name="isCuramExpertRulesFantastic">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <false/>
        </not>
      </derivation>
    </Attribute>

    <Attribute name="didCookbookWinPulitzerPrize">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <false/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

filter:
Dieser Ausdruck erstellt eine neue Liste mit allen Einträgen aus einer vorhandenen Liste, die die Filterbedingung erfüllen.

Der Ausdruck `filter` enthält Folgendes:

- **list**

  Die vorhandene zu filternde Liste.
- **listitemexpression**

  Der Test, der auf jeden Listeneintrag anzuwenden ist.

Der Ausdruck `listitemexpression` enthält normalerweise eine oder mehrere Berechnungen, die auf das Element "current" (siehe „`current`" auf Seite 194) in der Liste angewendet werden.

Die relative Reihenfolge der Listeneinträge in der gefilterten Liste behält die relative Reihenfolge der Listeneinträge in der ursprünglichen Liste bei. Falls keiner der Einträge in der Liste die Filterbedingung erfüllt, wird eine leere Liste zurückgegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_filter"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
```

```
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- The spouse of this person, or
    null if unmarried -->
  <Attribute name="spouse">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- The children of this person -->
  <Attribute name="children">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>



</Class>



<Class name="Household">

  <!-- All the people in the household -->
  <Attribute name="members">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- All the adults in the household -->
  <Attribute name="adultMembers">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <filter>
        <list>
          <reference attribute="members"/>
        </list>
        <listitemexpression>
          <compare comparison=">=">
            <reference attribute="age">
              <current/>
            </reference>
            <Number value="18"/>
```

```
          </compare>
        </listitemexpression>
      </filter>
    </derivation>
  </Attribute>

  <!-- All the lone parents in the household -->
  <Attribute name="loneParents">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <filter>
        <list>
          <reference attribute="members"/>
        </list>
        <listitemexpression>
          <all>
            <fixedlist>
              <listof>
                <javaclass name="Boolean"/>
              </listof>
              <members>

                <!-- No spouse -->
                <equals>
                  <reference attribute="spouse">
                    <current/>
                  </reference>
                  <null/>
                </equals>
                <!-- At least one child -->
                <not>
                  <property name="isEmpty">
                    <object>
                      <reference attribute="children">
                        <current/>
                      </reference>
                    </object>
                  </property>
                </not>
              </members>
            </fixedlist>
          </all>
        </listitemexpression>
      </filter>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

fixedlist:
Dieser Ausdruck erstellt eine neue Liste aus Einträgen, die zum Zeitpunkt des Regelwerkentwurfs bekannt sind.

Der Ausdruck `fixedlist` gibt Folgendes an:

- **listof**

  Der Typ des Eintrags in der zurückgegebenen Liste (siehe „Unterstützte Datentypen" auf Seite 42).
- **members**

Die Einträge in der Liste.

Die erstellte Liste enthält die Einträge in der im Regelwerk angegebenen Reihenfolge.

Tipp: Das Element `members` kann 0, 1 oder viele untergeordnete Elemente enthalten.

Falls der Ausdruck `fixedlist` in einer Listenverarbeitungsoperation enthalten ist, aber nur 0 oder 1 Listeneintrag enthält, gibt der CER-Regelwerkvalidierer jedoch die Warnung aus, dass die Liste möglicherweise unnötig ist.

Wenn Sie eine Liste erstellen müssen, bei der die Anzahl der Einträge während der Entwurfszeit nicht bekannt ist, sollten Sie stattdessen den Ausdruck "dynamiclist" (siehe „dynamiclist" auf Seite 197) verwenden.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_fixedlist"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- The pets owned by this Person -->
    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>

        <!-- A fixed list of Pets -->
        <fixedlist>
          <listof>
            <ruleclass name="Pet"/>
          </listof>
          <members>
            <!-- Every Person has exactly two pets,
                 Skippy and Lassie -->
            <create ruleclass="Pet">
              <String value="Skippy"/>
              <String value="Kangaroo"/>
            </create>
            <create ruleclass="Pet">
              <String value="Lassie"/>
              <String value="Dog"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>


    <Attribute name="isEntitledToBenefits">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <all>
          <!-- A fixed list of Boolean conditions -->
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
```

```
            <members>
              <!-- Must be an adult -->
              <compare comparison=">=">
                <reference attribute="age"/>
                <Number value="18"/>
              </compare>
              <!-- Must be resident in the state -->
              <reference attribute="isResidentInTheState"/>
              <!-- Must have income under the threshold for benefits
-->
              <compare comparison="&lt;">
                <reference attribute="totalIncome"/>
                <Number value="100"/>
              </compare>
            </members>
          </fixedlist>

        </all>

      </derivation>
    </Attribute>

    <Attribute name="totalIncome">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- A pointless sum of one item -
             the CER rule set validator will warn that this
             fixedlist may be unnecessary. -->
        <sum>
          <fixedlist>
            <listof>
              <javaclass name="Number"/>
            </listof>
            <members>
              <!-- Sum up only the earned income -->
              <reference attribute="earnedIncome"/>
            </members>
          </fixedlist>

        </sum>

      </derivation>
    </Attribute>

    <Attribute name="earnedIncome">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>


    <Attribute name="isResidentInTheState">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
```

```
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Initialization>

      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

      <Attribute name="species">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

    </Initialization>

  </Class>

</RuleSet>
```

FrequencyPattern:
Ein literaler konstanter Wert für das Intervallmuster mit dem Typ `curam.util-.type.FrequencyPattern`.

Der Wert des Ausdrucks `FrequencyPattern` wird als neunstellige Zahl angegeben. Die Bedeutung der Ziffernfolge ist im JavaDoc für `curam.util.type.FrequencyPattern` erläutert.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_FrequencyPattern"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="FrequencyPatternExampleRuleClass">

    <Attribute name="nullFrequencyPattern">
      <type>
        <javaclass name="curam.util.type.FrequencyPattern"/>
      </type>
      <derivation>
        <!-- A null FrequencyPattern -->
        <null/>
      </derivation>
    </Attribute>

    <Attribute name="weeklyOnMondays">
      <type>
        <javaclass name="curam.util.type.FrequencyPattern"/>
      </type>
      <derivation>
        <!-- The Frequency Pattern string for
             "Weekly on Mondays" -->
        <FrequencyPattern value="100100100"/>
      </derivation>
    </Attribute>
```

```
  </Class>

</RuleSet>
```

Interval:
Dieser Ausdruck erstellt ein Intervall (siehe Abschnitt „Verarbeitung von Daten, die sich mit der Zeit ändern" auf Seite 52) eines bestimmten Typs mit einem Wert, der ab einem angegebenen Datum gültig ist.

Dieser Ausdruck wird normalerweise bei der Erstellung einer Zeitlinie (siehe „Timeline" auf Seite 245) verwendet.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Interval"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateInterval">

    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- Another interval-->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <!-- Creates an interval of the specified type.
                     Typically used as input into a <Timeline>. -->
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <Number value="10000"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>

          </intervals>
        </Timeline>

      </derivation>
    </Attribute>
```

```
  </Class>
</RuleSet>
```

intervalvalue:
Dieser Ausdruck schließt einen Ausdruck ein, der eine Zeitlinie zurückgibt (siehe Abschnitt „Verarbeitung von Daten, die sich mit der Zeit ändern" auf Seite 52), und ermöglicht die Ausführung eines eingeschlossenen Ausdrucks für die einzelnen Werte innerhalb der Zeitlinie. Im Grunde genommen dient dieser Ausdruck dazu, für einen äußeren Ausdruck zu "verschleiern", dass er für eine Zeitlinie ausgeführt wird.

Dieser Ausdruck kann nur verschachtelt innerhalb eines Ausdrucks "timelineoperation" (siehe „timelineoperation" auf Seite 248) eingesetzt werden. Eine ausführlichere Beschreibung sowie Verwendungsbeispiele für den Ausdruck `intervalvalue` enthält der Abschnitt „timelineoperation" auf Seite 248.

joinlists:
Dieser Ausdruck erstellt eine neue Liste, indem einige vorhandene Listen zusammengeführt werden.

Der Ausdruck `joinlists` verwendet als Eingabe ein einziges Argument, bei dem es sich um eine Liste von Listen handeln muss.

Die Reihenfolge der Einträge in der neuen Liste ist mit ihrer Reihenfolge in der Quellenliste identisch. Die Listen werden in der Reihenfolge zusammengeführt, in der sie bereitgestellt werden.

Falls die zusammenzuführenden Listen doppelte Einträge enthalten können, kann es sinnvoll sein, den Ausdruck `joinlists` in einen Ausdruck "removeduplicates" (siehe „removeduplicates" auf Seite 232) einzuschließen.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_joinlists"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>
```

```
  </Class>

  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- get all the pets in the household,
         by joining together each person's
         list of pets -->
    <Attribute name="allPets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <joinlists>
          <!-- a list of list of pets, one
               list for each household
               member -->
          <dynamiclist>
            <list>
              <reference attribute="members"/>
            </list>
            <listitemexpression>
              <reference attribute="pets">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>

        </joinlists>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

legislationChange:
Weitere Informationen finden Sie im Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

max:
Dieser Ausdruck ermittelt den größten Wert in einer Liste (bzw. null, falls die Liste leer ist).

Die Liste kann jeden beliebigen Typ eines vergleichbaren Objekts enthalten. Hierzu zählen (unter anderem) die folgenden Typen:

- Number
- String
- curam.util.type.Date.

Anmerkung: Alle Instanzen von `Number` werden vor dem Vergleich in das eigene numerische Format von CER (gestützt auf `java.math.BigDecimal`) konvertiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_max"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="MaxExampleRuleClass">

    <!-- Will pick out "Cherry" as the "largest" String value -->
    <Attribute name="alphabeticallyLastFruit">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <max>
          <reference attribute="fruits"/>
        </max>
      </derivation>
    </Attribute>

    <Attribute name="fruits">
      <type>
        <javaclass name="List">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <javaclass name="String"/>
          </listof>
          <members>
            <String value="Apple"/>
            <String value="Banana"/>
            <String value="Cherry"/>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Determines the number of spots on the spottiest dog -->
    <Attribute name="largestNumberOfSpots">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <max>
          <dynamiclist>
            <list>
              <reference attribute="dalmatians"/>
            </list>
            <listitemexpression>
              <reference attribute="numberOfSpots">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </max>
      </derivation>
    </Attribute>

    <Attribute name="dalmatians">
      <type>
        <javaclass name="List">
          <ruleclass name="Dalmation"/>
```

```
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Dalmation">

    <Attribute name="numberOfSpots">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

min:
Dieser Ausdruck ermittelt den kleinsten Wert in einer Liste (bzw. `null`, falls die Liste leer ist).

Die Liste kann jeden beliebigen Typ eines vergleichbaren Objekts enthalten. Hierzu zählen (unter anderem) die folgenden Typen:

- `Number`
- `String`
- `curam.util.type.Date`.

Anmerkung: Alle Instanzen von `Number` werden vor dem Vergleich in das eigene numerische Format von CER (gestützt auf `java.math.BigDecimal`) konvertiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_min"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="MinExampleRuleClass">

    <!-- Will pick out New Year as the "earliest" Date value -->
    <Attribute name="eariestDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <min>
          <reference attribute="publicHolidays"/>
        </min>
      </derivation>
    </Attribute>

    <Attribute name="publicHolidays">
      <type>
        <javaclass name="List">
          <javaclass name="curam.util.type.Date"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
```

```
          <listof>
            <javaclass name="curam.util.type.Date"/>
          </listof>
          <members>
            <Date value="2007-01-01"/>
            <Date value="2007-12-25"/>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Determines the number of strips on the least-stripey
  zebra-->
    <Attribute name="smallestNumberOfStripes">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <min>
          <dynamiclist>
            <list>
              <reference attribute="zebras"/>
            </list>
            <listitemexpression>
              <reference attribute="numberOfStripes">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </min>
      </derivation>
    </Attribute>

    <Attribute name="zebras">
      <type>
        <javaclass name="List">
          <ruleclass name="Zebra"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Zebra">

    <Attribute name="numberOfStripes">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

not:
Dieser Ausdruck verneint einen booleschen Wert.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_not"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="isLivingInUSA">

      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Note that this not-within-not is somewhat contrived.
 -->
        <not>
          <reference attribute="isLivingOutsideUSA"/>
        </not>
      </derivation>

    </Attribute>

    <Attribute name="isLivingOutsideUSA">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <equals>
            <reference attribute="country"/>
            <String value="USA"/>
          </equals>
        </not>
      </derivation>
    </Attribute>

    <!-- The country in which this person resides. -->
    <Attribute name="country">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

null:
Dieser Ausdruck gibt einen konstanten Wert `null` an.

Das Festlegen eines Wertes mit `null` kann hilfreich sein, um anzugeben, dass kein Wert gilt.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_null"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>
```

```
  </Class>

  <Class name="Person">

    <!-- This Person's favorite Pet, or
         null if the Person owns no pet. -->
    <Attribute name="favoritePet">
      <type>
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- The name of this Person's
         favorite Pet, or null if
         the Person owns no pet.

         We have to test for the favoritePet
         being null before performing the
         (simple) calculation.-->
    <Attribute name="favoritePetsName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <choose>
          <type>
            <javaclass name="String"/>
          </type>
          <when>
            <!-- if this Person has no
                 favorite pet, then calculate the
                 name of the favorite pet as null. -->
            <condition>
              <equals>
                <reference attribute="favoritePet"/>
                <null/>
              </equals>
            </condition>
            <value>
              <null/>
            </value>
          </when>
          <otherwise>
            <value>
              <!-- get the name of the favorite pet -->
              <reference attribute="name">
                <reference attribute="favoritePet"/>
              </reference>
            </value>
          </otherwise>
        </choose>

      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Number:
Dieser Ausdruck gibt einen literalen konstanten Zahlenwert (Typ "Number") an.

Eine Zahl ist in CER ein beliebig langer Dezimalwert, der unter Verwendung eines Punkts (".") als Dezimalzeichen und ohne Tausendertrennzeichen angegeben wird.

CER-Geschäftsberechnungen beziehen häufig Prozentsätze ein (z. B. "10 % des Einkommens der Person abziehen"). Um die Codierung solcher Regeln zu unterstützen lässt CER die Angabe eines Wertes des Typs "Number" als Prozentsatz zu. Hierzu wird der Zahl einfach das Suffix % nachgestellt. Beispielsweise verhalten sich die Zahlen *12.345%* und *0.12345* in Berechnungen identisch (die Variante mit dem Prozentzeichen wird jedoch als Prozentsatz angezeigt).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Number"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="NumberExampleRuleClass">

    <Attribute name="aPositiveInteger">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- A positive integer -->
        <Number value="1"/>
      </derivation>
    </Attribute>

    <Attribute name="aNegativeInteger">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- A negative integer -->
        <Number value="-2"/>
      </derivation>
    </Attribute>

    <Attribute name="aDecimalNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- A decimal number.

             Numbers are arbitrarily long/precise, use "." for
             the decimal separator and have no thousands
             separator.
        -->
        <Number value="-12345.6789"/>
      </derivation>
    </Attribute>

    <Attribute name="aPercentage">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- A percentage
            (12.345% is equivalent to the number 0.12345) -->
        <Number value="12.345%"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

periodlength:
Dieser Ausdruck berechnet die Menge der Zeiteinheiten zwischen zwei Datumsangaben.

Es muss eine der folgenden Zeiteinheiten angegeben werden:

- *days* (Tage)
- *weeks* (Wochen)
- *months* (Monate)
- *years* (Jahre)

Der Ausdruck `periodlength` muss außerdem angeben, ob der Zeitraum das Enddatum einschließen soll (*inclusive*) oder ausschließen soll (*exclusive*). Das Startdatum ist im Zeitraum immer eingeschlossen.

Die Berechnung für die Länge des Zeitraums wird stets auf die nächste ganze Zahl abgerundet.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_periodlength"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="PeriodLengthExampleClass">

    <!-- NB 1970 was not a leap year -->
    <Attribute name="firstDayOfJanuary1970">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1970-01-01"/>
      </derivation>
    </Attribute>

    <Attribute name="lastDayOfDecember1970">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1970-12-31"/>
      </derivation>
    </Attribute>

    <Attribute name="firstDayOfJanuary1971">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1971-01-01"/>
      </derivation>
    </Attribute>

    <Attribute name="sameDay_LengthInDays">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- starts and ends on the same day = 1 day -->
        <periodlength endDateInclusion="inclusive" unit="days">
          <reference attribute="firstDayOfJanuary1970"/>
          <reference attribute="firstDayOfJanuary1970"/>
        </periodlength>
      </derivation>
```

```
</Attribute>

<Attribute name="sameDay_LengthInWeeks">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- starts and ends on the same day = 0 weeks-->
    <periodlength endDateInclusion="exclusive" unit="weeks">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="firstDayOfJanuary1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="januaryToDecember_LengthInDays">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 365 days -->
    <periodlength endDateInclusion="inclusive" unit="days">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="lastDayOfDecember1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="januaryToDecember_LengthInYearsExclusive">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 0 years (nearly 1 year, but just 1 day short) -->
    <periodlength endDateInclusion="exclusive" unit="years">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="lastDayOfDecember1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="januaryToDecember_LengthInYearsInclusive">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 1 year (exactly) -->
    <periodlength endDateInclusion="inclusive" unit="years">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="lastDayOfDecember1970"/>
    </periodlength>
  </derivation>
</Attribute>


<Attribute name="januaryToJanuary_LengthInYearsExclusive">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 1 year (exactly) -->
    <periodlength endDateInclusion="exclusive" unit="years">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="firstDayOfJanuary1971"/>
    </periodlength>
  </derivation>
</Attribute>
```

```
  </Class>

</RuleSet>
```

property:
Dieser Ausdruck ruft die Eigenschaft eines Java-Objekts ab.

Der Ausdruck `property` gibt den Namen der aufzurufenden Java-Methode sowie Folgendes an:

- **object**
  Das Java-Objekt, für das die Operation ausgeführt werden soll.
- **arguments**
  Optional eine Liste von Argumenten, die an die Java-Methode übergeben werden sollen.

Der Ausdruck `property` ermöglicht CER die Nutzung von leistungsstarken Java-Klassen, ohne ein beliebiges Subset von Methoden als CER-Ausdrücke replizieren zu müssen. Beispielsweise enthält `java.util.List` eine Methode `size`, weshalb CER keinen expliziten Ausdruck besitzt, um die Anzahl von Einträgen in einer Liste zu berechnen.

Um das CER-Prinzip der Unveränderlichkeit einzuhalten, können jedoch nur Java-Methoden aufgerufen werden, die nicht den Wert eines Objekts ändern. CER lässt den Aufruf einer Methode für Eigenschaften nur dann zu, wenn die Methode in der "Sicherheitsliste" der Methoden für die Klasse des Objekts (bzw. für eine ihrer übergeordneten Klassen oder Schnittstellen) enthalten ist.

Eine Methode gilt als sicher, wenn sie in der Sicherheitsliste explizit als solche gekennzeichnet ist. Ist sie in der Sicherheitsliste nicht vorhanden, gibt der CER-Regelwerkvalidierer einen Fehler aus.

Tipp: Das explizite Festlegen der Sicherheit mit *false* ist unnötig, kann jedoch zur Vollständigkeit der Dokumentation (wie beispielsweise bei den in CER enthaltenen Sicherheitslisten) vorgenommen werden.

Die Sicherheitsliste für eine Klasse ist eine Eigenschaftendatei, die sich in demselben Paket wie die Klasse befindet und mit *<klassenname>_CREOLE.properties* benannt ist.

CER enthält Sicherheitslisten für die folgenden Java-Klassen und- Schnittstellen:

- `curam.creole.value.Timeline`
- `java.lang.Object`
- `java.lang.Number`
- `java.util.List`

Sicherheitsliste für Methoden `curam.creole.value.Timeline`.

```
# Safe list for curam.creole.value.Timeline

# safe
valueOn.safe=true
```

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_property"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
```

```
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <Attribute name="isMinor">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <ruleclass name="Boolean"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Whether this person is a child. -->
    <Attribute name="isAChild">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <property name="valueOn">
          <object>
            <reference attribute="isMinor"/>
          </object>
          <arguments>
            <Date value="2000-01-01"/>
          </arguments>
        </property>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Sicherheitsliste für Methoden `java.lang.Object`.

```
# Safe list for java.lang.Object

# safe
toString.safe=true

# force equality to be evaluated using <equals>
equals.safe=false

# not exposed, even though they're "safe"
hashCode.safe=false
getClass.safe=false
```

Sicherheitsliste für Methoden `java.lang.Number`.

```
# Safe list for java.lang.Number

byteValue.safe=true
doubleValue.safe=true
floatValue.safe=true
intValue.safe=true
longValue.safe=true
shortValue.safe=true
```

Sicherheitsliste für Methoden `java.util.List`.

```
# Safe list for java.util.List

contains.safe=true
containsAll.safe=true
```

```
get.safe=true

indexOf.safe=true
isEmpty.safe=true
lastIndexOf.safe=true
size.safe=true
subList.safe=true


# not exposed
hashCode.safe=false
listIterator.safe=false
iterator.safe=false
toArray.safe=false

# mutators - unsafe
add.safe=false
addAll.safe=false
clear.safe=false
remove.safe=false
removeAll.safe=false
retainAll.safe=false
```

Beschreibungen für einige nützliche Eigenschaften der Java-Schnittstelle `List` finden Sie in „Nützliche Listenoperationen" auf Seite 261.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_property"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Whether this person has any children.

         Tests the isEmpty property of List. -->
    <Attribute name="hasChildren">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <property name="isEmpty">
            <object>
              <reference attribute="children"/>
            </object>
          </property>
        </not>

      </derivation>
    </Attribute>

    <!-- All this person's children, excluding the first child.

         Uses the subList property of List, passing in:
         - (inclusive) from item at position "1" (denoting the
```

```
 second
           member in the list; lists in Java are zero-based)
         - (exclusive) to item at position "size of list" (denoting
           the position after the last item in the list)
    -->
    <Attribute name="secondAndSubsequentChildren">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <property name="subList">
          <object>
            <reference attribute="children"/>
          </object>
          <arguments>
            <!-- The number must be converted to an integer
                 (as required by List.subList). -->
            <property name="intValue">
              <object>
                <Number value="1"/>
              </object>
            </property>
            <property name="size">
              <object>
                <reference attribute="children"/>
              </object>
            </property>

          </arguments>
        </property>

      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Wichtig: Seit Cúram Version 6 unterstützen CER und der Abhängigkeitsmanager die Speicherung von berechneten Attributwerten in der Datenbank sowie die automatische Neuberechnung von Attributwerte, falls sich ihre Abhängigkeiten ändern.

Wenn Sie die Implementierung der Methode für eine Eigenschaft ändern, wissen CER und der Abhängigkeitsmanager *nicht* automatisch, dass Attributwerte neu berechnet werden müssen, die unter Verwendung der alten Version der Methode für die Eigenschaft berechnet wurden.

Sobald eine Methode für eine Eigenschaft in einer Produktionsumgebung für gespeicherte Attributwerte verwendet wurde, sollten Sie anstelle einer Änderung der Implementierung eine neue Methode für die Eigenschaft erstellen (mit der erforderlichen neuen Implementierung) und Ihre Regelwerke so ändern, dass die neue Methode für die Eigenschaft verwendet wird. Wenn Sie Ihre Regelwerkänderungen, die auf die neue Methode für die Eigenschaft verweisen, veröffentlichen, berechnet CER automatisch alle Instanzen des betroffenen Attributwerts neu.

rate:
Weitere Informationen finden Sie im Handbuch `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`.

readall:

Dieser Ausdruck ruft alle externen (also durch Client-Code erstellten) Regelobjektinstanzen einer Regelklasse ab. Interne (also aus Regeln erstellte) Regelobjektinstanzen werden mit diesem Ausdruck *nicht* abgerufen.

Weitere Details über die Erstellung von Regelobjekten finden Sie im Abschnitt „Externe und interne Regelobjekte" auf Seite 35.

Seit Cúram Version 6 kann der Ausdruck `readall` verwendet werden, um Instanzen einer Regelklasse aus einem *anderen* Regelwerk abzurufen, indem für das optionale XML-Attribut `ruleset` ein Wert definiert wird.

Seit Cúram Version 6 unterstützt der Ausdruck `readall` ein optionales Element `match`. Dieses Element bewirkt, dass der Ausdruck `readall` nur diejenigen Regelobjekte abruft, deren Wert für ein bestimmtes Attribut mit dem in der Suchbedingung angegebenen Wert übereinstimmt.

Wichtig: Vor Cúram Version 6 konnten Regelobjekte, die mit einer Bedingung übereinstimmen, unter anderem dadurch abgerufen werden, dass der Ausdruck `readall` in einen Ausdruck "filter" (siehe „`filter`" auf Seite 204) eingeschlossen wurde.

Bei CER-Sitzungen, die einen Datenbankdatenspeicher verwenden (siehe „CER-Sitzungen" auf Seite 32), kann mit der in Cúram Version 6 eingeführten Syntax `readall / match` im Allgemeinen jedoch ein besseres Leistungsverhalten erzielt werden. Dies trifft in den folgenden Situationen zu:

- Das Attribut, das den Ausdruck `readall` enthält, wird zuerst berechnet.
- CER und der Abhängigkeitsmanager stellen fest, dass das Attribut, in dem der Ausdruck `readall` enthalten ist, nicht auf dem neuesten Stand ist und neu berechnet werden muss (siehe „Abhängigkeitsmanager" auf Seite 81).

Bei Fällen, in denen Regelobjekte mit mehreren Bedingungen übereinstimmen müssen, sollten Sie mit der Syntax `readall / match` eine Übereinstimmung mit dem selektivsten Attribut erzielen und anschließend die Ergebnisse in einen Ausdruck "filter" (siehe „`filter`" auf Seite 204) einschließen, um eine zusätzliche Filterung nach den übrigen Bedingungen vorzunehmen.

Tipp: Erwarten Sie lediglich eine Singleton-Instanz der Regelklasse (möglicherweise nach einer Filterung oder Übereinstimmung), kann es sinnvoll sein, den Ausdruckk in einen Ausdruck "singleitem" (siehe „`singleitem`" auf Seite 237) einzuschließen.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_readall"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="socialSecurityNumber">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
    Retrieve the one-and-only claim which will have been used to
```

```
seed the session
-->

<Attribute name="claim">
  <type>
    <ruleclass name="Claim"/>
  </type>
  <derivation>
    <singleitem onEmpty="error" onMultiple="error">
      <readall ruleclass="Claim"/>
    </singleitem>
  </derivation>
</Attribute>


<!--
Retrieve the benefit rule objects for this person (created from
client code, probably by querying external storage).

This implementation uses a <readall> with a nested <match> to
retrieve only the matching rule objects, and (depending on data
storage) will be more performant than the
"benefitsFilterReadall" implementation below.
-->

<Attribute name="benefitsReadallMatch">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <readall ruleclass="Benefit">
      <match retrievedattribute="socialSecurityNumber">
        <reference attribute="socialSecurityNumber"/>
      </match>
    </readall>
  </derivation>
</Attribute>


<!--
Retrieves the same rule objects as for "benefitsReadallMatch"
above, but (depending on data storage) may not be as performant.
-->
<Attribute name="benefitsFilterReadall">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- retrieve all Benefit rule objects from external
             storage -->
        <readall ruleclass="Benefit"/>
      </list>
      <listitemexpression>
        <equals>
          <!-- match up the social security numbers on
            the person rule object and the benefit
            rule object -->
          <reference attribute="socialSecurityNumber">
            <current/>
          </reference>
          <reference attribute="socialSecurityNumber"/>
```

```
        </equals>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>

<!--
Retrieves the person's benefits of type "IncomeAssistance",
using a <match> to retrieve all the person's benefits, and then
a <filter> to extract only the "Income Assistance" benefits from
the benefits for that person.

This implementation may be suitable when the
socialSecurityNumber is the most selective attribute for a
Benefit in the data storage (i.e. there are many Benefit rule
objects, but each socialSecurityNumber value is present on
relatively few Benefit rule objects).
-->
<Attribute name="incomeAssistanceBenefitsMatchSSNFilterType">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- retrieve all Benefit rule objects for the Person
-->
        <readall ruleclass="Benefit">
          <match retrievedattribute="socialSecurityNumber">
            <reference attribute="socialSecurityNumber"/>
          </match>
        </readall>
      </list>
      <listitemexpression>
        <equals>
          <!-- filter the Benefit rule objects for the Person
               down to those of type "Income Assistance" only
            -->
          <reference attribute="type">
            <current/>
          </reference>
          <Code table="BenefitType">
            <!-- The value for Income Assistance -->
            <String value="BT1"/>
          </Code>
        </equals>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>


<!--
Retrieves the person's benefits of type "IncomeAssistance",
using a <match> to retrieve all the "Income Assistance"
benefits, and then a <filter> to extract only the "Income
Assistance" benefits for this Person.

This implementation may be suitable when the type is the most
selective attribute for a Benefit in the data storage (i.e.
there are few Benefit rule objects of each type).
-->
<Attribute name="incomeAssistanceBenefitsMatchTypeFilterSSN">
  <type>
    <javaclass name="List">
```

```
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- retrieve all Benefit rule objects of type "Income
             Assistance" -->
        <readall ruleclass="Benefit">
          <match retrievedattribute="type">
            <Code table="BenefitType">
              <!-- The value for Income Assistance -->
              <String value="BT1"/>
            </Code>
          </match>
        </readall>
      </list>
      <listitemexpression>
        <equals>
          <!-- filter the Benefit rule objects of type "Income
               Assistance"  down to those for this Person only
            -->
          <reference attribute="socialSecurityNumber">
            <current/>
          </reference>
          <reference attribute="socialSecurityNumber"/>
        </equals>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>

<!--
Retrieves the rule objects for Benefits whose amount is greater
than 100.

Because "greater than" is not an exact match predicate, a
<filter> must be used (<match> can only be used for an exact
match criterion).
-->
<Attribute name="highPaymentBenefits">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- retrieve all Benefit rule objects for the Person
          -->
        <readall ruleclass="Benefit">
          <match retrievedattribute="socialSecurityNumber">
            <reference attribute="socialSecurityNumber"/>
          </match>
        </readall>
      </list>
      <listitemexpression>
        <!-- filter the Benefit rule objects for the Person down
             to those of amount greater than 100 only -->
        <compare comparison=">">
          <reference attribute="amount">
            <current/>
          </reference>
          <Number value="100"/>
        </compare>
      </listitemexpression>
```

```
      </filter>
    </derivation>
  </Attribute>



</Class>

<Class name="Benefit">

  <Attribute name="socialSecurityNumber">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="type">
    <type>
      <codetableentry table="BenefitType"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="amount">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<!--
This rule set expects that the code creating the session also
creates a "bootstrap" single instance of this Claim rule
class.
-->
<Class name="Claim">
  <Initialization>
    <Attribute name="claimIdentifier">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
    <Attribute name="claimDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
    </Attribute>
  </Initialization>
</Class>

</RuleSet>
```

reference:
Dieser Ausdruck ruft den Wert eines Attributs aus einem Regelobjekt ab.

Der Ausdruck `reference` kann optional einen untergeordneten Ausdruck enthalten, mit dem das Regelobjekt bestimmt wird, aus dem das Attribut abgerufen werden

soll. Wird dieser untergeordnete Ausdruck nicht angegeben, wird das Regelobjekt verwendet, das den Ausdruck `reference` enthält.

Der Ausdruck `reference` ist für die Erstellung von wiederverwendbaren und aussagekräftigen Regeln von wesentlicher Bedeutung. Sie können einen Ausdruck `reference` für ein benanntes Attribut anstelle eines Ausdrucks "any" verwenden. Der CER-Regelwerkvalidierer gibt einen Fehler aus, wenn der Typ des Attributs, auf das verwiesen wird, nicht mit dem vom Ausdruck erforderten Typ übereinstimmt.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_reference"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- A simple reference to another
         attribute on this rule class -->
    <Attribute name="ageNextYear">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <!-- This <reference> has no child element,
               so the rule object is taken to be "this rule
 object"-->
          <reference attribute="age"/>
          <Number value="1"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <Attribute name="favoritePet">
      <type>
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Attribute name="name">
```

```
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="species">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Household">

  <!-- All the people in the household -->
  <Attribute name="members">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- One special person is designated
       as the "head" of the household -->
  <Attribute name="headOfHousehold">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- A reference to an attribute on a
       different rule object:

  name OF favoritePet OF headOfHousehold

  In a programming language, this might be
  written back-to-front using a dereferencing
  "dot" notation like this:

  headOfHousehold.favoritePet.name

  -->
  <Attribute name="nameOfHeadOfHouseholdsFavoritePet">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>

      <!-- The name... -->
      <reference attribute="name">

        <!-- ...of the favorite pet... -->
        <reference attribute="favoritePet">
```

```
            <!-- ...of the head of household. -->
            <!-- The inner-most reference must refer to
                   an attribute on this rule object.  -->
            <reference attribute="headOfHousehold"/>
          </reference>
        </reference>
      </derivation>
    </Attribute>


    <!-- Identifies the dog owners in the household
         by checking which people have a favorite
         pet which is a dog. -->
    <Attribute name="dogOwners">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <!-- simple reference to the members
                 of the household -->
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <equals>
              <String value="Dog"/>
              <!-- A reference to an attribute on an item
                   in the list. -->

              <reference attribute="species">
                <reference attribute="favoritePet">
                  <current/>
                </reference>
              </reference>

            </equals>

          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

removeduplicates:
Dieser Ausdruck erstellt eine neue Liste, indem alle doppelten Einträge aus einer vorhandenen Liste entfernt werden.

Falls ein Eintrag in der ursprünglichen Liste mehrfach enthalten ist, wird nur die erste Instanz beibehalten. Die Reihenfolge der Einträge bleibt ansonsten erhalten.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_removeduplicates"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- The list of relationships where
         this person is the "fromPerson". -->
```

```
  <Attribute name="relationships">
    <type>
      <javaclass name="List">
        <ruleclass name="Relationship"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- The people who are related to this person.

       Any relative appears in this list only once,
       even though one person can be
       related to another in more than one way, e.g.
       my grandparent can also be my legal guardian.-->
  <Attribute name="uniqueRelatives">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <removeduplicates>
        <reference attribute="allRelatives"/>
      </removeduplicates>
    </derivation>
  </Attribute>

  <Attribute name="allRelatives">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <!-- get the relatives of this person by forming a
           list of the "toPerson" on the other end of each
           relationship. -->
      <dynamiclist>
        <list>
          <reference attribute="relationships"/>
        </list>
        <listitemexpression>
          <reference attribute="toPerson">
            <current/>
          </reference>
        </listitemexpression>
      </dynamiclist>
    </derivation>
  </Attribute>

</Class>

<!-- A relationship from one person to another. -->
<Class name="Relationship">

  <Attribute name="fromPerson">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
```

```
    <Attribute name="relationshipType">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>


    <Attribute name="toPerson">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

ResourceMessage:
Dieser Ausdruck erstellt aus einer Eigenschaftenressource eine lokalisierbare Nachricht (siehe „Lokalisierungsunterstützung” auf Seite 9).

Die Eigenschaft kann optional Platzhalter für formatierte Argumente enthalten. Die Unterstützung und die Syntax für die Formatierung ist im JavaDoc für `MessageFormat` beschrieben.

Warnung: Falls die Ausgabe ein einfaches Anführungszeichen oder Hochkomma (') enthalten soll, müssen Sie - wie auch im JavaDoc erläutert - im Eigenschaftentext *zwei* einfache Anführungszeichen ('') angeben.

Wenn die Ausgabe im XML- oder HTML-Format erfolgen soll und keine komplexe Tokenformatierung erfolgen oder keine Möglichkeit zur Änderung des Nachrichtentextes ohne eine Änderung von Regeln bestehen muss, kann es sinnvoll sein, stattdessen den Ausdruck "XmlMessage" (siehe „`XmlMessage`” auf Seite 255) zu verwenden.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ResourceMessage"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="gender">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarried">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
```

```
</Attribute>

<Attribute name="firstName">
  <type>
    <javaclass name="String"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="surname">
  <type>
    <javaclass name="String"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="income">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<!-- Returns a greeting which can be
    output in the user's locale -->
<Attribute name="simpleGreetingMessage">
  <type>
    <javaclass name="curam.creole.value.Message"/>
  </type>
  <derivation>
    <ResourceMessage key="simpleGreeting"
      resourceBundle="curam.creole.example.Messages"/>
  </derivation>
</Attribute>

<!-- Returns a greeting which contains
     the person's title and surname.
     The greeting and title are localized,
     the surname is not (it is identical
     in all locales). -->
<Attribute name="parameterizedGreetingMessage">
  <type>
    <javaclass name="curam.creole.value.Message"/>
  </type>
  <derivation>
    <!-- pass in arguments to
         the message placeholders -->
    <ResourceMessage key="parameterizedGreeting"
      resourceBundle="curam.creole.example.Messages">
      <!-- Title -->
      <choose>
        <type>
          <javaclass name="curam.creole.value.Message"/>
        </type>
        <when>
          <condition>
            <equals>
              <reference attribute="gender"/>
              <String value="Male"/>
            </equals>
          </condition>
```

```
            <value>
              <ResourceMessage key="title.male"
                resourceBundle="curam.creole.example.Messages"/>
            </value>
          </when>
          <when>
            <condition>
              <reference attribute="isMarried"/>
            </condition>
            <value>
              <ResourceMessage key="title.female.married"
                resourceBundle="curam.creole.example.Messages"/>
            </value>
          </when>
          <otherwise>
            <value>
              <ResourceMessage key="title.female.single"
                resourceBundle="curam.creole.example.Messages"/>
            </value>
          </otherwise>
        </choose>

        <!-- Surname -->
        <reference attribute="surname"/>

      </ResourceMessage>
    </derivation>
  </Attribute>

  <!-- Formats a number to 2 decimal places,
       with decimal point and thousands
       separator in the user's locale -->
  <Attribute name="incomeStatementMessage">
    <type>
      <javaclass name="curam.creole.value.Message"/>
    </type>
    <derivation>
      <ResourceMessage key="incomeStatement"
        resourceBundle="curam.creole.example.Messages">
        <reference attribute="income"/>
      </ResourceMessage>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

Beispiel für Eigenschaften, Englisch.

```
# file curam/creole/example/Messages_en.properties

simpleGreeting=Hello
parameterizedGreeting=Hello, {0} {1}
title.male=Mr.
title.female.single=Miss
title.female.married=Mrs.
incomeStatement=Income: USD{0,number,#0.00}
```

Beispiel für Eigenschaften, Französisch.

```
# file curam/creole/example/Messages_fr.properties

simpleGreeting=Bonjour
parameterizedGreeting=Bonjour, {0} {1}
title.male=M.
```

```
title.female.single=Mlle.
title.female.married=Mme.
incomeStatement=Revenue: EUR{0,number,#0.00}
```

singleitem:
Dieser Ausdruck ruft einen einzelnen Eintrag aus einer Liste ab.

Der Ausdruck `singleitem` kann nützlich sein, wenn erwartet wird, dass eine Liste nur einen einzigen Eintrag enthält, beispielsweise beim Filtern einer Liste nach Bedingungen, die lediglich einen einzigen Eintrag in der Liste auswählen sollten.

Der Ausdruck `singleitem` gibt Folgendes an:

- **onEmpty**

  Das Verhalten, wenn festgestellt wird, dass die Liste leer ist. Möglich sind folgende Angaben:

  – **error**

    Ein Laufzeitfehler tritt auf (verwenden Sie diese Option, wenn erwartet wird, dass die Liste nicht leer ist).

  – **returnNull**

    Der Wert *null* wird zurückgegeben.

- **onMultiple**

  Das Verhalten, wenn festgestellt wird, dass die Liste mehrere Einträge enthält. Möglich sind folgende Angaben:

  – **error**

    Ein Laufzeitfehler tritt auf (verwenden Sie diese Option, wenn nicht erwartet wird, dass die Liste mehrere Einträge enthält).

  – **returnNull**

    Der Wert *null* wird zurückgegeben.

  – **returnFirst**

    Der erste Eintrag in der Liste wird zurückgegeben.

  – **returnLast**

    Der letzte Eintrag in der Liste wird zurückgegeben.

Informationen zum Abrufen eines Eintrags an einer bestimmten Position in einer Liste enthalten die Angaben über `get` in „Nützliche Listenoperationen" auf Seite 261.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_singleitem"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="children">
      <type>
        <javaclass name="List">
```

```
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- The first child born to this person -->
  <Attribute name="firstBornChild">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <!-- get the first child, if any
           - if no children, return null -->
      <singleitem onEmpty="returnNull" onMultiple="returnFirst">
        <!-- sort the children in date-of-birth order -->
        <sort>
          <list alias="child">
            <reference attribute="children"/>
          </list>
          <sortorder>
            <sortitem direction="ascending">
              <reference attribute="dateOfBirth">
                <current alias="child"/>
              </reference>
            </sortitem>
          </sortorder>
        </sort>

      </singleitem>
    </derivation>

  </Attribute>

  <!-- Retrieve the single household information record
       from external storage - there should always
       be exactly one - anything else is an error.  -->
  <Attribute name="householdInformation">
    <type>
      <ruleclass name="HouseholdInformation"/>
    </type>
    <derivation>
      <singleitem onEmpty="error" onMultiple="error">
        <readall ruleclass="HouseholdInformation"/>
      </singleitem>
    </derivation>
  </Attribute>

</Class>

<Class name="HouseholdInformation">

  <Attribute name="householdContainsDisabledPerson">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

sort:
Dieser Ausdruck erstellt eine neue Liste, indem die Einträge einer vorhandenen Liste in einer angegebenen Reihenfolge sortiert werden.

Ein Ausdruck `sort` gibt Folgendes an:

- **list**

  Die vorhandene Liste, die sortiert werden soll (und nicht beinflusst wird).
- **sortorder**

  Die Reihenfolge, in der die Liste sortiert werden soll.

Der Ausdruck `sortorder` gibt eines oder mehrere Elemente `sortitem` an, die jeweils den zu sortierenden Eintrag sowie die Sortierreihenfolge (aufsteigend oder absteigend) angeben.

Die Elemente `sortitem` sind mit dem höchstwertigsten Element an erster Stelle aufgeführt. Jedes Element `sortitem` wird nur dann ausgewertet, wenn zwei sortierte Einträge bei höherwertigen Elementen `sortitem` identisch sind.

Innerhalb jedes Elements `sortitem` können Sie mit dem Ausdruck "current" (siehe „current" auf Seite 194) auf den sortierten Listeneintrag verweisen. Normalerweise verweist jedes Element `sortitem` auf ein Attribut oder eine Berechnung im Listeneintrag des Ausdrucks "current" (siehe„current" auf Seite 194).

Falls zwei (oder mehr) Einträge in der Liste im Hinblick auf alle Elemente `sortitem` identisch sind, werden sie in derselben relativen Reihenfolge wie in der Quellenliste zurückgegeben.

Das Verhalten des Ausdrucks `sort` ähnelt der SQL-Klausel `ORDER BY`.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sort"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Arranges the members in order of age (oldest to youngest);
         for members which are the same age, the members are
         arranged in alphabetical order by first name. -->
    <Attribute name="sortedMembers">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <sort>
          <list>
            <reference attribute="members"/>
          </list>
```

```
          <sortorder>
            <sortitem direction="descending">
              <!-- The age of the person in the list -->
              <reference attribute="age">
                <current/>
              </reference>
            </sortitem>
            <!-- The first name of the person in the list -->
            <sortitem direction="ascending">
              <reference attribute="firstName">
                <current/>
              </reference>
            </sortitem>
          </sortorder>
        </sort>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Initialization>
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="age">
        <type>
          <javaclass name="Integer"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>
```

specified:
Dieser Markierungsausdruck macht kenntlich, dass der Wert des Attributs nicht durch die Regelverarbeitung berechnet, sondern extern angegeben wird (z. B. durch das Abrufen aus einer Datenbank oder das Füllen mittels Testcode).

Mit `specified` gekennzeichnete Attribute enthalten normalerweise Informationen, die direkt aus einer systemexternen Position stammen. Andere Attribute verwenden diese externen Informationen, um neue Informationen abzuleiten.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_specified"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- This information cannot be calculated or derived -
         it must be specified from an external source -->
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
    <!-- This information cannot be calculated or derived -
         it must be specified from an external source -->
    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Other attributes are likely to derive/calculation more
         information based on the "specified" attributes above -->
  </Class>
</RuleSet>
```

String:
Dieser Ausdruck gibt einen literalen konstanten Zeichenfolgewert (Typ "String") an.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_String"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="StringExampleRuleClass">

    <Attribute name="emptyString">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- An empty String -->
        <String value=""/>
      </derivation>
    </Attribute>

    <Attribute name="helloWorld">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- The String "Hello, World!" -->
        <String value="Hello, World!"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

sublists:
Dieser Ausdruck berechnet alle Unterlisten der bereitgestellten Liste und gibt diese Unterlisten als Liste von Listen zurück.

Für eine Liste, die n Elemente enthält, gibt es 2^n Unterlisten (inklusive der leeren Liste und der ursprünglichen Liste).

Die Reihenfolge der Listeneinträge in den einzelnen Unterlisten ist mit der Reihenfolge in der ursprünglichen Liste identisch.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sublists"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">
```

```
    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Person"/>
          </listof>
          <members>
            <create ruleclass="Person">
              <String value="Mother"/>
            </create>
            <create ruleclass="Person">
              <String value="Father"/>
            </create>
            <create ruleclass="Person">
              <String value="Child"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- All the different combinations of members of the household
  -->
    <Attribute name="memberCombinations">
      <!-- Note that the type is list of lists of Persons -->
      <type>
        <javaclass name="List">
          <javaclass name="List">
            <ruleclass name="Person"/>
          </javaclass>
        </javaclass>
      </type>
      <derivation>
        <sublists>
          <reference attribute="members"/>
        </sublists>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>
```

In diesem Beispielregelwerk wird der Wert von *memberCombinations* als Liste der folgenden 8 Listen berechnet:

- Liste ohne Inhalt (keine Haushaltsmitglieder)
- Mother

- Father
- Mother und Father
- Child
- Mother und Child
- Father und Child
- Mother, Father und Child (die vollständige ursprüngliche Liste)

sum:
Dieser Ausdruck berechnet die numerische Summe einer Liste von Zahlenwerten (Typ "Number").

Falls die Liste leer ist, gibt dieser Ausdruck das Ergebnis *0* zurück.

Die Liste der Zahlenwerte wird normalerweise durch einen Ausdruck "fixedlist" (siehe „fixedlist" auf Seite 206) oder "dynamiclist" (siehe „dynamiclist" auf Seite 197) bereitgestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sum"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="netWorth">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Example of <sum> operating on a <fixedlist> -->
        <!-- A person's net worth is the sum of their
             cash, savings and assets -->
        <sum>
          <fixedlist>
            <listof>
              <javaclass name="Number"/>
            </listof>
            <members>
              <reference attribute="totalCash"/>
              <reference attribute="totalSavings"/>
              <reference attribute="totalAssets"/>
            </members>
          </fixedlist>
        </sum>
      </derivation>
    </Attribute>

    <Attribute name="totalAssets">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Example of <sum> operating on a <dynamiclist> -->
        <!-- The total value of a person's assets is derived by
             summing the value of each asset -->
        <sum>
          <dynamiclist>
            <list>
              <reference attribute="assets"/>
            </list>
            <listitemexpression>
              <reference attribute="value">
                <current/>
```

```
            </reference>
          </listitemexpression>
        </dynamiclist>
      </sum>
    </derivation>
  </Attribute>

  <!-- The assets of that this person owns -->
  <Attribute name="assets">
    <type>
      <javaclass name="List">
        <ruleclass name="Asset"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- NB this example doesn't show how
       total cash/savings is derived -->
  <Attribute name="totalCash">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
  <Attribute name="totalSavings">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Asset">

  <!-- The monetary value of the asset -->
  <Attribute name="value">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

this:
Dieser Ausdruck verweist auf das aktuelle Regelobjekt (analog zum Schlüsselwort this in Java).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_this"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
```

```
    <!-- The pets owned by this Person -->
    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Pet"/>
          </listof>
          <members>
            <!-- Every Person has exactly two pets,
                 Skippy and Lassie -->
            <create ruleclass="Pet">
              <!-- set the owner to be THIS Person -->
              <this/>
              <String value="Skippy"/>
              <String value="Kangaroo"/>
            </create>
            <create ruleclass="Pet">
              <!-- set the owner to be THIS Person -->
              <this/>
              <String value="Lassie"/>
              <String value="Dog"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Initialization>
      <Attribute name="owner">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="species">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>
```

Timeline:
Dieser Ausdruck erstellt eine Zeitlinie (siehe Abschnitt „Verarbeitung von Daten, die sich mit der Zeit ändern” auf Seite 52) eines bestimmten Typs mit Werten, die ab bereitgestellten Datumsangaben gültig sind.

Eine Zeitlinie muss einen Wert für den Zeitbeginn (Datum `null`) besitzen. Um dies zu unterstützen, enthält der Ausdruck `Timeline` ein optionales Element `initialva-`

lue, mit dem der Wert angegeben wird, der ab dem Zeitbeginn gültig ist. Wird dieses Element nicht verwendet, *muss* die verwendete Gruppe des Ausdrucks "interval" (siehe „Interval" auf Seite 210) ein Intervall mit einem Startdatum null enthalten, da andernfalls ein Fehler auftritt, wenn dieser Ausdruck zur Laufzeit ausgewertet wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Timeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateTimelines">

    <!-- This example uses <initialvalue> to set the value valid
         from the start of time. -->
    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <!-- Value from start of time -->
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- The remaining intervals -->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <Number value="10000"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
                  <start>
                    <Date value="2004-12-01"/>
                  </start>
                  <value>
                    <Number value="12000"/>
                  </value>
                </Interval>

              </members>
            </fixedlist>

          </intervals>
```

```
    </Timeline>

  </derivation>
</Attribute>


<!-- This example does not use <initialvalue>. -->
<Attribute name="aStringTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="String"/>
    </javaclass>
  </type>
  <derivation>
    <Timeline>
      <intervaltype>
        <javaclass name="String"/>
      </intervaltype>

      <!-- The list of intervals must include one valid from the
         null date (start of time), otherwise an error will
         occur at runtime, if this expression is evaluated. -->
      <intervals>
        <fixedlist>
          <listof>
            <javaclass name="curam.creole.value.Interval">
              <javaclass name="String"/>
            </javaclass>
          </listof>
          <members>
            <Interval>
              <intervaltype>
                <javaclass name="String"/>
              </intervaltype>
              <start>
                <!-- "from the start of time" -->
                <null/>
              </start>
              <value>
                <String value="Start of time string"/>
              </value>
            </Interval>
            <Interval>
              <intervaltype>
                <javaclass name="String"/>
              </intervaltype>
              <start>
                <Date value="2001-01-01"/>
              </start>
              <value>
                <String value="2001-only String"/>
              </value>
            </Interval>
            <Interval>
              <intervaltype>
                <javaclass name="String"/>
              </intervaltype>
              <start>
                <Date value="2002-01-01"/>
              </start>
              <value>
                <String value="2002-onwards String"/>
              </value>
            </Interval>
          </members>
        </fixedlist>
      </intervals>
```

```
        </Timeline>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>
```

timelineoperation:
Dieser Ausdruck setzt eine Zeitlinie (siehe Abschnitt „Verarbeitung von Daten, die sich mit der Zeit ändern" auf Seite 52) aus wiederholten Aufrufen eines untergeordneten Ausdrucks zusammmen. Normalerweise wird der Ausdruck `timelineoperation` zusammen mit dem Ausdruck "intervalvalue" (siehe „`intervalvalue`" auf Seite 211) verwendet. Gemeinsam ermöglichen diese beiden Ausdrücke, dass andere Ausdrücke Operationen für Werte aus Zeitlinien ausführen (als ob es sich um Basiselementwerte handeln würde) und dass die resultierenden Daten anschließend in einer Zeitlinie neu zusammengesetzt werden.

Tipp: Bei jeder der Zeitlinien, die als Eingabe für einen Algorithmus verwendet wird, sollte der Ausdruck, der die Zeitlinie zurückgibt, normalerweise in einen Ausdruck "intervalvalue" (siehe „`intervalvalue`" auf Seite 211) eingeschlossen und anschließend das Gesamtergebnis in einen Ausdruck `timelineoperation` eingeschlossen werden.

Nachfolgend ist kurz beschrieben, wie sich der Ausdruck `timelineoperation` zur Auswertungszeit verhält:

- Der Ausdruck `timelineoperation` erstellt einen neuen Auswertungskontext, um die Serie der Aufrufe zu protokollieren, die an seinen untergeordneten Ausdruck erfolgen (der untergeordnete Ausdruck wird in der Regel mehrmals für unterschiedliche Datumsangaben aufgerufen).
- Der Ausdruck `timelineoperation` ruft seinen einzelnen untergeordneten Ausdruck mit dem Kontextdatum `null` auf, das den Zeitbeginn kennzeichnet.
- Während der Auswertung des untergeordneten Ausdrucks (und seiner Abhängigkeitsprodukte) werden bei jeder Feststellung eines Ausdrucks "intervalvalue" (siehe „`intervalvalue`" auf Seite 211) die folgenden Aktionen ausgeführt:
  - Der Ausdruck "intervalvalue" („`intervalvalue`" auf Seite 211) wertet seinen eigenen untergeordneten Ausdruck aus, um eine Zeitlinie zu erhalten. Aus dieser Zeitlinie wird der Wert für das Datum abgerufen, das dem aktuellen Datum im Auswertungskontext des Ausdrucks `timelineoperation` entspricht.
  - Der Ausdruck "intervalvalue" („`intervalvalue`" auf Seite 211) überprüft die anderen Datumsangaben, bei denen sich der Wert in der Zeitlinie ändert. Jede dieser Datumsangaben wird zu einer Warteschlange von anderen Datumsangaben hinzugefügt, die von dem Ausdruck `timelineoperation` (in einem nachfolgenden Aufruf) verarbeitet werden sollen.
- Sobald die Steuerung an den Ausdruck `timelineoperation` zurückgegeben wird, wird für ein bestimmtes Datum ein Wert berechnet und zusätzliche Datumsangaben identifiziert, an denen sich der Wert der Eingabezeitlinie ändert. Für jedes Datum ruft der Ausdruck `timelineoperation` den untergeordneten Ausdruck erneut auf, bis die Warteschlange keine Datumsangaben mehr enthält.

Das zuvor beschriebene Verhalten bedeutet, dass die verschachtelten Ausdrücke in keinem Fall Kenntnis darüber haben müssen, dass sie im Rahmen der Verarbeitung von Zeitlinien eingesetzt werden. Darüber hinaus ist die Verarbeitung effizient, weil Ausdrücke nur für diejenigen Datumsangaben aufgerufen werden, an denen sich der Wert der Eingabezeitlinien ändert.

Anmerkung: Falls ein Ausdruck `timelineoperation` für einen Ausdruck ausgeführt wird, der *nicht* in einen Ausdruck "intervalvalue" (siehe „`intervalvalue`" auf Seite 211) eingeschlossen ist, enthält die resultierende Zeitlinie für den gesamten Zeitraum einen konstanten Wert.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_timelineoperation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <!--
     true during a person's lifetime; false before date of birth,
     and false again after date of death (if any)
      -->
    <Attribute name="isAliveTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Boolean"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
     The assets owned by the person, at some time or another.  Each
     asset's value may vary over time.
      -->
    <Attribute name="ownedAssets">
      <type>
        <javaclass name="List">
          <ruleclass name="Asset"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>



    <!--
     The total value of the assets owned by the person (or by the
     person's estate, if the person has died).
      -->
    <Attribute name="totalAssetValueTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <!--
         Total the value of all the owned assets.  The value of each
         asset may change over time.
          -->

        <!--
         <timelineoperation> will create a timeline from the series
         of <sum> calculations performed within it.

         Each execution of <sum> will calculate the total for a
         particular day; <timelineoperation> will assemble these
         daily totals into a timeline of numbers.
```

```
      -->
    <timelineoperation>

      <sum>
        <!--
         For each owned asset, get its countable-value timeline.
          -->
        <dynamiclist>
          <list>
            <reference attribute="ownedAssets"/>
          </list>
          <listitemexpression>

            <!--
             Wrap the timeline returned by
             countableValueTimeline, so that the <sum> thinks
             that it is operating on a list of numbers, not a
             list of timelines.
              -->
            <intervalvalue>
              <reference attribute="countableValueTimeline">
                <current/>
              </reference>
            </intervalvalue>
          </listitemexpression>
        </dynamiclist>

      </sum>

    </timelineoperation>

  </derivation>
</Attribute>


<!--
 The cut-off point for being entitled to benefit.  Persons with
 assets above this varying threshold do not qualify for benefit.

  -->
<Attribute name="maximumAssetsThreshold">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <!--
     In a real implementation, this value would tend to vary
     over time (e.g. from a rate table).

     However, for the sake of example, this implementation uses
     a <timelineoperation> WITHOUT a nested <intervalvalue> to
     create a Timeline which has a constant value for all time.

     This use of <timelineoperation> can often be useful for
     dummy implementations early on in rule set development.
    -->

    <timelineoperation>
      <!--
       Hard coded forever-constant value - to be replaced by a
       varying value later in rules development.
        -->
      <Number value="10000"/>
    </timelineoperation>
  </derivation>
```

```
  </Attribute>

  <!--
   The person qualified for benefit if (on any particular day)
   the person is alive and the total value of the person's assets
   does not exceed the maximum asset threshold.
    -->
  <Attribute name="qualifiesForBenefitTimeline">
    <type>
      <javaclass name="curam.creole.value.Timeline">
        <javaclass name="Boolean"/>
      </javaclass>
    </type>
    <derivation>
      <timelineoperation>
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!--
               operate on the Timelines as if they were primitive
               values
                -->
              <intervalvalue>
                <reference attribute="isAliveTimeline"/>
              </intervalvalue>

              <compare comparison="&lt;=">
                <intervalvalue>
                  <reference attribute="totalAssetValueTimeline"/>
                </intervalvalue>
                <intervalvalue>
                  <reference attribute="maximumAssetsThreshold"/>
                </intervalvalue>
              </compare>
            </members>
          </fixedlist>

        </all>
      </timelineoperation>

    </derivation>
  </Attribute>



</Class>

<!--
 An asset, owned by a person at some time or other.

 Each asset is bought, and may subsequently be sold.

 The value of an asset varies over time; the asset still has a
 value even before or after it is owned by a Person; however,
 the _countable_ value is zero outside of the period of
 ownership.
  -->
<Class name="Asset">
  <Attribute name="boughtDate">
    <type>
      <javaclass name="curam.util.type.Date"/>
    </type>
    <derivation>
      <specified/>
```

```
  </derivation>
</Attribute>

<!-- will be null if the asset has not been sold -->
<Attribute name="soldDate">
  <type>
    <javaclass name="curam.util.type.Date"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="isOwnedTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Boolean"/>
    </javaclass>
  </type>
  <derivation>
    <existencetimeline>
      <intervaltype>
        <javaclass name="Boolean"/>
      </intervaltype>
      <intervalfromdate>
        <reference attribute="boughtDate"/>
      </intervalfromdate>
      <intervaltodate>
        <reference attribute="soldDate"/>
      </intervaltodate>
      <preExistenceValue>
        <false/>
      </preExistenceValue>
      <existenceValue>
        <true/>
      </existenceValue>
      <postExistenceValue>
        <false/>
      </postExistenceValue>
    </existencetimeline>

  </derivation>
</Attribute>

<!--
 the varying value of the asset, regardless of whether it is
 owned by the person at the time
  -->
<Attribute name="valueTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>


<!--
 The value which counts towards the person's assets - i.e. the
 value of the asset during the period when it is owned,
 otherwise 0 when it is not owned.
  -->
<Attribute name="countableValueTimeline">
  <type>
```

```
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <!--
         reassemble the outputs from each <choose> invocation into a
         timeline
          -->
        <timelineoperation>
          <choose>
            <type>
              <javaclass name="Number"/>
            </type>
            <when>
              <condition>
                <!--
                 operate on each of the intervals of constant
                 ownership
                -->

                <intervalvalue>
                  <reference attribute="isOwnedTimeline"/>
                </intervalvalue>
              </condition>
              <value>
                <!--
                 if on a particular date, the asset is owned, then
                 its countable value on that date is simply its
                 value
                  -->
                <intervalvalue>
                  <reference attribute="valueTimeline"/>
                </intervalvalue>
              </value>
            </when>
            <otherwise>
              <value>
                <!--
                 if on a particular date, the asset is owned, then
                 its countable value on that date is zero
                  -->
                <Number value="0"/>
              </value>
            </otherwise>
          </choose>

        </timelineoperation>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Tipp: Falls ein verschachtelter Ausdruck eine Zeitlinie zurückgibt und Sie vergessen, diesen Ausdruck in einen Ausdruck "intervalvalue" (siehe „intervalvalue" auf Seite 211) einzuschließen, werden CER-Gültigkeitsfehler wie im folgenden Beispiel angezeigt:

```
<!--
     The value which counts towards the person's assets - i.e. the
     value of the asset during the period when it is owned,
     otherwise 0 when it is not owned.
      -->
    <Attribute name="countableValueTimeline">
      <type>
```

```
      <javaclass name="curam.creole.value.Timeline">
        <javaclass name="Number"/>
      </javaclass>
    </type>
    <derivation>
      <!--
       reassemble the outputs from each <choose> invocation into a
       timeline
        -->
      <timelineoperation>
        <choose>
          <type>
            <javaclass name="Number"/>
          </type>
          <when>
            <condition>
              <!--
               operate on each of the intervals of constant
               ownership
              -->

              <!--
                **** Forgot to wrap the Timeline returned
                in <intervalvalue> ****
                -->
                <reference attribute="isOwnedTimeline"/>
            </condition>
            <value>
              <!--
               if on a particular date, the asset is owned, then
               its countable value on that date is simply its
               value
                -->
              <intervalvalue>
                <reference attribute="valueTimeline"/>
              </intervalvalue>
            </value>
          </when>
          <otherwise>
            <value>
              <!--
               if on a particular date, the asset is owned, then
               its countable value on that date is zero
                -->
              <Number value="0"/>
            </value>
          </otherwise>
        </choose>

      </timelineoperation>
    </derivation>
  </Attribute>
```

Beispielfehler.

```
ERROR ... Example_timelineoperation.xml(276, 19)
  AbstractRuleItem:INVALID_CHILD_RETURN_TYPE: Child 'condition' returns
  'curam.creole.value.Timeline<? extends java.lang.Boolean>',
  but this item requires a 'java.lang.Boolean'.
```

true:
Dies ist der boolesche konstante Wert "true".

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_true"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
```

```
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="TrueExampleRuleClass">

    <Attribute name="isCuramExpertRulesFantastic">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <true/>
      </derivation>
    </Attribute>

    <Attribute name="didCookbookWinPulitzerPrize">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <true/>
        </not>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

XmlMessage:
Dieser Ausdruck erstellt aus frei formatiertem XML-Inhalt eine lokalisierbare Nachricht (siehe „Lokalisierungsunterstützung” auf Seite 9).

Die erstellte Nachricht ist der Inhalt mit XML-Literalcodierung im Element `XmlMessage`. Es gibt jedoch eine Ausnahme: Als Inhalt eines Elements `replace` wird der Ausdruck festgelegt, den es enthält.

Das Element `replace` stellt einen Ersatzmechnismus für einfache Token dar. Falls eine komplexere Tokenformatierung erforderlich ist oder die Möglichkeit bestehen muss, Nachrichtentext ohne die Änderung von Regeln zu ändern, kann es sinnvoll sein, stattdessen den Ausdruck "ResourceMessage" (siehe „`ResourceMessage`” auf Seite 234) zu verwenden.

Anmerkung: Vor Cúram Version 6 wurden bei Verwendung des Ausdrucks `XmlMessage` Leerzeichen abgeschnitten, die eingebettete XML-Zeichen umgaben. Ab Cúram Version 6 werden Leerzeichen bei Verwendung von `XmlMessage` nicht mehr abgeschnitten und das Quellenformat der XML-Zeichen bleibt (unter Entfernung von XML-Kommentaren) erhalten.

Falls Sie das Abschneideverhalten benötigen, das der Ausdruck `XmlMessage` vor Cúram Version 6 aufwies, müssen Sie die Anwendungsumgebungsvariable *curam.creole.XmlFormat.enableWhitespaceTrimming* in Ihrer Entwicklungsumgebung auf den Wert *true* setzen.

In einer Produktionsumgebung sollten Sie bei einer dynamischen Änderung der Umgebungsvariablen *curam.creole.XmlFormat.enableWhitespaceTrimming* sicherstellen, dass alle abgeleiteten Daten in Ihrem System, die von Regelattributen abhängig sind, bei denen der Ausdruck `XmlMessage` verwendet wird, zwingend alle gespeicherten Attributwertinstanzen neu berechnen müssen.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_XmlMessage"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
```

```
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="XmlMessageExampleRuleClass">

    <Attribute name="emptyMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- contains no XML at all -->
        <XmlMessage/>
      </derivation>
    </Attribute>

    <Attribute name="simpleHtmlMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Using XmlMessage can ensure that
             XML elements are started and
             ended correctly, e.g. <b> and </b> -->
        <XmlMessage>The following text will appear in bold in a
          browser: <b>Some in bold text.</b>
        </XmlMessage>
      </derivation>
    </Attribute>

    <Attribute name="tokenReplacementHtmlMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <XmlMessage><p/>This calculated number will appear in
          italics and formatted according to locale preferences:<i>
            <replace>
              <arithmetic operation="+">
                <Number value="1.23"/>
                <Number value="3.45"/>
              </arithmetic>
            </replace>
          </i>
          <p/>And here's a resource message: <replace>
            <ResourceMessage key="simpleGreeting"
              resourceBundle="curam.creole.example.Messages"/>
          </replace>
        </XmlMessage>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Anmerkungen

Seit Cúram Version 6 unterstützt CER so genannte "Anmerkungen". Eine Anmerkung sind zusätzliche Metadaten in einem Regelwerk, die für CER-Clients verfügbar sind, sich jedoch *nicht* auf das Verhalten von CER-Berechnungen auswirken.

CER-Clients verwenden Anmerkungen, um ihr Verhalten bei der Interaktion mit CER-Regelwerken zu steuern.

Anmerkungen können für die folgenden Elemente in einem CER-Regelwerk angegeben werden (allerdings kann jede Anmerkung eine Validierung enthalten, die ihre Einsatzpositionen einschränkt):

- Regelwerk (siehe „Regelwerk” auf Seite 165)
- Regelklasse (siehe „Regelklasse” auf Seite 167)
- Regelattribut (siehe „Attribut” auf Seite 169)
- Ausdruck (siehe „Ausdrücke” auf Seite 169)

Jedes Regelelement kann keine, eine oder viele Anmerkungen enthalten. Jeder Typ Anmerkung darf höchstens ein Mal für jedes Regelelement angegeben sein. Ein Regelwerk kann beispielsweise sowohl eine Anmerkung des Typs `Label` als auch eine Anmerkung des Typs `EditorMetadata`, jedoch nicht mehrere Anmerkungen des Typs `Label` enthalten.

Vollständige alphabetische Liste der Anmerkungen

Dieser Abschnitt enthält Definitionen für alle Anmerkungen, die in der Anwendung enthalten sind.

Anmerkung: Einige Anmerkungen dienen geschäftsspezifischen Ableitungen in der Anwendung. Solche Anmerkungen sind hier zwar aufgeführt, jedoch unter Verweis auf andere Cúram-Handbücher, in denen diese Anmerkungen in ihrem Geschäftskontext beschrieben sind.

`ActiveInEditSuccessionSetPopulation`:
Weitere Informationen enthält das Handbuch `Cúram Advisor Configuration Guide`.

`Display`:
Weitere Informationen finden Sie im Handbuch `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`.

`DisplaySubscreen`:
Weitere Informationen finden Sie im Handbuch `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`.

`EditorMetadata`:
Diese Anmerkung speichert Diagramminformationen für ein Regelwerk. Sie wird vom CER-Editor automatisch verwaltet.

Diese Anmerkung darf nur in einem Regelwerk angegeben werden (siehe „Regelwerk” auf Seite 165).

`Indexed`:
Diese Anmerkung wird nicht mehr verwendet. Sie ist hier lediglich aus Gründen der Abwärtskompatibilität aufgeführt.

`Label`

Diese Anmerkung stellt eine lokalisierte Beschreibung für eines der folgenden Regelwerkelemente bereit:

- Regelwerk (siehe „Regelwerk” auf Seite 165)
- Regelklasse (siehe „Regelklasse” auf Seite 167)
- Regelattribut (siehe „Attribut” auf Seite 169)
- Ausdruck (siehe „Ausdrücke” auf Seite 169)

Die Anmerkung "Label" enthält eine (vom CER-Editor festgelegte) Kennung sowie eine (vom Benutzer eingegebene) Beschreibung.

Wenn ein CER-Regelwerk gespeichert oder veröffentlicht wird, werden die Werte in den Anmerkungen "Label" einer Regel verwendet, um (in der Ländereinstellung des Benutzers) eine Eigenschaftenressource in den Anwendungsressourcenspeicher zu schreiben. Umgekehrt wird beim Anzeigen eines Regelwerks im CER-Editor die Eigenschaftenressource für die Ländereinstellung des Benutzers aus dem Ressourcenspeicher abgerufen und zum Füllen der Werte für die Anmerkungen "Label" im Regelwerk-XML verwendet.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Label"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Annotations>
    <!-- Rule-set level description -->
    <Label name="Example rule set for labels"
 label-id="annotation1"/>
  </Annotations>
  <Class name="Person">
    <Annotations>
      <!-- Rule-class level description -->
      <Label name="A Person" label-id="annotation2"/>
    </Annotations>
    <Attribute name="age">
      <Annotations>
        <!-- Attribute-class level description -->
        <Label name="The current age of the person, in years"
 label-id="annotation3"/>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified>
          <Annotations>
            <!-- Expression-level description -->
            <Label name="This value comes directly from evidence"
 label-id="annotation4"/>
          </Annotations>
        </specified>
      </derivation>
    </Attribute>

    <Attribute name="ageNextBirthday">
      <Annotations>
        <!-- Attribute-class level description -->
        <Label name="The age of the person at the person's next
 birthday, in years"
          label-id="annotation5"/>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <Annotations>
            <!-- Expression-level description -->
            <Label name="Compute the person's age at next birthday"
 label-id="annotation6"/>
          </Annotations>
          <reference attribute="age">
            <Annotations>
              <!-- Expression-level description -->
              <Label name="Get the person's current age"
 label-id="annotation7"/>
```

```
          </Annotations>
        </reference>
        <Number value="1">
          <Annotations>
            <!-- Expression-level description -->
            <Label name="The number to add to get the age next
 birthday" label-id="annotation8"/>
          </Annotations>
        </Number>
      </arithmetic>
    </derivation>
  </Attribute>


 </Class>

</RuleSet>
```

Legislation

Weitere Informationen enthält das Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

SuccessionSetPopulation

Weitere Informationen enthält das Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

primary

Gibt das primäre Attribut für eine Regelklasse wie im CER-Editor festgelegt an.

Diese Anmerkung kann nur für eine Regelklasse angegeben werden (siehe „Regelklasse” auf Seite 167). Das benannte Attribut muss exakt den Namen eines für die Regelklasse deklarierten Attributs angeben (es kann nicht verwendet werden, um ein Attribut zu benennen, das übernommen, jedoch nicht überschrieben wurde).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_primary"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">


  <Class name="Person">
    <Annotations>
      <!-- Declaration of the "primary" rule attribute for this rule
 class, as shown in the CER Editor -->
      <primary attribute="age"/>
    </Annotations>
    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

relatedActiveInEditSuccessionSet

Weitere Informationen enthält das Handbuch Cúram Advisor Configuration Guide.

relatedEvidence

Weitere Informationen enthält das Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

relatedSuccessionSet

Weitere Informationen enthält das Handbuch Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

tags

Diese Anmerkung ordnet den folgenden Elementen beliebige Zeichenfolgetags zu:

- Regelwerk (siehe „Regelwerk" auf Seite 165)
- Regelklasse (siehe „Regelklasse" auf Seite 167)
- Regelattribut (siehe „Attribut" auf Seite 169)
- Ausdruck (siehe „Ausdrücke" auf Seite 169)

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_tags"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Annotations>
    <!-- Rule-set level tags -->
    <tags>
      <tag value="A rule-set tag"/>
      <tag value="Another tag"/>
    </tags>
  </Annotations>
  <Class name="Person">
    <Annotations>
      <!-- Rule-class level tags-->
      <tags>
        <tag value="A rule-class tag"/>
        <tag value="Another tag"/>
      </tags>
    </Annotations>
    <Attribute name="age">
      <Annotations>
        <!-- Attribute-class level tags -->
        <tags>
          <tag value="A rule-attribute tag"/>
          <tag value="Another tag"/>
        </tags>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified>
          <Annotations>
            <!-- Expression-level tags -->
            <tags>
              <tag value="An expression tag"/>
              <tag value="Another tag"/>
            </tags>
```

```
          </Annotations>
        </specified>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Nützliche Listenoperationen

Der Ausdruck `property` lässt den Aufruf von "sicheren" Java-Methoden zu.

Siehe „Eigenschaft" auf Seite 133 und „`property`" auf Seite 221.

Regelwerke enthalten häufig viele Instanzen von `java.util.List`.

Die Sicherheitsliste der Methoden für `java.util.List` ist in CER enthalten.

Sicherheitsliste für Methoden `java.util.List`.

```
# Safe list for java.util.List

            contains.safe=true
            containsAll.safe=true


            get.safe=true

            indexOf.safe=true
            isEmpty.safe=true
            lastIndexOf.safe=true
            size.safe=true
            subList.safe=true


            # not exposed
            hashCode.safe=false
            listIterator.safe=false
            iterator.safe=false
            toArray.safe=false

            # mutators - unsafe
            add.safe=false
            addAll.safe=false
            clear.safe=false
            remove.safe=false
            removeAll.safe=false
            retainAll.safe=false
```

Die Beschreibungen dieser Methoden sind im JavaDoc für `java.util.List` verfügbar. Die besonders nützlichen Eigenschaften sollen hier zur Referenz jedoch kurz beschrieben werden:

- **isEmpty()**

  Gibt *true* zurück, wenn diese Liste keine Elemente enthält.
- **size()**

  Gibt die Anzahl der Elemente in dieser Liste zurück.
- **get(int index)**

Gibt das Element an der angegebenen Position in dieser Liste zurück. Bitte beachten Sie, dass Sie einen Wert des Typs `Number` mit `intValue` in eine Ganzzahl konvertieren müssen, weil CER gerundete numerische Werte als Instanzen des Typs `Number` übergibt.

- **contains(Object o)**

  Gibt *true* zurück, falls diese Liste das angegebene Element enthält.

```
<?xml version="1.0" encoding="UTF-8"?>
            <RuleSet name="Example_UsefulListOperations"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
            "http://www.curamsoftware.com/CreoleRulesSchema.xsd">

            <Class name="Person">

            <!-- Exactly one Person (in each household) will
            be designated as the head of household -->
            <Attribute name="isHeadOfHousehold">
            <type>
            <javaclass name="Boolean"/>
            </type>
            <derivation>
            <specified/>
            </derivation>
            </Attribute>

            <!-- The children of this person. -->
            <Attribute name="children">
            <type>
            <javaclass name="List">
            <ruleclass name="Person"/>
            </javaclass>
            </type>
            <derivation>
            <specified/>
            </derivation>
            </Attribute>

            <!-- Whether this person has any children.

            Tests the isEmpty property of List. -->
            <Attribute name="hasChildren">
            <type>
            <javaclass name="Boolean"/>
            </type>
            <derivation>
            <not>
            <property name="isEmpty">
            <object>
            <reference attribute="children"/>
            </object>
            </property>
            </not>
            </derivation>
            </Attribute>

            <Attribute name="numberOfChildren">
            <type>
            <javaclass name="Number"/>
            </type>
            <derivation>
            <property name="size">
            <object>
            <reference attribute="children"/>
            </object>
            </property>
```

```
</derivation>
</Attribute>

<!-- This person's second child, if any, otherwise null -->
<Attribute name="secondChild">
<type>
<ruleclass name="Person"/>
</type>
<derivation>
<!-- We have to check whether the person has two
or more children -->
<choose>
<type>
<ruleclass name="Person"/>
</type>
<when>
<condition>
<compare comparison=">=">
<reference attribute="numberOfChildren"/>
<Number value="2"/>
</compare>
</condition>
<value>
<!-- Use the "get" property to get the second item
in the list - denoted by index 1 (lists in
Java are zero-based) -->
<property name="get">
<object>
<reference attribute="children"/>
</object>
<arguments>
<!-- The number must be converted to an integer
(as required by List.subList). -->
<property name="intValue">
<object>
<Number value="1"/>
</object>
</property>

</arguments>
</property>
</value>
</when>
<otherwise>
<!-- This person has no second child -->
<value>
<null/>
</value>
</otherwise>
</choose>
</derivation>
</Attribute>

<Attribute name="isChildOfHeadOfHousehold">
<type>
<javaclass name="Boolean"/>
</type>
<derivation>
<property name="contains">
<object>
<!-- The children of the head of household -->
<reference attribute="children">
<!-- retrieve the single Person rule object which
has isHeadOfHousehold equal to true-->
<singleitem onEmpty="error" onMultiple="error">
<readall ruleclass="Person">
<match retrievedattribute="isHeadOfHousehold">
```

```
<true/>
</match>
</readall>
</singleitem>
</reference>
</object>
<!-- check whether the list of the head of household's
children contains THIS Person -->
<arguments>
<this/>
</arguments>
</property>
</derivation>
</Attribute>

</Class>

</RuleSet>
```

CER mit Datenspeicher verwenden

Die Anwendung enthält Integrationscode, der aus Einträgen im Anwendungsdatenspeicher CER-Regelobjekte erstellen kann. Die Funktion `DataStoreRuleObjectCreator` wird vom Modul "Universal Access" (Universalzugriff) verwendet, um Angaben, die durch ein IEG-Script erfasst wurden, in CER-Regelobjekte zu konvertieren. In den nachfolgenden Abschnitten wird die Funktionsweise der Funktion `DataStoreRuleObjectCreator` beschrieben.

Funktion `DataStoreRuleObjectCreator`

Die Funktion `DataStoreRuleObjectCreator` verwendet einen Datenspeichersatz (normalerweise ein Datensatz für einen Benutzer oder eine Person) als Eingabe und navigiert zu allen untergeordneten Datensätzen dieses "Stammdatensatzes" (normalerweise mit allen erfassten Angaben für die Person).

Anschließend werden durch eine unkomplizierte "wirklichkeitsgetreue Zuordnung" zwischen den folgenden Elementen Regelobjekte erstellt:

- Entitätstypen und Attribute im Datenspeicherschema
- Regelklassen und Regelattribute im CER-Regelwerk

Die Funktion `DataStoreRuleObjectCreator` führt außerdem besondere Aktionen für CER-Regelattribute mit bestimmten Namen aus:

- **parentEntity**

  Falls eine Regelklasse ein Regelattribut namens `parentEntity` enthält, legt die Funktion `DataStoreRuleObjectCreator` seinen Wert mit dem Regelobjekt fest, das aus dem übergeordneten Datensatz im Datenspeicher (sofern vorhanden) erstellt wurde. CER gibt einen Laufzeitfehler aus, wenn der Typ dieses Regelattributs nicht mit der Regelklasse des Regelobjekts für die übergeordnete Entität übereinstimmt.

- **`childEntities_<regelklassenname>`**

  Falls eine Regelklasse Attribute namens `childEntities_` gefolgt vom Namen einer Regelklasse enthält, legt die Funktion `DataStoreRuleObjectCreator` den Wert für jedes solcher Attribute mit einer Liste der Regelobjekte fest, die aus den untergeordneten Datensätzen dieses Typs im Datenspeicher (sofern vorhanden) erstellt wurden. CER gibt einen Laufzeitfehler aus, falls der Typ dieses Regelattributs keine Liste der benannten Regelklasse ist.

Beispiel

Das Verhalten der Funktion `DataStoreRuleObjectCreator` lässt sich am besten anhand eines Beispiels veranschaulichen. Dieses Beispiel basiert auf dem Modul "Universal Access".

Ein IEG-Script kann Einkommensdaten für einen Haushalt erfassen. Der Haushalt kann eine beliebige Anzahl von Personen umfassen und für jede Person kann es eine beliebige Anzahl von Einkommensdetails geben.

Die Struktur des Datenspeicherschemas ist nachfolgend vereinfacht dargestellt:

- Application
  - Person (0..n)
    - firstName (Zeichenfolge)
    - lastName (Zeichenfolge)
    - Income (0..n)
      - type (Code aus Codetabelle "IncomeType")
      - amount (Zahl)

Ein Bürger (John) zeichnet im Rahmen eines Self-Screenings Angaben für seinen Haushalt (der aus ihm selbst und seiner Frau Mary besteht) sowie Einkommensdetails auf (John ist arbeitssuchend, Mary führt zwei Teilzeittätigkeiten aus). Johns Angaben werden im Datenspeicher als Datensätze gespeichert:

- Application #1234
  - Person #1235
    - firstName: John
    - lastName: Smith
    - Income <keine Datensätze>
  - Person #1236
    - firstName: Mary
    - lastName: Smith
    - Income #1238
      - type: Part-time
      - amount 30

Das CER-Regelwerk, das für die Verwendung bei diesem Screeningtyp konfiguriert ist, enthält dieselben Regelklassen (Programmregelklassen sind nicht aufgeführt):

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="DataStoreMappingExample"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- NB no rule class for the CDS entity type "Application";
       the attributes stored directly on an Application are
       not required by rules, so no point creating a rule object
       which won't get used. -->

  <!-- The name of this rule class matches that of a CDS entity
       type -->
  <Class name="Person">
    <!-- The name of this rule attribute matches that of an
         attribute on the CDS entity type, and so its value will
         be automatically specified by the
         DataStoreRuleObjectRetriever. -->
```

```
   <Attribute name="firstName">
     <!-- The type of the rule attribute must agree with the
          type of the CDS attribute type, otherwise CER will
          issue a runtime error. -->
     <type>
       <javaclass name="String"/>
     </type>
     <derivation>
       <specified/>
     </derivation>
   </Attribute>

   <!-- NB no rule attribute for the CDS attribute for lastName.
-->

   <!-- Rule attributes matching the "childEntities_<rule class
name>"
        pattern will receive special treatment from the
        DataStoreRuleObjectRetriever.

        The DataStoreRuleObjectRetriever will specify the value of
        this attribute to be all the rule objects created from the
        child Income records which belong to this Person's record
        in the CDS. -->
   <Attribute name="childEntities_Income">
     <!-- The type must be a list of Income rule objects -->
     <type>
       <javaclass name="List">
         <ruleclass name="Income"/>
       </javaclass>
     </type>
     <derivation>
       <specified/>
     </derivation>
   </Attribute>
 </Class>

 <!-- The name of this rule class does not match any CDS entity
type,
      so the DataStoreRuleObjectRetriever will not create any rule
      objects for this rule class. -->
 <Class name="Benefit">
   <Attribute name="amount">
     <type>
       <javaclass name="Number"/>
     </type>
     <derivation>
       <specified/>
     </derivation>
   </Attribute>
 </Class>

 <Class name="Income">
   <!-- A rule attribute named "parentEntity" will receive
       special treatment from the
       DataStoreRuleObjectRetriever.

       The DataStoreRuleObjectRetriever will specify the value
       of this attribute to be the rule object created from the
       Person record which is the parent of this Income record
       in the CDS. -->
   <Attribute name="parentEntity">
     <!-- The type must be a single Person rule object -->
     <type>
       <ruleclass name="Person"/>
     </type>
     <derivation>
```

```
    <specified/>
  </derivation>
</Attribute>

<Attribute name="type">
  <type>
    <!-- The type of this attribute must specify the correct
         code table, matching the CDS domain definition. -->
    <codetableentry table="IncomeType"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="amount">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- This derivation will never be executed, as
         the DataStoreRuleObjectRetriever will automatically
         "specify" the value from that on the CDS record;
         once a value is specified CER does not attempt to
         calculate it.

         In general, attributes which expect to be populated
         by the DataStoreRuleObjectRetriever should be marked
         as <specified/> to avoid any confusion between
         stand-alone testing of your rule sets and testing
         with the DataStoreRuleObjectRetriever.
    -->
    <Number value="123"/>
  </derivation>
</Attribute>

<!-- This attribute is not present on the CDS entity type,
     so will not be populated.  This is exactly what we
     want, because its value is derived from other
     rule attributes in the normal CER way. -->
<Attribute name="isCountable">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <choose>
      <type>
        <javaclass name="Boolean"/>
      </type>
      <test>
        <reference attribute="type"/>
      </test>
      <when>
        <condition>
          <Code table="IncomeType">
            <String value="Full-time"/>
          </Code>
        </condition>
        <value>
          <true/>
        </value>
      </when>
      <when>
        <condition>
          <Code table="IncomeType">
            <String value="Part-time"/>
          </Code>
```

```
            </condition>
            <value>
              <true/>
            </value>
          </when>
          <otherwise>
            <value>
              <false/>
            </value>
          </otherwise>
        </choose>
      </derivation>
    </Attribute>

    <!-- This rule attribute is not calculated, nor
         does it correspond to an attribute on the
         CDS entity type.

         If the value of this attribute is referenced
         at runtime, CER will report a runtime error:
         "Value must be specified before it is used
          (it cannot be calculated)."
    -->
    <Attribute name="employerName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>
```

Sobald John sein Self-Screening abgeschlossen hat, lädt das Modul "Universal Access" das obige CER-Regelwerk und erstellt eine CER-Sitzung.

Das Modul "Universal Access" ruft die Funktion `DataStoreRuleObjectCreator` auf und gibt den Stammdatensatz im Datenspeicher (Application #1234) an.

Die Funktion `DataStoreRuleObjectCreator` ruft alle untergeordneten Datensätze von "Application #1234" ab und verarbeitet sie folgendermaßen:

- **Application #1234**

  Wird übersprungen, da es im Regelwerk keine Regelklasse namens "Application" gibt.
- **Person #1235**

  Es wird eine Regelobjektinstanz der Regelklasse `Person` mit den folgenden Elementen erstellt:
  - **firstName**

    Wird mit "John" angegeben.
  - **lastName**

    Wird übersprungen, da es kein Regelattribut namens "lastName" in der Regelklasse `Person` gibt.
  - **childEntities_Income**

    Wird als leere Liste angegeben, da es keine Datensätze des Typs "Income" für den Personendatensatz #1235 gibt.
- **Person #1236**

Es wird eine Regelobjektinstanz der Regelklasse `Person` mit den folgenden Elementen erstellt:

- **firstName**

  Wird mit "Mary" angegeben.

- **lastName**

  Wird übersprungen.

- **childEntities_Income**

  Wird als Liste mit den beiden (nachfolgend erstellten) Regelobjekten des Typs "Income" für Mary angegeben.

- **Income #1237**

  Es wird eine Regelobjektinstanz der Regelklasse `Income` mit den folgenden Elementen erstellt:

  - **parentEntity**

    Wird als (zuvor erstelltes) Regelobjekt des Typs `Person` für Mary aus dem Personendatensatz #1236 angegeben.

  - **type**

    Wird mit dem Code "Part-time" angegeben.

  - **amount**

    Wird mit der Zahl "25" angegeben.

  - **(employerName)**

    Wird nicht angegeben, da die Datenspeicherentität kein Attribut namens "employerName" enthält.

- **Income #1238**

  Es wird eine Regelobjektinstanz der Regelklasse `Income` mit den folgenden Elementen erstellt:

  - **parentEntity**

    Wird als (zuvor erstelltes) Regelobjekt des Typs `Person` für Mary aus dem Personendatensatz #1236 angegeben.

  - **type**

    Wird mit dem Code "Part-time" angegeben.

  - **amount**

    Wird mit der Zahl "30" angegeben.

Abschließend stellt das Modul "Universal Access" (Universalzugriff) dem Regelwerk die Standardfragen über Programme, Anspruchsberechtigung und Erläuterung. Die (im obigen Beispiel nicht dargestellten) Regelklassen für die Programme greifen bei der Beantwortung dieser Fragen auf die (von der Funktion `DataStoreRuleObjectCreator` erstellten) Regelobjekte zu.

Weitere Informationen zum Anwendungsdatenspeicher und seinen Schemas enthält das Handbuch `Creating Datastore Schemas`.

Konformitätsinformationen zu CER

Eine Beschreibung, wie Sie bei der Entwicklung von CER-Regeln die Konformität einhalten. Wenn Sie die hier aufgeführten Hinweise befolgen, wird ein Upgrade auf künftige Versionen erleichtert.

Öffentliche API von CER

Die CER-Infrastruktur ist mit einer öffentlichen API ausgestattet, die Sie in Regelwerktests und im Anwendungscode aufrufen können. Die in diesem Cookbook aufgeführten Beispiele zeigen, wie viele Funktionen der öffentlichen API in der CER-Infrastruktur verwendet werden. Die Anwendung ändert oder entfernt keine Elemente in dieser öffentlichen API ohne Berücksichtigung der Standards für die Handhabung von Auswirkungen für Kunden.

Sofern dies im JavaDoc nicht ausdrücklich als zulässig angegeben ist, *dürfen Sie keine* eigene Implementierung von CER-Java-Schnittstellen bereitstellen oder für eine Java-Klasse der CER-Implementierung eine Unterklasse erstellen.

Informationsquelle für API

Das in CER enthaltene JavaDoc ist die einzige Informationsquelle dafür, welche öffentlichen Klassen, Schnittstellen und Methoden die öffentliche API von CER bilden.

API-externe Elemente

Die CER-Infrastruktur enthält auch einige öffentliche Klassen, Schnittstellen und Methoden, die *nicht* Bestandteil der API sind.

Wichtig: Zur Einhaltung der Konformität *dürfen Sie keine* Abhängigkeit von einer CER-Klasse oder -Schnittstelle erstellen noch andere als die im JavaDoc beschriebenen Methoden aufrufen.

CER-Klassen, -Schnittstellen und -Methoden außerhalb der öffentichen API können jederzeit ohne Ankündigung geändert oder entfernt werden.

Sofern dies im JavaDoc nicht ausdrücklich als zulässig angegeben ist, *dürfen Sie keine* eigenen Klassen oder Schnittstellen in das Paket `curam.creole` oder in eines seiner Unterpakete aufnehmen.

CER-Ausdrücke

Es werden ausschließlich die im vorliegenden Cookbook aufgelisteten Ausdrücke unterstützt. Eine Liste der unterstützten Ausdrücke finden Sie im Abschnitt „Vollständige alphabetische Liste der Ausdrücke" auf Seite 171.

Bitte beachten Sie, dass das in CER enthaltene Regelwerkschema auch einige nicht unterstützte Ausdrücke enthält. Diese Ausdrücke sind experimenteller Natur und können jederzeit ohne Ankündigung geändert oder entfernt werden. Verwenden Sie in Ihren Regelwerken keine nicht unterstützten CER-Ausdrücke.

In der Anwendung enthaltene Regelwerke

Einige Komponenten in der Anwendung enthalten möglicherweise CER-Regelwerke, für die gesonderte Konformitätsanforderungen gelten. Diese Konformitätsanforderungen von CER-Regelwerken, die in den Anwendungskomponenten enthalten sind, werden im vorliegenden Dokument nicht behandelt.

Informationen dazu, ob die Anpassung des CER-Regelwerks gegebenenfalls zulässig ist und welche Einschränkungen in diesem Fall für die Anpassung gelten, enthält die Dokumentation, die mit diesen Komponenten bereitgestellt wird.

Das in CER enthaltene Stammregelwerk (RootRuleSet) darf von Kunden nicht geändert werden.

Beispiele in diesem Handbuch

Die Beispielartefakte im vorliegenden Handbuch (Regelwerke, Java-Code und Eigenschaftendateien) können jederzeit ohne Ankündigung geändert oder entfernt werden.

Sie können diese Beispielartefakte natürlich für den Eigengebrauch *kopieren*. Bitte beachten Sie jedoch, dass für diese Beispielartefakte keine Upgradeunterstützung besteht.

CER-Datenbanktabellen

Die CER-Infrastruktur beinhaltet eine Reihe von Datenbanktabellen. Diese Tabellen sind im Allgemeinen interne Tabellen von CER und die in ihnen enthaltenen Daten können nur über die öffentliche API von CER gelesen oder geschrieben werden.

Weitere Details über die Einschränkungen bei Lese- und Schreibvorgängen für diese Datenbanktabellen enthalten die folgenden Unterabschnitte.

Anmerkung: Allen Namen von Datenbanktabellen der CER-Infrastruktur ist das Präfix CREOLE vorangestellt. Der Umkehrschluss gilt jedoch nicht.

Es gibt Tabellen mit dem Präfix CREOLE, die Bestandteil anderer Anwendungskomponenten sind und eigenen Konformitätsanforderungen unterliegen. Nachfolgend sind ausschließlich die Konformitätsanforderungen für Datenbanktabellen der *CER-Infrastruktur* beschrieben.

CREOLERuleSet

In dieser Datenbanktabelle wird für jedes im System veröffentlichte CER-Regelwerk eine Zeile gespeichert.

Der Lese- und Schreibzugriff auf diese Datenbanktabelle ist im Grunde genommen ausschließlich auf die öffentliche API von CER beschränkt. Sie können jedoch den Datenmanager der Anwendung verwenden, um diese Datenbanktabelle zu füllen, sofern die folgenden Bedingungen erfüllt sind:

- Der Wert der Spalte `name` muss mit dem Regelwerknamen übereinstimmen, der im XML für die Spalte `ruleSetDefinition` definiert ist.
- Der Wert der Spalte `ruleSetVersion` muss `null` lauten.

Nachfolgend finden Sie einen Beispieleintrag für die konforme Datei `CREOLERuleSet.dmx`.

```
<?xml version="1.0" encoding="UTF-8"?>
<table name="CREOLERULESET">
<column name="creoleRuleSetID" type="id"/>
<column name="name" type="text"/>
<column name="ruleSetDefinition" type="blob"/>
<column name="versionNo" type="number"/>
<column name="ruleSetVersion" type="number"/>
<row> ... </row>
<row>
<attribute name="creoleRuleSetID">
<!-- Entsprechende eindeutige Kennung verwenden -->
<value>99999</value>
</attribute>
<attribute name="name">
<!-- Dieser Name muss dem Wert <RuleSet name="..."> 
im Regelwerk-XML-Code in 'ruleSetDefinition' weiter unten
entsprechen -->
<value>MyRuleSet</value>
```

```
</attribute>
<attribute name="ruleSetDefinition">
<value>./path/to/MyRuleSet.xml</value>
</attribute>
<attribute name="ruleSetVersion">
<!-- Muss ein Element <value/> sein, dass den Datenbankwert
NULL bezeichnet -->
<value/>
</attribute>
<attribute name="versionNo">
<!-- Anfangswert 'versionNo' für optimistisches Sperren -->
<value>1</value>
</attribute>
</row>
<row> ... </row>
</table>
```

Im Handbuch Working with Cúram Express Rules sind die Schritte beschrieben, mit denen die Daten für die Tabelle "CREOLERuleSet" aus der Anwendung (und sofern erforderlich die zugehörigen Daten der Anwendungsressource) extrahiert werden können.

CREOLEMigrationControl

Die Steuertabelle "CREOLEMigrationControl" besteht aus einer einzigen Zeile und dient dazu, eine gleichzeitige Veröffentlichung von CER-Regelwerken zu verhindern.

Die Daten in dieser Tabelle sind CER-interne Daten und können durch eine andere Komponente weder gelesen noch geschrieben werden.

Die einzige Zeile dieser Tabelle wird mittels einer DMX-Datei gefüllt, die in der Anwendung enthalten ist. Kunden dürfen weder diese DMX-Datei ändern noch andere DMX-Dateien erstellen, deren Ziel die Tabelle "CREOLEMigrationControl" ist.

Weitere Datenbanktabellen der CER-Infrastruktur

In der CER-Infrastruktur sind des Weiteren die folgenden Datenbanktabellen enthalten:

- CREOLEAttributeAvailability
- CREOLEAttributeInheritance
- CREOLERuleAttribute
- CREOLERuleAttributeValue
- CREOLERuleClass
- CREOLERuleClassInheritance
- CREOLERuleObject
- CREOLERuleSetDependency
- CREOLERuleSetEditAction
- CREOLERuleSetSnapshot
- CREOLEValueOverflow

Auch für diese Datenbanktabellen der CER-Infrastruktur gelten die vorliegenden allgemeinen Konformitätsbestimmungen: Die Daten in diesen Datenbanktabellen dürfen ausschließlich mittels der öffentlichen API von CER gelesen oder geschrieben werden. Insbesondere wird das erstmalige Füllen dieser Datenbanktabellen über DMX-Dateien *nicht* unterstützt.

Abhängigkeitsmanager

Der Abhängigkeitsmanager ist eine interne Cúram-Komponente. Ein Zugriff aus benutzerdefiniertem Code heraus oder eine etwaige Anpassung wird nicht unterstützt.

Alle den Abhängigkeitsmanager betreffenden Artefakte sind im Codepaket `curam.dependency` und seinen Unterpaketen enthalten. Manche Artefakte in diesem Codepaket werden von CER beigesteuert, andere Pakete sind Beiträge der Basisanwendung. Sie dürfen keine eigenen Klassen oder Schnittstellen in das Codepaket `curam.dependency` oder in eines seiner Unterpakete aufnehmen.

Der Abhängigkeitsmanager ist Eigner der folgenden Datenbanktabellen:

- Dependency
- PrecedentChangeSet
- PrecedentChangeItem
- PrecedentChangeSetBatchCtrl

Diese Datenbanktabellen dürfen keinesfalls angepasst werden. Die Daten in diesen Datenbanktabellen dürfen ausschließlich durch den Abhängigkeitsmanager selbst gelesen oder geschrieben werden.

Das erstmalige Füllen dieser Datenbanktabellen über DMX-Dateien wird (mit Ausnahme von in der Anwendung enthaltenen DMX-Dateien) *nicht* unterstützt. Außerdem kann das Auffüllen der Tabellen mithilfe der DMX-Dateien aus der Anwendung nicht angepasst oder übergangen werden.

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

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. IBM kann daher die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programm nicht garantieren oder implizieren. Die Beispielprogramme werden ohne Wartung (auf "as-is"-Basis) und ohne jegliche Gewährleistung zur Verfügung gestellt. 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.

© Copyright IBM Corp. _Jahreszahl oder Jahreszahlen eingeben_. Alle Rechte vorbehalten.

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

Hinweise zur Datenschutzrichtlinie

IBM Softwareprodukte, einschließlich Software as a Service-Lösungen ("Softwareangebote"), können Cookies oder andere Technologien verwenden, um Informationen zur Produktnutzung zu erfassen, die Endbenutzererfahrung zu verbessern und Interaktionen mit dem Endbenutzer anzupassen oder zu anderen Zwecken. In vielen Fällen werden von den Softwareangeboten keine personenbezogenen Daten erfasst. Einige der IBM Softwareangebote können Sie jedoch bei der Erfassung personenbezogener Daten unterstützen. Wenn dieses Softwareangebot Cookies zur Erfassung personenbezogener Daten verwendet, sind nachfolgend nähere Informationen über die Verwendung von Cookies durch dieses Angebot zu finden.

Je nachdem, welche Konfigurationen implementiert wurden, ist es möglich, dass dieses Softwareangebot Sitzungscookies und persistente Cookies zum Erfassen der Namen, Benutzernamen, Kennwörter, Profilnamen oder anderer personenbezogener Daten einzelner Benutzer für die Sitzungsverwaltung, Authentifizierung, Single-Sign-on-Konfiguration oder für einen besseren Bedienungskomfort und/oder andere Zwecke der Nutzungsverfolgung bzw. funktionale Einsatzmöglichkeiten. Diese Cookies oder ähnliche Technologien können nicht inaktiviert werden.

Wenn die für dieses Softwareangebot genutzten Konfigurationen Sie als Kunde in die Lage versetzen, personenbezogene Daten von Endbenutzern über Cookies und andere Technologien zu erfassen, müssen Sie sich zu allen gesetzlichen Bestimmungen in Bezug auf eine solche Datenerfassung, einschließlich aller Mitteilungspflichten und Zustimmungsanforderungen, rechtlich beraten lassen.

Weitere Informationen zur Nutzung verschiedener Technologien, einschließlich Cookies, für diese Zwecke finden Sie in der "IBM Online-Datenschutzerklärung, Schwerpunkte" unter http://www.ibm.com/privacy und in der "IBM Online-Da-

tenschutzerklärung" unter http://www.ibm.com/privacy/details im Abschnitt "Cookies, Web-Beacons und sonstige Technologien" und unter "IBM Software Products and Software-as-a-Service Privacy Privacy Statement" unter http://www.ibm.com/software/info/product-privacy.

Marken

IBM, das IBM Logo und ibm.com sind eingetragene Marken der International Business Machines Corporation in den USA und/oder anderen Ländern. 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.

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

Gedruckt in Deutschland