



Cúram 8.2.2

Creating Datastore Schemas Guide

Note

Before using this information and the product it supports, read the information in [Notices on page 17](#)

Edition

This edition applies to Cúram 8.2.2.

© Merative US L.P. 2012, 2026

Merative and the Merative Logo are trademarks of Merative US L.P. in the United States and other countries.

Contents

Note.....	iii
Edition.....	v
1 Creating Datastore Schemas.....	9
1.1 Important Information Before Creating Datastore Schemas.....	9
The Purpose of Datastores.....	9
How to Properly Access Datastores.....	9
1.2 Creating a Datastore Schema.....	9
Datastore Schema Overview.....	10
Namespaces.....	10
Datatypes.....	10
Entity and Attribute Definitions.....	12
Identity Constraints.....	13
1.3 Compliancy for datastore schemas.....	14
Public API.....	15
Identifying the API.....	15
Outside the API.....	15
Notices.....	17
Privacy policy.....	18
Trademarks.....	18

1 Creating Datastore Schemas

Use this information to create Datastore schemas. The datastore stores data that is collected from citizens during intake. A datastore schema defines the entities across which intake data is shredded, the fields or attributes of those entities, and any relationships between entity definitions.

1.1 Important Information Before Creating Datastore Schemas

The Purpose of Datastores

The Datastore stores data that is collected from citizens during online screening and intake of applications. The contents of the datastore are dynamically definable by way of a datastore schema definition. Other dynamically definable components depend on the Datastore Schema, in particular screening and intake scripts, and screening rules, so the datastore contents must be kept compatible with these. In particular, these other components might depend on certain fixed elements of Datastore definitions, which must therefore exist in order for these components to operate properly.

Data on the Datastore is stored in XML format. For reasons of efficiency, the data for a single application is shredded across multiple database rows so that updates to an application during intake do not result in large amounts of inefficient LOB I/O every time a page of an intake script is traversed. Each type of shredded row content is called a datastore *entity*. The datastore schema definition conforms to the World Wide Web Consortium (W3C) XML Schema Definition Language. In addition to having to conform to the W3C specification, a datastore schema must follow certain datastore-specific rules. These rules are described in [1.2 Creating a Datastore Schema on page 9](#).

How to Properly Access Datastores

The datastore is accessed through an API defined in the Java® package `curam.datastore.impl`. This package allows for opening a datastore by locating its schema definition, and accessing its data.

For further details, see the Javadoc information for the API package. Note that other `curam.datastore.*` packages do not form part of the API (even where classes and methods are public) and are not to be used.

1.2 Creating a Datastore Schema

Datastore Schema Overview

A datastore schema defines the following:

- The entities across which screening and intake data are shredded - These are created as XML *complexType* definitions.
- The fields or *attributes* of those entities - These are created as XML *attribute* definitions.
- The simple datatypes, called *domains*, which define the allowable types of entity attributes - These are created as XML *simpleType* definitions.
- Any key relationships between entity definitions - These are created as XML *key* and *keyref* definitions.

Namespaces

A datastore schema must be defined in a single XML namespace, or in no namespace. It cannot be split across multiple namespaces.

The schema *may* be split across multiple schema fragments that include each other. Schema definitions contain no XML `import` elements other than importing the base domains namespace **http://www.curamsoftware.com/BaseDomains**, but they might contain `include` elements.

Datatypes

The attributes in a datastore schema definition cannot directly be of W3C XML built-in datatypes. Instead, they must be valid Cúram *domain definitions*.

This means that they must be of a simple type that is defined in the XML namespace **http://www.curamsoftware.com/BaseDomains**, or a simple type that is defined in the datastore schema, which inherits from one of those types. In the example above, DECEASED_IND is such a domain definition. The built-in domain definitions in **http://www.curamsoftware.com/BaseDomains** are:

Table 1: Built-in Domain Definitions

Base Domain	Built-in XML Type	Remarks
SVR_BOOLEAN	Boolean	
SVR_INT8	Byte	
SVR_INT16	Short	
SVR_INT32	Int	
SVR_INT64	Long	
SVR_KEY	Long	
SVR_FLOAT	Float	
SVR_DOUBLE	Double	
SVR_DATE	Date	
SVR_DATETIME	dateTime	
SVR_MONEY	Decimal	fractionDigits value="2"
SVR_STRING	String	

Base Domain	Built-in XML Type	Remarks
SVR_CHAR	String	minLength value="1", maxLength value="1"
CODETABLE_CODE	String	minLength value="1", maxLength value="10"
SHORT_CODETABLE_CODE	String	minLength value="1", maxLength value="10"

When defining new domains, their names cannot conflict with domains that are modeled in the application as they might potentially be displayed with the wrong renderer.

Domain definitions support a limited number of value constraints. The following XML constraints are permitted:

- minLength
- maxLength
- minInclusive
- maxInclusive

In addition to the supported XML constraints, there are a number of Cúram-specific constraints that can be specified by using XML annotations, as follows:

```
<xsd:simpleType name="SOME_DOMAIN">
  <xsd:annotation>
    <xsd:appinfo>
      <d:options>
        <d:option name="min-size">3</d:option>
      </d:options>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:restriction base="d:SVR_STRING" />
</xsd:simpleType>
```

The list of supported annotation-based constraints is as follows:

Table 2: Supported Annotation-based Constraints

Annotation Option	Remark
min-size	Same as XML minLength
max-size	Same as XML maxLength
minimum	Same as XML minInclusive
maximum	Same as XML maxInclusive
default	Default value
compress	Reduce embedded runs of spaces to a single space
to-upper	Convert text to uppercase
trim-leading	Trim leading space
trim-trailing	Trim trailing spaces
code-table-name	Name of a Cúram code table that is used to translate code value
code-table-root	Do not use this option in user domain definitions.

Note that domain constraints have no effect on processing in the datastore when the datastore is accessed programmatically. They exist purely to provide metadata to screen processing facilities such as Cúram Intelligent Evidence Gathering(IEG), which can use the metadata to constrain user input.

Entity and Attribute Definitions

Entities in the Datastore are defined as elements with *complexType* definitions. Each entity's definition must appear at global level in the schema. If an entity incorporates another entity as part of its definition, the incorporated entity must still be defined at global level and *referred* to by the incorporating entity.

In the example below, the `Application` entity incorporates the `Person` entity, which in turn incorporates the `Address` entity. All three entities are defined at global level and use *element ref* to refer to the definitions of other entities.

```
<xs:element name="Application">
  <xs:complexType>
    <xs:sequence minOccurs="0">
      <xs:element ref="Person" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Person">
  <xs:complexType>
    <xs:sequence minOccurs="0">
      <xs:element ref="Address"
        minOccurs="0"
        maxOccurs="1" />
      ...
    </xs:sequence>
    <xs:attribute name="firstName" type="D:SVR_STRING" />
    ...
  </xs:complexType>
</xs:element>

<xs:element name="Address">
  ...
</xs:element>
```

There are other restrictions on entities. All complex type compositors must be sequences, that is, you can use the “sequence” compositor but not “all” or “choice”. All sequences must have a “minOccurs” of zero, and all elements must have a “minOccurs” of zero. There is no restriction on the value of the “maxOccurs” constraint.

Attributes of entities must be of “domain” types, that is, they must be *simpleType* definitions that are defined elsewhere in the schema. An attribute can have a default value, which (unlike the “default” annotation on domains) *does* directly affect the values that get stored on the Datastore if the attribute value is not explicitly set.

The datastore definition usually consists of many *simpleType* (domain) and *complexType* (entity) definitions. All of these must appear at global level, that is, they cannot be nested inside each other.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:D="http://www.curamsoftware.com/BaseDomains"
>

  <xs:import
    namespace="http://www.curamsoftware.com/BaseDomains" />

  <xs:simpleType name="DECEASED_IND">
    <xs:restriction base="D:SVR_BOOLEAN"/>
  </xs:simpleType>

  <xs:element name="Person">
    <xs:complexType>
      <xs:attribute name="firstName" type="D:SVR_STRING" />
      <xs:attribute name="middleInitial"
        type="D:SVR_STRING" />
      <xs:attribute name="lastName" type="D:SVR_STRING" />
      <xs:attribute name="deceased" type="DECEASED_IND" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Identity Constraints

Consider the example that is shown here. It models a Person that can have an Address, and zero or more Relationships to other Persons. A Relationship does not “contain” a Person. Rather, it has a reference to a Person.

How do you ensure that one entity uniquely refers to another entity? You use XML identity constraints. These constraints are well-described in the relevant XML specifications and no elaborations are required, other than to point out more restrictions on the use of identity constraints in a Datastore Schema.

```
<xs:element name="Person">
  <xs:complexType>
    <xs:sequence minOccurs="0">
      <xs:element ref="Address"
        minOccurs="0"
        maxOccurs="1" />
      <xs:element ref="Relationship" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="personID"
      type="D:SVR_KEY"/>
    <xs:attribute name="firstName"
      type="D:SVR_STRING" />
    <xs:attribute name="middleInitial"
      type="D:SVR_STRING" />
    <xs:attribute name="lastName" type="D:SVR_STRING" />
```

```

</xs:complexType>
<xs:key name="PersonKey">
  <xs:selector xpath="./Person"/>
  <xs:field xpath="@personID"/>
</xs:key>
<xs:keyref name="RelationshipRef" refer="PersonKey">
  <xs:selector xpath="./Person/Relationship"/>
  <xs:field xpath="@personID"/>
</xs:keyref>
</xs:element>
<xs:element name="Address">
  <xs:complexType>
    <xs:attribute name="street1" type="D:SVR_STRING" />
    <xs:attribute name="street2" type="D:SVR_STRING" />
    <xs:attribute name="city" type="D:SVR_STRING" />
    <xs:attribute name="state" type="D:SVR_STRING" />
    <xs:attribute name="zipCode" type="D:SVR_INT32" />
  </xs:complexType>
</xs:element>
<xs:element name="Relationship">
  <xs:complexType>
    <xs:attribute name="type" type="D:SVR_STRING" />
    <xs:attribute name="personID" type="D:SVR_KEY" />
  </xs:complexType>
</xs:element>

```

An attribute referred to in a *key* or *keyref* identity constraint must be defined to be of the base domain type SVR_KEY. Conversely, every attribute of type SVR_KEY must be referred to by some *key*/ or *keyref* definition. Key attributes have their values that are automatically populated upon insertion into the Datastore. They are set to a numeric value that is unique across all entities on the Datastore.

1.3 Compliancy for datastore schemas

This section explains how to develop datastore schemas in a compliant manner. By following these considerations, customers will also find it easier to upgrade to future versions of Cúram.

Public API

The Datastore has a public API which you may use in your application code. This public API does not have any components that are changed or removed without following Cúram standards for handling customer impact.

Identifying the API

The Javadoc that is included is the sole means of identifying which public classes, interfaces, and methods form the public API.

Outside the API

The Datastore also contains some public classes, interfaces, and methods, which do not form part of the API.

Important: To be compliant, dependencies on any class or interface cannot be made. No methods are called other than those described in the Javadoc information.

Classes, interfaces, and methods outside of the public API are subject to change or removal without notice. Unless otherwise stated in the Javadoc, you must not place any of your own classes or interfaces in the same package as the Datastore.

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

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

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those

websites are not part of the materials for this Merative product and use of those websites is at your own risk.

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

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

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

Privacy policy

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

Trademarks

Merative™ and the Merative™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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

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

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