Access 4 Learning Community

Powered by: SIF

Simple. Secure. Scalable. Standard.

SIF Data Model v3 User's Guide

www.A4L.org

Version 3, August 2015

Table of Contents

Introduction

Purpose of this Document

This document is meant to provide a quick start to the 3.x data models for both experienced SIF users and new SIF users. For experienced users, information concerning what is new and what is different from previous major releases of the Data Model will be provided. For new users, important concepts needed to understand and use the Data Model will be included in this document.

Other documents will provide different and more detailed information about the Data Model and how to use the model. For example, specific information about how to use a particular SIF Object will be provided in separate object usage documents.

This document will continue to grow as frequently asked questions or important topics are identified. So, check back to the website for updated versions.

What's New in Data Model Version 3

Some major new things in SIF 3 data models include:

- Extension Points – A new way of extending SIF objects has been implemented.
- Annotations – More documentation of SIF objects and elements is included as XML annotations. All definition information, excluding usage and guides, will be delivered with the schema in the form of annotations.
- Global Data Model – The SIF 3 data models now include conceptual and logical components that will make data models more consistent from one locale to another. Locale models will be built upon a global conceptual/logical model. However, users of SIF data models only have to deal with their locale model in practice.
- Object Types – SIF 3 data models will now offer objects in three categories: Entity Objects, Composite Objects, and Report Objects. Each category contains objects that are used in a similar manner. This will help clarify which object to use when more than one object contains the same content and will help implementers maintain master data management.

What's Different in Data Model Version 3

Here are some things that have changed from version 2 to version 3 data models:

- Separation from Infrastructure – The infrastructure and data model no longer have dependencies. In theory, any data model can be transmitted on the SIF 3 infrastructure, and the SIF 3 data models can be used in situations other than on the SIF transport.
- Lists – Lists have been standardized in form, cardinality, and naming.
- Code Sets – A new approach to code sets removes them from the schema documents and makes them easier to revise and extend.

SIF Versions

The SIF Implementation Specification uses the following version numbering scheme:

<*major version*> . <*minor version*> (r)(*revision number*)

Major versions typically introduce additions/changes to the SIF infrastructure and/or data model changes that impact a significant percentage of SIF-enabled applications (e.g. making previously optional elements mandatory, removal of deprecated objects, elements or values). The first release of a major version has a minor version of 0 (2.0); major version numbers start at 1 and are incremented as major versions are released (1.0, 2.0, 3.0 …).

Minor releases typically introduce new data objects, or optional additions to data objects, to the marketplace, and may include minor infrastructure additions/changes that do not impact existing SIF-enabled applications and that vendors have agreed to implement. The first minor version released subsequent to and within a major release has a minor version of 1 and is incremented as new minor versions are released (2.1, 2.2, …). If a significant number of minor release features is introduced in a specification, the SIF Association may decide to increment the minor version number by more than 1 (e.g. 1.1 to 1.5), though a number like 1.5 is not an indication of being halfway to a major release: minor version numbers may be incremented significantly past 10 (2.10, 2.11, …) as data objects and other minor version features are released.

Corrections resulting from identified errata, as well as textual changes, may be incorporated into a revision release. These typically include minor corrections to messages or data objects, corrections of typographical errors, or corrected/expanded documentation. If major errors in any release are identified, a revision release may incorporate changes more typical of a major or minor release. First major and minor releases have a revision number of 0, which is omitted from the version number (2.0, not 2.0r0); subsequent revision numbers start at 1 and are incremented as new revisions are released (2.0r1, 2.0r2, …).

Design Patterns and Object Types

First some terms:

Data Structure - This is a general term for a set of XML elements that are part of the same physical package.

SIF Object - An arbitrarily defined data structure. In SIF, an object is a root-level complex XML type. However, not all complex types are SIF objects. The SIF object is a convenient concept that does not exist in XML but does exist in modelling paradigms.

Entity Object - An Entity object represents a distinct concept taken from the logical and conceptual SIF models. The design of these objects is model-driven, not use-case-driven, so they may need to be combined to fit a particular use case. Examples of Entity objects include student, teacher, school, and student-school-enrollment.

Composite Object - A Composite object is an object designed for a particular use case or limited set of use cases. The object is usually made up of the combined parts of Entity objects. These objects are part of the SIF Physical Model but are not part of the Entity model (sub-model). An example of a Composite object is the StudentPersonal object from the US 2.6 specification. This object contains student information from the Student object in the Entity model as well as information from other objects, such as StudentEnrollment. Composite objects may be updatable.

Report Object - A Report object is a subclass of Composite objects, designed to represent point-in-time information and can contain summary information, cross tabulations, or information associated with a range of entities. This is the same kind of information as would be contained in a typical report. Report objects are read-only.

Previous versions of SIF have developed objects primarily in response to use cases. This approach has addressed the use case of generating reports well, as reports can combine information from various sources. However, it has presented two major problems:

- It has made it difficult to model and update SIF objects: objects are un-normalized, with all the known problems that un-normalized objects present in data management.
- It has made it difficult to synchronise updates to those objects, particularly if they are to be written back to their original sources of truth. A StudentPersonal record, for example, might draw the email address, the ELL status, and the MostRecent/GradeLevel of the student from three different systems—one for Student data, one for programs, and one for enrollments. If all three are updated, the broker needs to know which system to update with which information. In the case of direct REST connections, this

kind of negotiation of content is impossible: we are updating only a single system, and we need objects that can cleanly be updated on one system.

SIF 3 needs to continue to address the use-case driven requirements; but SIF 3.0 has also decided that it needs a more manageable and cleanly defined data model, to enable clearer relationships between locales, more internal consistency in data definitions, and more straightforward update pathways for direct connections. For that reason, SIF 3 defines a normalized entity model, including all the distinct concepts in education that we need to communicate about. You can think of entity objects as SIF database tables, with all the good practice around avoiding redundancy that comes with database tables. (You can also think of the entity model as classes in a class hierarchy—which is in fact exactly how they are modelled in the Data Model. The same kinds of concern about normalization and redundancy apply to classes.) Because they are highly normalized, and have to fit with each other, entity objects are defined by SIF data architects.

The use-case-driven objects can still be supported by SIF 3, but they are modelled separately, as Composite objects. Composite objects are explicitly based on entity objects; each element in a composite object should be traceable back to either an entity object element, or a calculated value, based on one or more values (including but not limited to entity object elements). You can think of composite objects as SIF database views.

Reports, which are a subset of composite objects, are read-only, point-in-time descriptions, and rely heavily on information aggregated out of individual entity objects. But a composite object can also combine entity objects without summarising them; StudentPersonal also counts as a composite object, combining Student and StudentEnrollment. Provided that the source each Composite Object field is traced and the data dependencies work out, you should be able to update a Composite object—which makes it correspond to an Updatable database view. And unlike previous versions of SIF, the dependence of the Composite object on its underlying data is now made explicit, making it much easier to perform such updates.

The following table summarizes the differences between Entity objects and Composite/Report objects:

Entity Model	Composite/Report Model
Model-Driven Design	Use-Case-Driven Design
Queryable using a flexible language	Designed for a narrow set of, or one, specific query (e.g., roster report).

Flexible so as to allow previously unknown questions to be asked.	Easy to use and is extremely well defined. Answers a small set or a single question.
Trades some efficiency for power.	Lightweight and efficient
Normalized with interrelated objects.	Generally flat files with simple structures.
Harder to implement but ROI is high.	Easier to implement but ROI is limited.
Flexible	Concrete
Long-Term Interoperability	Short-Term Interoperability
SIF Data Architects make proposals for new objects. Project Teams make proposals for new elements.	Project Teams make proposals for objects and elements. Most objects and elements will come from the Object Model unless they are aggregate statistics.

Extending SIF Objects

Rationale

There is an increasing need to be able to extend the SIF Specification, to add elements that have not yet been vetted by the standard and to also extend the specification for specific locations such as states, provinces, and local authorities.

In the past, SIF allowed the extension of its specification by the introduction of SIF_ExtendedElements tag, which currently exists at the end of every object. However, this mode of extension has the following shortcomings:

- SIF_ExtendedElements/SIF_ExtendedElement structure required this artificial wrapper around the extension. It required a unique attribute "Name" to make it unique.
- While each SIF_ExtendedElement allowed the content to be XML and therefore also allowed for XSD validation, its most common use was as Name/Value pairs. These name/value pairs tended to be added without the proper thought for structure and naming, which made it more difficult to promote to be standard elements in the future of the core spec.
- Extensions outside this tag were not allowed, so they could only be found at the root at the end of the object. This was by far the biggest drawback of this extension, as profiles were forced to use complex naming of the Name/Value pair to attempt to specify where the extension belongs within the object
- The Name attribute was forced to be unique. However, there was nowhere in the specification where the name definitions could be found, therefore collisions were likely as more extensions were made.
- Because the specification did not allow for addition of attributes in elements, it became very hard to map an extension to a deep element unless the name/value pair was some form of XPath that could hint to the structure.
- The use of namespaces was suggested but could not be enforced in the specification for these extensions.
- SIF_ExtendedElement could be abused as a back-door to the specification where everything could be added
- The specification re-duplicated the definition of SIF_ExtendedElements in each object rather than using a typed structure and using ref="" in the element.

- There was no standard way provided of finding where such "standard" extensions were defined.

Goals of the Design

The current design satisfies the following main goals:

- Keep the interface simple so that SIF providers and consumers can easily implement extensions. We do not want the consumer to have to connect these extensions through references etc., as this introduces unnecessary complexity.
- Support the clean extension of the specification, and allow extensions to happen anywhere in the object (subject to the constraints below).
- Enforce the use of namespaces to identify the extensions
- Support the extensions in a standard manner and using the existing XML language to describe the extensions
- Promote the proper definition of extensions; some extensions may eventually become part of the core specification, while others will always remain local to a region/locale. Even so-called local extensions, such as extensions for a specific state, should aim to become standard eventually within that region. Having a core SIF specification plus the addition of a specification that is standard within a whole state is by far preferable over extensions that become proprietary to an instance or specific implementation.
- Provide a standard way of finding where such "standard" extensions are defined.

Extension points exist at the root of every SIF object. In addition, an extension point will be present in most complex elements except for lists.

Extending SIF Schemas in the 3.x Architecture

Not every single element in SIF should be extensible: patterns can be identified to restrict where extensions are allowed. Wherever we want to allow an extension within those patterns, we add the following in the specification:

```
<xs:any processContents="lax"
          namespace="##other"
          minOccurs="0"
          maxOccurs="unbounded">
</xs:any>
```


Of the possible processContext attribute values, processContents="lax" instructs any XML parser to ignore tags for which it does not have a definition. However, if the XSD is included for the namespace attribute, then the XML parser can validate the contents of the XML and enforce the rules defined in that namespace XSD. processContents="strict" instructs the XML parser to require the definition of an XSD. processContents="skip" ignores the tags for which it does not have a definition. **processContents="lax" is currently used for SIF extensions.**

Namespace="##other" instructs the parser that any extension of the object, elements, and attributes MUST belong to a different namespace. In other words these extensions could not be part of the default namespace (usually sif:). This solution not only enforces that the extensions cannot belong to the core model, but it also enforces that the extensions MUST belong to a namespace. This provides uniqueness and greatly reduces the chances of conflict between extension elements.

minOccurs="0" instructs the parser that it is acceptable if no extensions are found.

maxOccurs="unbounded" allows the inclusion of any number of namespace:elements.

Extensions under this convention may be added in any order, and require no artificial tag to surround them.

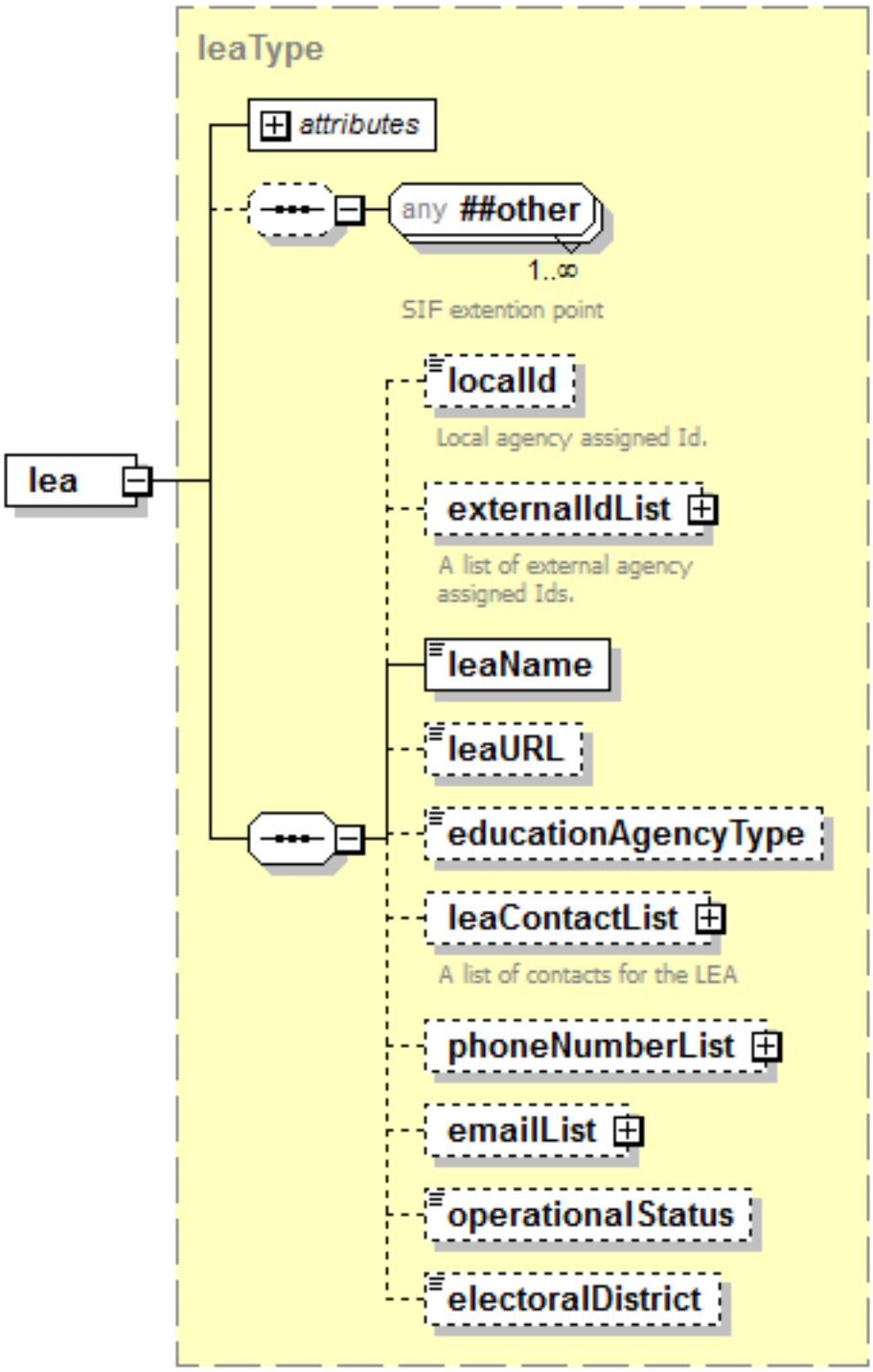
Extension Points & Privacy

Our use of "lax" processing encourages the use of objects with extensions and allows those needing validation to add it. However if you need to ensure private information is not being passed in an extended element that you are not aware of, some tweaking of the schemas is in order. By changing either all extension points or the ones you are concerned with from "lax" processing to "strict" before you validate, an error will be generated if any unknown elements are attempted to be passed. Only if you need this level of privacy, are you encouraged to make these necessary changes.

Example

Following is an example of how an extension is implemented. An LEA (Local Education Authority) SIF object is shown in unextended and extended forms. A schema is shown for the extended XML content. The example also shows how to specify the schema for the extended XML content so that it is validated along with the content that is part of the SIF schema.

LEA Object Schema in Diagram Form



Example XML for the LEA Object

```
<?xml version="1.0" encoding="UTF-8"?>
<lea refId="00000000-0000-4000-0000-000000000000"
      xmlns="http://www.sifassociation.org/datamodel/us/3.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.sifassociation.org/datamodel/us/3.0
SIFNA.xsd " >
      <localId>id text</localId>
      <leaName>Sample School District</leaName>
      <leaURL>http://www.sample.edu</leaURL>
      <educationAgencyType>state education agency</educationAgencyType>
      <phoneNumberList>
            <phoneNumber>
                  <number>555-555-5555</number>
                  <extension>201</extension>
                  <listedStatus>Yes</listedStatus>
            </phoneNumber>
      </phoneNumberList>
      <emailList>
            <email>sample@sample.com</email>
      </emailList>
      <operationalStatus>open</operationalStatus>
</lea>
```


Schema for the Extended Content

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.other.edu/samplexml/other-namespace"
      targetNamespace="http://www.other.edu/samplexml/other-namespace"
      elementFormDefault="qualified">
      <xs:element name="field1"/>
      <xs:complexType name="field2type">
            <xs:sequence>
                  <xs:element name="field3">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="field4"/>
                                    <xs:element name="field5"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="field6"/>
            </xs:sequence>
      </xs:complexType>
      <xs:element name="field2" type="field2type"/>
</xs:schema>
```


Example XML for the Extended LEA Object

```
<?xml version="1.0" encoding="UTF-8"?>
<lea refId="00000000-0000-4000-0000-000000000000"
      xmlns="http://www.sifassociation.org/datamodel/us/3.0"
      xmlns:n2="http://www.other.edu/samplexml/other-namespace"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.sifassociation.org/datamodel/us/3.0
SIFNA.xsd
      http://www.other.edu/samplexml/other-namespace OtherNamespace.xsd" >
      <!--Extended Fields-->
      <n2:field1>one</n2:field1>
      <n2:field2>
            <n2:field3>
                  <n2:field4>four</n2:field4>
                  <n2:field5>five</n2:field5>
            </n2:field3>
            <n2:field6>six</n2:field6>
            <field7>seven</field7>
      </n2:field2>
      <!--Regular Fields-->
      <localId>id text</localId>
      <leaName>Sample School District</leaName>
      <leaURL>http://www.sample.edu</leaURL>
      <educationAgencyType>state education agency</educationAgencyType>
      <phoneNumberList>
            <phoneNumber>
                  <number>555-555-5555</number>
                  <extension>201</extension>
                  <listedStatus>Yes</listedStatus>
            </phoneNumber>
      </phoneNumberList>
      <emailList>
            <email>sample@sample.com</email>
      </emailList>
      <operationalStatus>open</operationalStatus>
</lea>
```


Notice that a second namespace is declared and that the schema is added to the schemaLocation declaration.

```
<lea refId="00000000-0000-4000-0000-000000000000"
      xmlns="http://www.sifassociation.org/datamodel/us/3.0"
      xmlns:n2="http://www.other.edu/samplexml/other-namespace"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.sifassociation.org/datamodel/us/3.0
SIFNA.xsd
      http://www.other.edu/samplexml/other-namespace OtherNamespace.xsd" >
```

A New Approach to Code Sets

The implementation of code sets in previous versions of SIF was fragile. External code lists, which the SIF specification relied on, were frequently updated, and corresponding updates to the SIF representation of those code sets had to wait for updates to the specification. Allowing the flexibility of alternate code sets (OtherCodeList) has proven to be problematic in practice, and the pain is only increasing due to the rise of data profiles.

As a result of these pressures, SIF 3 data models adopt a new approach to code sets, which is intended to ensure that:

- code sets can be updated without requiring updates to the specification
- code set values can be transmitted with reference to a particular version of the code set
- users can exchange locally agreed code sets without needing them to be formally registered
- users can extend code sets freely while preserving the original code set values

The approach involves registering code sets in a SIF code set registry. Each registered code set is assigned an identifier, a name, a description, and a version; an updated version of a code set is registered as a distinct code set.

- The code set identifier is a hexadecimal digit, at least three digits long.
- The code set version is a hexadecimal digit; it is by default two digits long, expanding to four digits if required.

Any code set value used in SIF is referenced against the code set it comes from; if the code set is registered with SIF, the identifier used to register the code is referenced. Code-set-values are assigned identifiers on creation of the code set. The identifier for the code set value is a hexadecimal number (less than 0x10000000), and is stable; that means that once a code is entered it is never deleted, and a code value identifier can never be reused within a code set. Keeping the code set value identifier stable guarantees that it is interpreted consistently, according to the registered code set, without needing to add the code set to the SIF specification.

The registered code set value is accompanied by the following information in the SIF code set registry:

- The code set identifier and name

- The original text of the code set value in its source definition. The code text may or may not be numeric. Unlike the code set value identifier, the text code can change without forcing a new code set version to be registered.
- A long name for the code set value.
- A status for the code set value: the stats can be New, Change, or Delete.
- A description/definition of the code set value. Unlike the code set value identifier, the definition of a code set value can be modified, without forcing a new code set version to be registered.

Code set values that do not come from a registered code set must use a hexadecimal number above 0x10000000 for their identifier; this communicates that the code value is not registered with SIF. Implementations should still document locally defined codes before putting them into use.

The code set value zero is reserved for blank or missing values; it is not otherwise a valid value for a code.

The code set value is associated with other elements when transmitted through SIF as a gCodedElementType element:

- The code set that the value belongs to is identified through a code set name. If the code set is registered in SIF, SIF will have recorded that name against a code set identifier. A code set name must be provided, even if the code set is locally agreed and not registered.
- The code element is the original text of the code set value in its source definition.
- The code set identifier can be expressed as either a dashed or an undashed hexadecimal number.

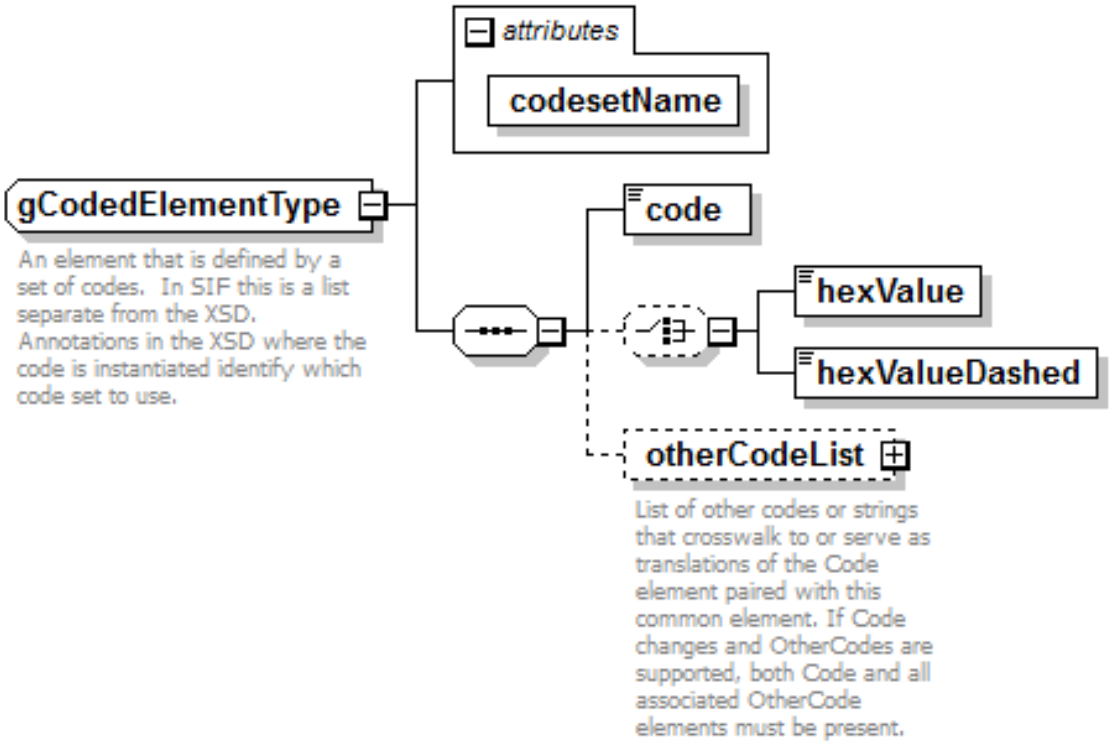
The code set value can optionally be accompanied by an otherCodeList, a list of other codes or strings that crosswalk to the code set value, or serve as translations for it.

Coded Field Type

Code sets are used in the SIF 3 schemas in conjunction with fields of type *gCodedElementType*. The XML for this type of element is a potentially very simple element. For example:

```
<codedElement codesetName="theCodeset">
      <code>3</code>
</codedElement>
```


However, this element has optional elements to handle other code lists as well as the option to transmit the hex value of the element in order to remove any possible ambiguity. The hex value can be with or without dashes, but only one hex value can be used at a time. The structure of the coded field type is shown below.

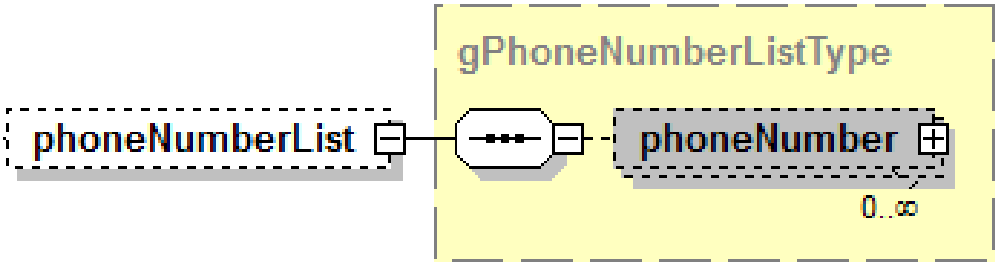


Lists in SIF 3.x

Lists in SIF 3.x have now been normalized as follows:

1. The name of the list will always be the element name followed by the word *List*, for example phoneNumber + List = phoneNumberList
2. Items in the list will be a single, repeatable, element. This element can be simple or complex.
3. The cardinality of the element will be 0 .. *.

For example:



In past versions of the SIF Standard, list items could be updated and deleted one element at time. This caused complexity in tracking individual list items. In version 3 data models, the whole list (including any changes such as added, changed, or deleted items) should be sent in a transmission. An empty list means all the items in the list have been deleted.

Important SIF Types

gRefIdType

As has always been the case in SIF, SIF objects are identified by a GUID; as of SIF 3.0, all objects are required to have a GUID. The GUID can be used in attributes and elements of objects, to reference other objects; for example, gAssociationType (associating two SIF objects), or gPersonType/addressrefIdList (associating a person to one or more address objects).

The GUID used to identify an object is a refID, and the SIF specification requires it to follow the RFC 4122 definition of a Type 1 or Type 4 GUID, as a 32-digit hexadecimal number, with the 13th digit identifying the GUID type.

Compliance with this definition is enforced by restricting refIDs to be of type gRefIdType: this enforces the following requirements:

1. The GUID is 32 hexadecimal digits long
2. The 13th digit of the GUID is either 1 or 4
3. The hexadecimal digits A through F can be either uppercase or lowercase
4. The GUID is hyphenated, with hyphens after the 8th, 12th, 16th, and 20th digits.

Note: In order to accommodate existing SIF data, i.e., SIF 2 datasets, requirement 2 above may need to be relaxed. It has been recognized that some existing SIF datasets do not have a 1 or 4 in the 13th place, most probably because they were not generated according to the RFC 4122 definition. SIF will in the future publish an alternate schema that allows any numeric digit or the letters a through f (lower and upper case) in the 13th place.

gRefIdPointerType

SIF 3.0 differentiates between gRefIdType and gRefIdPointerType (RefID and IDRef in SIF 2.x). gRefIdType is merely a GUID used as-is (for example, in gSIF_EntityType, for the GUID of an object). If a GUID is used to point to another object, it is considered an instance of gRefIdPointerType. Currently the two types have identical content; but gRefIdPointerType may be modified in the future, to contain more information useful for accessing a particular object instance.

gGenericRefIdType

SIF 3.0 also introduces gGenericRefIdType as a further specialisation of gRefIdType: this type includes both the refID of the object being pointed to, and the name of the object. This corresponds to SIF 2.x instances of IDRef accompanied by a SIF_RefObject attribute. This

allows the type of object being referenced to be constrained; for example, gStudentSchoolEnrollmentType/advisor is encoded as gGenericRefIdType, so that the type of object it points to is given, as well as its refID. A validator can then confirm that the type of object it points to is what it should be (i.e. staffPerson).

YearGroup

YearGroup is an internationalised name for an educational stage that normally lasts a year (and its associated cohort of students). In SiF 2.x, the names for this educational stage varied by locale: grade level in SIF US, year level in SIF AU, year group in SIF UK. The change of name allows the global data model of students and student enrolments to be used by all SIF locales, without arbitrary changing the common types invoked.

Collection Objects

Collection objects are new to SIF 3: they contain a number of instances of the same object type, returned in response to a request for all matching objects matching a particular query. Unlike lists, collections do not imply that its member objects have an association with each outside of the query; for example, an address list is intended to contain all the addresses of a single given entity, whereas an address collection may simply contain all addresses whose street name is five letters long. Any object can have an object collection, since any object can form the response to a query.

Collection objects only occur as the root level of a SIF message, as a single container containing a number of top-level objects. (In SIF 2.x, a query could return multiple root-level objects in response to a query; with SIF 3.0 used under REST, this is no longer practical.) A SIF object may not include collections: collections contain objects, not elements (so they are not themselves included in objects). Collections are in any case not well-defined as objects—so they do not have refIDs, and cannot be referenced by other objects either. SIF objects use refIdPointer lists instead in order to refer to multiple SIF objects.

Collection names are identified by taking the singular name of an object, and adding a lower-case **s** to the name. This convention holds in disregard of regular English spelling in order to make the names more machine-readable but still readily identifiable by humans. For example, the collection of address objects is called address and the collection of person objects is called persons.

Annotations

Annotations in SIF XSD documents contain two components: *documentation*, and *appinfo*. These are standard tags defined by XML.

The *documentation* component gives human-readable definitions of the elements, types, or objects to which the annotation applies. The documentation of an element is inherited; so if an element defined in the global layer is reused in a locale, the same definition is assumed to apply.

The locale-specific container or the global element may however have its own documentation, which can include usage notes about how the global element is to be understood. For example, the global type gSexusType is defined as "Sex of a person. This is different from gender". If the AU SIF locale consistently uses the global type to mean gender instead, then either the locale creates a new type for gender, or else annotates its locale-specific containers of the global type (for example, person or student), with a note saying "in the AU locale, gSexusType refers to gender not sex."

The *appinfo* component gives machine-readable metadata about elements, types, and objects. This metadata is used in the generation of documentation, and to provide information to services using the XSD documents (e.g. infrastructure services). The information includes the following:

- elementName: human readable, long form of the name of the element. Can contain spaces, as well as explanatory words not included in the XSD name of the element.
- sifChar: the obligation of an element: whether it is Optional, Mandatory, Optional Required, Mandatory Required, or Conditional. This information can be inferred from the XSD, but it is easier for automated generation of documentation to give it explicitly. Applies only to elements and not to types.
- cedsId: identifier of the CEDS element corresponding to this element or type.
- cedsURL: URL of the CEDS element corresponding to this element or type.
- events: whether an update to this element will generate an update event for the object containing it.
- isSIFObject; whether this type defines a SIF object or not.
- isCollectionObject: whether this type defines a collection (used in response to a query requesting multiple instances of an object)

Annotations for Elements

Below is an example of the annotation elements that can appear in a SIF element:

```
<xs:element name="schoolId" type="xs:string" minOccurs="0">
      <xs:annotation>
            <xs:appinfo>
                  <sifChar>O</sifChar>
                  <cedsId>001069</cedsId>
                  <cedsURL>
                        https://ceds.ed.gov/cedselementdetails.aspx?termid=3155
                  </cedsURL>
            </xs:appinfo>
            <xs:documentation>
                  A unique number or alphanumeric code assigned to an institution
                  by a school, school system, a state, or other agency or entity.
            </xs:documentation>
      </xs:annotation>
</xs:element>
```

Annotations for SIF Objects

Below is an example of the annotation elements that can appear in a SIF object:

```
<xs:element name="person" type="personType" >
      <xs:annotation>
            <xs:appinfo>
                  <elementName>Person</elementName>
                  <events>yes</events>
                  <isSIFObject>yes</isSIFObject>
                  <cedsId/>
                  <cedsURL/>
            </xs:appinfo>
            <xs:documentation>An individual within an educational
      setting.</xs:documentation>
      </xs:annotation>
</xs:element>
```

Namespaces

Namespaces are utilized in different manner in version 3 as compared to version 2 data models. In version 2 data models there was a single 2.* namespace that existed over all releases of the Specification. This 2.* namespace was updated overtime with the various releases of the Specification. For the version 3 data models, each release will create a unique namespace. So version 3.0 will have a 3.0 namespace, 3.1 will have a 3.1 namespace, and so on.

The version 3 SIF Data Model XSDs are packaged into separate folders according to type of object: Entity, Composite, and Report (see Design Patterns and Object Types in this document). Even though they are in separate files and folders, all these objects have the same namespace.

In the folder containing Entity objects, there are also XSD files for Common Types, Global Elements, and SIF Base Types. These XSD files do not have a namespace by themselves. These components are included into each SIF namespace so that a slow changing XSD file such as SIF Base Types can be a part of multiple namespaces and still be distinguished by its own version number.

Also, version 3 data models allow users to define one or more schemas in order to extend the SIF Specification for local requirements (see Extending SIF Objects in this document). Implementers **must** define a separate namespace for their extensions to the Specification. Implementers should create XML schemas (XSD files) for their extensions and are strongly advised to submit the schema files to the SIF Association for posting on the public website.

SIF Data Model to SIF Infrastructure Binding

SIF enhances interoperability among software components by providing both flexible (Entity objects) and concrete (composite/report objects) types of data structures with which to exchange data. The SIF data model can be transported in any number of ways. However, the recommended way to transmit SIF data objects is via the SIF 3 infrastructure.

Although this documents discusses usage of the SIF data model and not the SIF Infrastructure there is a conceptual overlap where a data model and a transport infrastructure must meet in order to for the data model objects to be able move from one application to another. Again, there are a number of methods to accomplish this. However, here we will discuss how SIF data objects can get on and ride the SIF infrastructure train, so to speak.

This section describes how the data model objects get hooked up to the infrastructure so they can move along the infrastructure tracks. Continuing the cargo and train analogy, the infrastructure determines the size of the packages, what contents can move, in what order, how to request a shipment, what to do with a package that you receive but did not expect, how often the trains run, whether you will receive your request in one or more packages, and many more things.

In order for an implementation of SIF to be successful, it must take into account purely data issues such mappings from source data to SIF and SIF to target data system, master data management, data integrity, data policy, etc. However, implementations must also make sure, from a transport perspective, that SIF data objects can get on the SIF infrastructure train and use the features provided by the infrastructure in order to maintain data integrity, accuracy, security, timeliness, etc.

In order to get on the train, SIF objects must make use of an infrastructure-data model binding in order to take advantage of the several transport options provided by the SIF Infrastructure.

The ***SIF Runtime Data Model*** consists of the SIF Physical Data Model coupled with a collection of object-specific ***Infrastructure Bindings*** that optimize the functionality obtained when payloads conformant with the SIF Physical Data Model are exchanged between applications using the SIF Infrastructure. The types of infrastructure binding are defined below.

SIF Infrastructure Binding Types

The following types of per-object infrastructure bindings are available when extending a version of a SIF Physical Data Model into a SIF Runtime Data Model. They only apply when the underlying infrastructure is SIF 3.0 or higher.

By including (standardizing) a given infrastructure binding in a SIF Runtime Data Model, client (Consumer) application developers are more likely to utilize it since they can count on it being supported.

Each object may have one or more bindings within a Binding Type.

Context

A Context is an optional object type-specific metadata element that may further scope the data contained in a particular object. When no context is specified then the binding is to the DEFAULT context.

For example a longitudinal Context binding might be placed on the Schedule Object type, with possible values of "*ThisTerm*" and "*NextTerm*". This would allow the SIF Infrastructure to separate the sources of current and future student schedule objects, which would be particularly useful during end of term reporting periods. This separation would be visible to the client even though the schemas of the objects were identical, and even if the supplying service interfaces shared a common implementation.

When specified as a binding, support for a Context is assumed to be mandatory.

Service Paths

These are predefined URL segments that are used to optimize Consumer Queries in important use cases. A typical Service Path binding might be "sections/{}/students". This allows the infrastructure to route a Query made to a URL containing "sections/1234/students" to a Service Provider that will respond with all Student objects currently "enrolled" in Section 1234.

Such Service Path bindings are most commonly used to "bridge" association objects such as that between Student and Section. Without a defined Service Path, the Consumer application would have to Query the StudentSectionAssociation Object Service Provider for all objects with a specific Section RefId. Upon receiving say 40 qualifying associations, the Consumer would then have to issue 40 additional Queries, one for each Student RefId.

With access to a Service Provider that supports the above Service Path binding however, the Consumer has to issue only a single Query request.

XQuery Templates

XQuery technology is used by the SIF Infrastructure to standardize the way in which Query Responses can be tailored to meet important Consumer requirements.

An XQuery Template is an XQuery script with externally defined parameters (ex: the RefId of an entity object used in the script). When a Consumer includes the token representing an XQuery Template in a Query Request along with the XQuery Template parameter values, depending upon the template, the Query Response can:

- Contain a subset of expected object elements (ex: no Student Health or Discipline information)
- Include calculated aggregates based on the data in multiple objects of the same type
- Represent a combination of data elements contained in multiple objects of multiple types

The exact format of the XQuery Template Response is specified by an XML schema specifically related to the XQuery Template. In addition, the inclusion of the XQuery Template itself in the Runtime Data Model binding standardizes **how** the defined elements of that Response must be produced.

Every XQuery Template binding in a SIF Runtime Data Model must include both the Schema and XQuery Template artifacts. As a result XQuery Template bindings are particularly useful for defining both Composite Objects (ex: StudentPersonal) and Report Objects (StudentSnaphot), even for Service Provider developers who do not plan to utilize XQuery Scripting technology directly in their implementations.

Events

The SIF Infrastructure supports the publication of change Events by an object Provider, and their asynchronous delivery to subscribing Consumers. Among other uses, this allows subscribers to "synchronize themselves with changes occurring to the data maintained by the owning Provider (for example the Student object data maintained by an SIS).

The default Event binding for a given object type is "Mandatory". If publishing Events is incorrect for a specific object type, it can be prevented by specifying an infrastructure

Event binding for that object type of "Disabled". Alternatively the Event binding for an object type can be "Optional"

A Report Object Provider would be unlikely to publish Events, implying all Report Object types have an Event binding value of "Disabled".

Whether a Composite object Provider would be required to publish Events might depend upon the particular Composite object type. The Event binding value for a Composite Object type could then be either "Mandatory", "Optional" or "Disabled".

An Entity Object should have an Event Binding of "Mandatory".

Request Types

The SIF Infrastructure supports 4 types of requests that a Consumer can invoke on an Object Provider: Query, Create, Update and Delete.

The default Requests Type binding for a given object type is all 4 requests set to "Mandatory". If one or more of those request types need to be restricted for a Consumers of a specific object type, they can be prohibited by an infrastructure Requests type binding for that object type, or set to "Optional".

A Report Object Provider would have a Request Type binding of "Mandatory", for Query and "Disabled" for the other request types. An Entity Object would have a Request Type binding of "Mandatory" for all request types.

Dynamic Query

A need was seen to provide a relatively "simple" way for a Consumer to dynamically include some limited qualifiers on any given Query which it could reasonably expect would be interpreted successfully by all Service Providers.

This functionality is achieved in the SIF 3.0 Infrastructure by attaching an additional "where" Query Parameter onto the URL specifying the intended Service Provider to which the Query is to be delivered. The value of this parameter qualifies which of the objects controlled by the Service Provider for a given object type should be returned in the Query response.

For example, a Dynamic Query designed to isolate and return all students who took the introductory course in Computer Science, might contain (depending on the Data Model) the following where clause:

?where=[(studentssections/section="CIS 101")]

In a more complex example, the URL Query arguments to the Student Service instructing it to return all students named "John Smith" would be:

../students?where=[(name/nameOfRecord/familyName="Smith")and(name/nameOfRecord/givenName="John")]

The Dynamic Query binding for an object type in the Data Model may be set to any of the following values:

- Disabled
- EqualityOperator (the "where" clause can include "=" operations to select a subset of all operations)
- OtherOperations (the "where" clause can include !=, <, <=, > and >= as well as the "=" operator)

Depending on the object type, the above Dynamic Query binding options may be specified for individual elements within the object.

Example Bindings

StudentSnapshot

Bindings	Value
Type	Data Report Object
Context	DEFAULT
ServicePaths	
XQueryTemplate	(URL of XQuery Template defining how to construct a Student Snapshot from the elements in a Student Entity or Student Personal Composite object
Events	*Disabled*
RequestTypes	Query: Mandatory Create: Disabled Update:Disabled Delete:Disabled
DynamicQuery	*Disabled*

Student

Bindings	Value
Type	Data Entity Object
Context	Current Archived
ServicePaths	schools/{}/students: Mandatory sections/{}/students: Mandatory
XQueryTemplate	
Events	*Mandatory*
RequestTypes	Query: Mandatory Create: Mandatory Update: Mandatory Delete: Mandatory
DynamicQuery	*EqualityOperator: Mandatory* *OtherOperators: Optional*

Glossary

Model Types

- **Conceptual Model** - A conceptual model is a description of a content domain, such as education information, that consists of the significant concepts, and the relationships among the concepts, in the domain. Conceptual models can be represented as a formal ontology. A conceptual model does not take into account database technology or any other details concerning how the information will be represented by computers.
- **Logical Model** - A Logical model is a rendering of a conceptual model that takes into account the technology that will be used to represent the information. The target technology for SIF is XML. However, the SIF logical model is also appropriate for use in relational database systems (RDBMS). Therefore, the SIF logical model defines objects, attributes, and the relations among entities using explicit cardinalities. UML is a framework for describing logical models.
- **Physical Model** - A physical data model describes a logical model in terms of the implementation environment. Such an implementation environment typically includes one or more of the following:
    - The physical means by which data objects are stored and retrieved, such as servers and disk drives, lookup tables, management tables, etc. The SIF Physical Data Model does not address this.
    - The complete definition of the format of each element within a set of defined data objects. The SIF Physical Data Model achieves this by standardizing its data structures in the XML Schema Definition (XSD) language, where an XML schema exactly defines the format of each element (mandatory and optional) comprising every SIF Data Object. Relations between data structures are represented using SIF Reference Identifiers (RefIds, see below).
    - The physical means by which data is exchanged between two communicating applications sharing a common messaging infrastructure. The SIF Physical Data Model is independent of the underlying infrastructure used to convey SIF Data Model-conformant message payloads.
- **Runtime Model** - A SIF-standard release from a locale such as the UK, AU or US combines a version of the SIF Runtime Data Model with a version of the SIF Infrastructure. The SIF Runtime Data Model consists of the SIF Physical Data Model coupled with a collection of object-specific "infrastructure bindings" that optimize the

functionality obtained when payloads conformant with the SIF Physical Data Model are exchanged between applications utilizing the SIF Infrastructure. The various "types" of infrastructure binding will be defined below.

SIF Data Models

- **Global Core** - The Global Core Model forms the basis for the logical model of all Locales, and is set up in order to facilitate easier alignment between locales. It is used to manage change in locales, so that alignment with other locales is enforced.
- **Locale** - A Locale Data Model is meant to be a complete SIF standard for a country or region in which terminology and education organizations are similar.
- **Instance** - A SIF Instance Data Model is a slight variation of a Locale Model based upon the needs of a particular implementation of SIF. The variations will be accomplished using SIF extension points and will be documented by the SIF Data Dictionary Tool. An instance model is intended to be associated with states, provinces, and local authorities.
- **SIF Entity Model** - The SIF Physical Data Model will contain two broad types of SIF objects, namely, Entity-type objects and other types of objects. Entity objects (defined below) are special because they each represent a distinct concept. Entity objects, together, represent a sub-model that is normalized, queryable, and flexible.

SIF Design Patterns

- **Data Structure -** This is a general term for a set of XML elements that are part of the same physical package.
- **SIF Object** - An arbitrarily defined data structure. In SIF, an object is a root-level complex XML type. However, not all complex types are SIF objects. The SIF object is a convenient concept that does not exist in XML but does exist in modelling paradigms.
- **Entity Object** - An Entity object represents a distinct concept taken from the logical and conceptual SIF models. The design of these objects is model-driven, not use-case-driven, so they may need to be combined to fit a particular use case. Examples of Entity objects include student, teacher, school, and student-school-enrollment.
- **Composite Object** - A Composite object is an object designed for a particular use case or limited set of use cases. The object is usually made up of the combined parts of

Entity objects. These objects are part of the SIF Physical Model but are not part of the Entity model (sub-model). An example of a Composite object is the StudentPersonal object from the US 2.6 specification. This object contains student information from the Student object in the Entity model as well as information from other objects.

- **Report Object** - A Report object is designed to represent point-in-time information and can contain summary information, cross tabulations, or information associated with a range of entities. This is the same kind of information as would be contained in a typical report.

SIF Elements and Types

- **RefId** - The Reference Identifier is a 32-digit hexadecimal number associated with an instance of a SIF data object which uniquely identifies the object instance.
- **SIF Base Type** - A SIF Base Type is an XML simple type. These are data structures that involve one and only one element that should be applied uniformly throughout the data model. For example, the postal code in a US address can be defined as a character string consisting of five numerals, a dash, and then four more numerals. Using this XML simple type, postal code can be defined uniformly throughout the data model. Not all simple types in the SIF Data Model are SIF Base Types.
- **SIF Common Type** - A SIF Common Type is an XML complex type. These are data structures that involve more than one element that should be applied uniformly throughout the data model. For example, the name of a person can be defined as first name, middle name, and last name. Using this XML complex type, name can be defined uniformly throughout the data model. Not all complex types in the SIF Data Model are SIF Common Types.
- **SIF Global Type** - A SIF Global Type is a SIF Base Type or a SIF Common Type inherited from the Global Core data model.
- **SIF Code Set** - A SIF Code Set is a closed set of values for a field (element) that can be defined as a set of values. In the SIF Data Model, Code Sets are specified in one of three ways: (1) an enumeration in the XML definition of the element in the data model, (2) an enumeration in a SIF Code Set file published separately from the Data Model, and (3) an enumeration (or range) published by an organization external to SIF. Number (3) is called an external code set and numbers (1) and (2) are called SIF Code Sets.

