LexEVS 6.0 Design Document - Detailed Design - XML Loader

Contents of this Page

Document Information

Author: Craig Stancl, Scott Bauer
Email: stancl.craig@mayo.edu, bauer.scott@mayo.edu
Team: LexEVS
Contract: CBITT BOA Subcontract# 29XS223
Client: NCI CBIIT
National Institutes of Heath
US Department of Health and Human Services

Revision History

Version	Date	Description of Changes	Author
1.0	5/14/10	Initial Version Approved via Design Review	Team

XML Loader Overview

Loads of LexGrid XML were formerly limited by the size of the Coding Scheme model element that could be constructed in memory. Loaded models had a single entry point at the CodingScheme element. With the intention of providing improved loading performance, access points to other levels of LexGrid Model elements, and a convenient format for authoring, the design of the 5.1 XML Loader has been updated for LexEVS 6.0 and will be implemented with the following considerations:

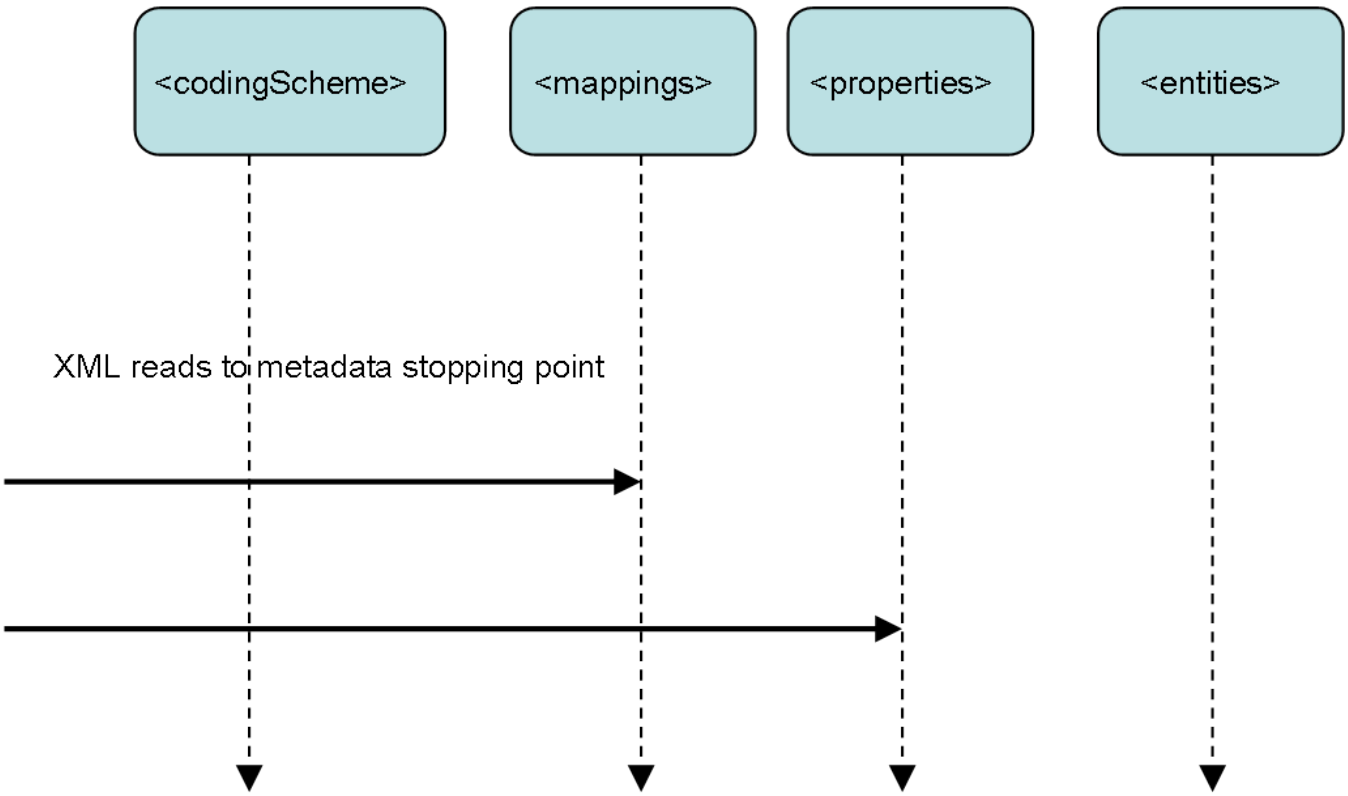
- A streaming implementation of the loading mechanism allowing larger loads in a smaller memory footprint.
- A variety of entry points to facilitate loading of Revisions, System Releases, Value Sets, Pick Lists, and Coding Schemes.
- A loading mechanism for Authoring based manipulation of LexGrid based xml files allowing entities and associations to be added to a given coding scheme in XML, then loaded into a LexGrid data repository.

Streaming XML Implementation

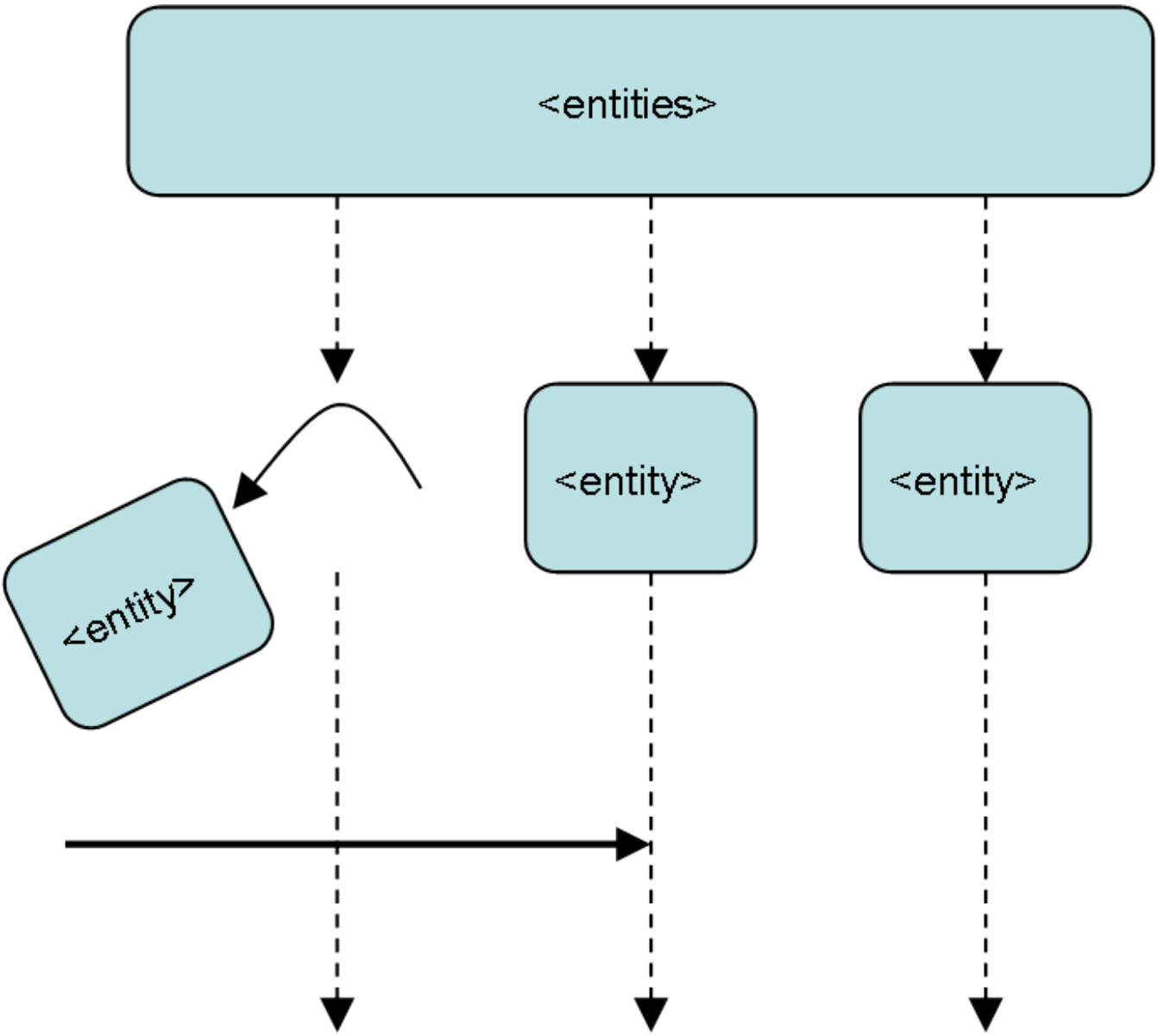
The latest implementation of the LexGrid XML loader provides a partitioned load of a coding scheme or other potentially large elements. The 6.0 loader will read coding scheme meta data into memory, load it to the LexGrid database and then begin stepping through entities and associations persisting elements to the database and then removing them from the coding scheme object as the unmarshaller reads objects from XML and into the coding scheme object in memory. Streaming XML reads while controlling the build of the coding scheme (and potentially other objects) in memory allowing the load of far larger terminologies constructed in LexGrid xml. This also allows users to eventually export larger coding schemes, revise them with authoring tools, and reload them as LexGrid Revision elements. This and other authoring scenarios will be exercised upon LexGrid XML with subsequent loads to the LexGrid database possible. (See authoring design below.)

LexGrid coding scheme meta data contains an number of required and optional elements as defined in the LexGrid schema. The last of these elements, before entity and associations are expressed, are the coding scheme mappings and the coding scheme properties. Mappings are required elements in the coding scheme and as such are a default stopping point of coding scheme reads. Coding scheme properties are not required and as such may not exist in a given rendering of LexGrid XML.

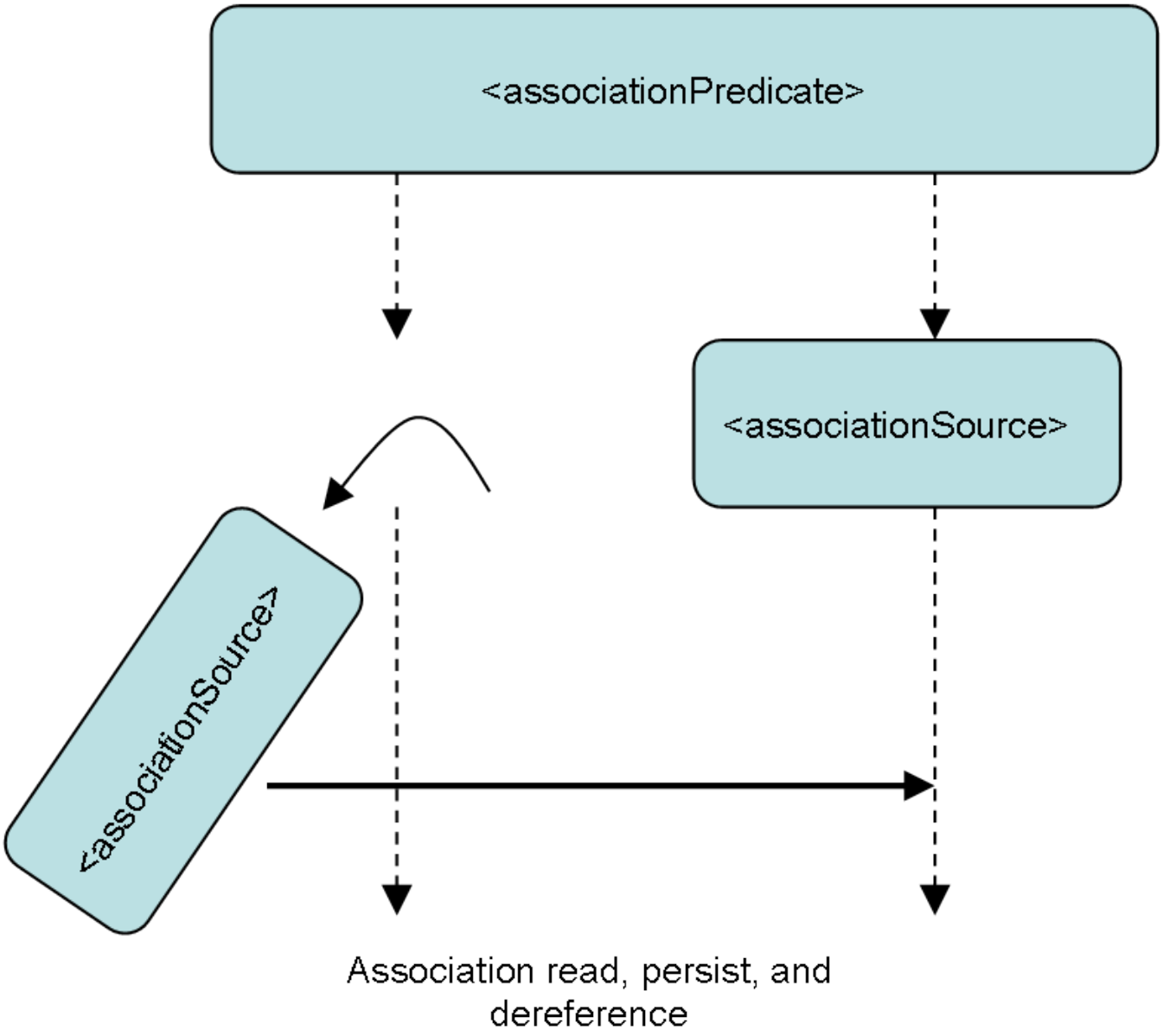
Before persistence of meta data to the LexGrid database takes place, a high speed pass is made over the coding scheme meta data portion of the XML with the STAX parser api to determine coding scheme properties presence.



The bulk of the coding scheme elements are read, stored to database, and de-referenced to allow sufficient memory management.



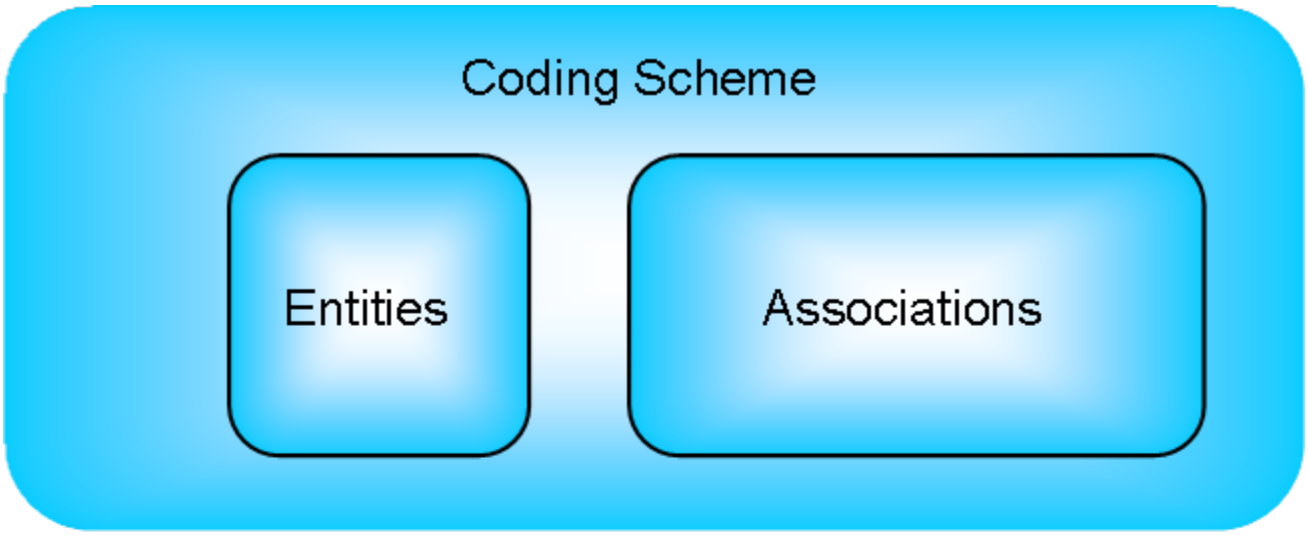
Entity read, persist, and dereference



Entry Points Expanded

Coding Scheme Entry Point

Previously, coding scheme was the only entry point for persisting loading to the LexGrid data base. Since pick lists, value domains, and revisions required more flexibility in XML based loads, entry points for revisions and system releases were also added.



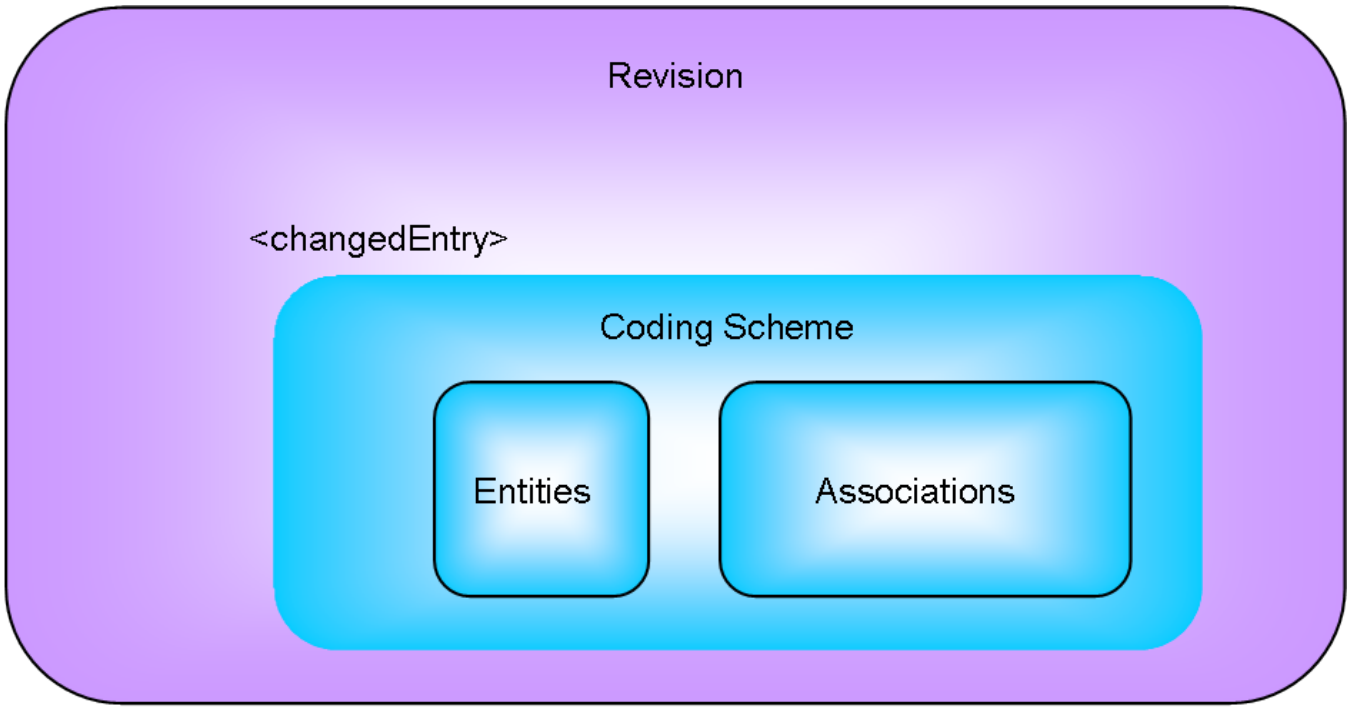

```
<codingScheme>

  <enitity/>

  <association/>
```

Revision Entry Point

Revisions can be applied to Coding Schemes, Pick Lists and Value Sets. A single revision element can load a set of revisions of coding schemes, pick lists and value sets.

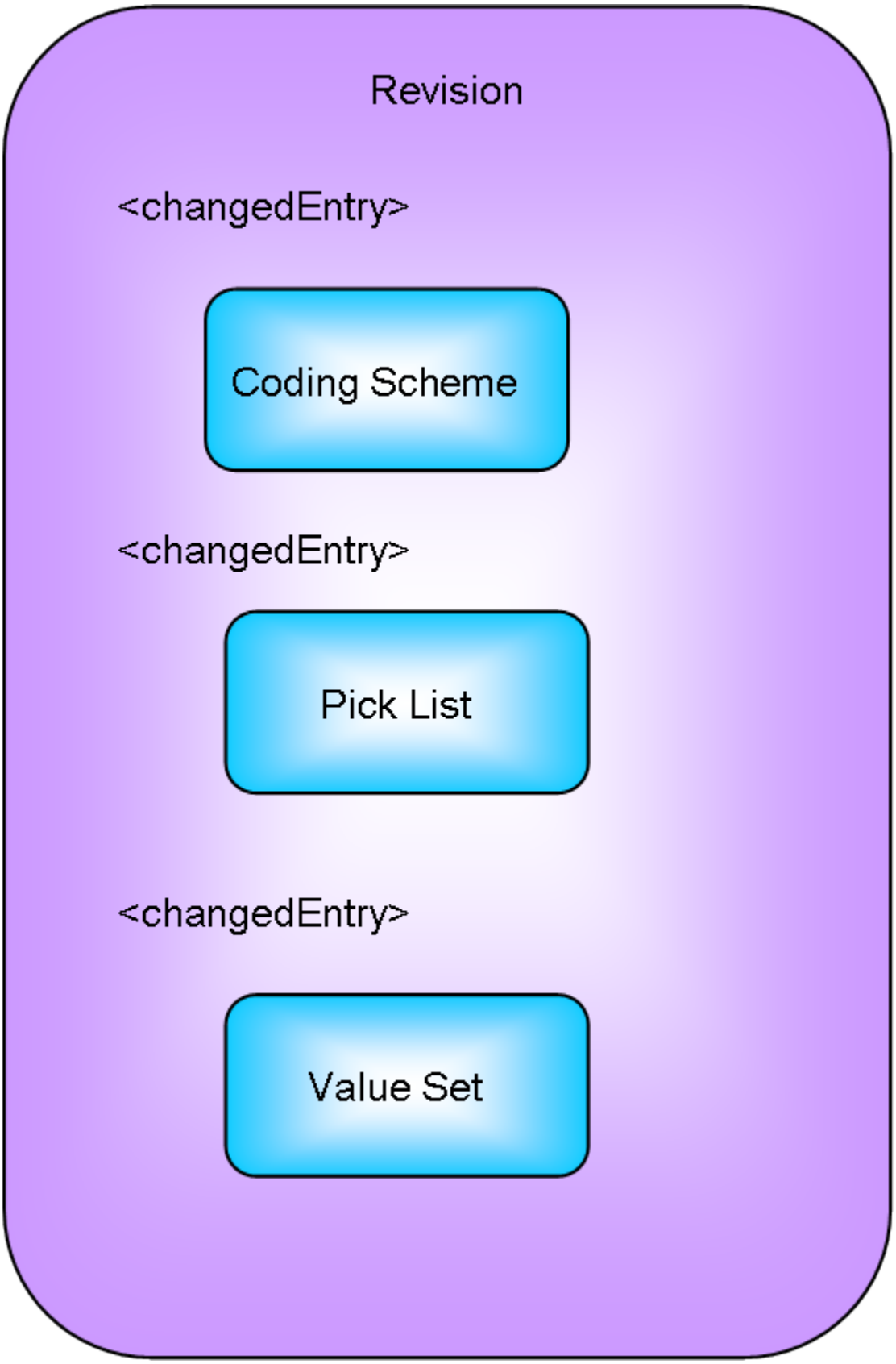



```
<revision>

<changedEntry>

<codingScheme>
```

Revision can contain a variety of changed elements.



System Release Entry Point

System Release is primarily intended as a release point for collections of pick lists and value sets. It is beyond the scope of this implementation to load multiple coding schemes contained within a system release The technical problems implied by the loads of a System Release containing multiple large coding schemes suggests that such a use case is impractical in many scenarios.