XCEDE 2.0 - A Manual

XCEDE 2.0 - A Manual

Table of Contents

List of Figures

Introduction

This is a manual for version 2 of XCEDE (XML-based Clinical and Experimental Data Exchange). XCEDE 2 is an extensible schema designed to enable the transfer and storage of several types of scientific data and metadata including (but not limited to) clinical, demographic, behavioral, physiological and image data. The target audience for this manual is anyone who is interested in using or learning more about XCEDE 2. This manual will serve as both a tutorial and as a reference.

Though the schema was designed in the context of neuroscientific projects, the structure of the schema is quite generic and can be applied as is to a wide variety of scientific disciplines.

1. Manual structure

This manual has two parts. Part I describes the various components of XCEDE 2 and describes the specification in some detail. Part II is a more practical guide to the use of XCEDE 2, providing guidance to those users who wish to extend the schema to represent project-specific metadata, or integrate XCEDE 2 into software applications and services.

2. Development

XCEDE 2 development started as an effort to "harmonize" the uses of XML within the testbeds of the Biomedical Informatics Research Network [BIRN] and their collaborators within the neuroscientific community. The feedback from these discussion sessions led to an initial prototype (based on XCEDE 1.0) of a next-generation schema. Over the course of four months of daily teleconferences, the first public release of XCEDE 2 took form.

XCEDE 2 has its origins in various XML schemas developed for collaborative neuroinformatics projects, including:

fMRI Data Center schema	a general scientific metadata storage framework.
XNAT (Extensible Neuroimaging Toolkit)	database and web service designed to facilitate management and exploration of neuroimaging and related data.
BIAC XML Header schema	provides a format-agnostic interface to image data and is the basis for the *binary data resource* in XCEDE 2.
BIRN Data Provenance schema	records the history of processing steps and is the basis for the `<provenance>` element in the schema.
XCEDE (1.0) SPM Toolbox	provides a general framework (and MATLAB interface) for storage and retrieval of thresholded statistical parametric maps and associated anatomical labels.

The primary developers of XCEDE 2 are (in alphabetical order): Syam Gadde (Duke University), Jeff Grethe (University of California, San Diego), Dave Keator (University of California, Irvine), and Dan Marcus (Washington University, St. Louis), all of whom have received support for this project through various BIRN testbeds.

3. For More Information

XCEDE is an evolving format -- later versions will be informed by feedback from users and developers. The latest schema and related information can be found on the project's web page at http://www.xcede.org/.

Part I. XCEDE 2 Core Specification

This section of the XCEDE 2.0 manual focuses on the high-level structure and major components of the XCEDE 2 Core schema. Each chapter contains a description of the component and provides examples of usage. The examples in this section are largely drawn from neuroscientific metadata, as this data served as the driving force behind development. However, these examples are by no means comprehensive; the schema was designed to be generally applicable (and extensible) to many types of scientific data, and this manual by necessity focuses on only a few.

Chapter 1. Structure and Conventions

1.1. The XCEDE 2 Dataset

An XCEDE 2 dataset is a data repository (stored in files, databases, or using other storage mechanisms) that can be represented as a collection of one or more XML documents, each of which validates against the XCEDE 2 XML schema (Appendix A). The XCEDE 2 specification does not prescribe any particular mechanism by which these documents are located, stored, grouped, or linked, though XCEDE 2 allows certain XML elements to link to other target elements and to optionally specify URLs as hints as to the location of the documents containing these targets.

For example, a given XCEDE 2 dataset may be stored as a single XML document, or a collection of files in a single directory on a file system, or may be distributed within a hierarchical directory structure (which may or may not reflect the structure of the data within), or may be stored within a database accessible by query through a web interface. The semantics of the dataset should be fully reflected in the XML representation, and should not be dependent on how the dataset is stored.

A schema-compliant XCEDE 2 document must be a valid XML 1.0 document [XML10], and must have one root element, `<XCEDE>`, which (like all other XCEDE 2 elements) is in the `http://www.xcede.org/xcede-2` XML namespace.

Several major components of the XCEDE 2 dataset are represented as children of the XCEDE root element. These components include:

- **Experiment hierarchy.** The components in the experiment hierarchy are represented by elements `<project>`, `<subject>`, `<visit>`, `<study>`, `<episode>`, `<acquisition>`. These are described in more detail in Chapter 2.
- **Data.** The element `<data>` is used to store actual data within the XML document. Examples include *events* (Chapter 6) and *assessments* (Chapter 8).
- **Data resources.** Represented by the element `<resource>`, these point to external files which store actual data. See Chapter 3.
- **Analyses.** The `<analysis>` element encapsulates metadata about data that is derived from one or more inputs, whose data may be represented by other XCEDE components, such as `<data>` or *data resources*. See Chapter 9.
- **Protocols.** Structures to describe the expected course of an experimental paradigm are provided by the `<protocol>` element. See Chapter 7.

Other XCEDE components appear as subcomponents of other XCEDE structures:

- **Data provenance.** The `<provenance>` element appears in the types `analysis_t` (on which the `<analysis>` element is based) and `dataResource_t` (one of the available data models for the top-level `<resource>` element). See Chapter 5.
- **Comments and annotations.** The `<commentList>` and `<annotationList>` elements provide mechanisms to store, respectively, descriptive text regarding the real-world entities or concepts represented by the associated XML element, or about the stored XML element itself. These are available in all elements based on `abstract_container_t`, which includes all the hierarchy elements and `<analysis>`.
- **Informational resources.** These point to documentation that may illuminate aspects of the data, data collection, protocol, etc. -- these might include peer-reviewed publications, equipment manuals,

and others. Informational resources are represented in the type `informationResource_t` which is used in the `<resourceList>` element in `abstract_container_t` (as with the comments and annotations).

- **Terminology.** Various elements in XCEDE 2 include the attribute group `terminology_ag` or use the type `terminologyString`, which allow the user to associate them with terms from particular nomenclatures/ontologies. For example, a subject's `<species>` may refer to a standard term listed in a well-known public nomenclature. See Chapter 10.

1.2. XCEDE 2 Schema conventions

The XCEDE XML document structure is defined using W3C XML Schema language [XMLSchema10]. With a few exceptions, the XCEDE 2 XML schema subscribes to the following conventions and best practices.

Naming and capitalization.

- Element and attribute names are alphanumeric (`<project>`). All letters are lowercase, with the exception of acronyms, which are uppercase (`<XCEDE>`, `projectID`), and, in multi-word names, the initial letters of the second and following words which are capitalized (`<dataResourceRef>`). For the purposes of this rule, `ID` (for "identifier") is considered an acronym.
- Schema type names (which do not appear in XML instance documents except in `xsi:type` attributes) follow the same naming conventions as element and attribute names, but have the `_t` suffix (`analysis_t`). If the type is "abstract", then it has an `abstract_` prefix (`abstract_container_t`).
- Attribute group names (which do not appear in XML instance documents) follow the same naming conventions as element and attribute names, but have the `_ag` suffix (`allLevelExternalIDs_ag`).

Schema best practices.

- Follow the schema versioning best practices recommended by Roger Costello in [CostelloSchemaVersioning]. Some of the implications are listed below:
  - The XCEDE namespace will change for major versions, but not minor versions. So, the namespace is as follows: `http://www.xcede.org/xcede-major_version`. For example: `http://www.xcede.org/xcede-2`
  - XSD files will be named `xcede-`*`VERSION-DOMAIN`*. For example: `xcede-2.0-core.xsd`
  - All elements should have a corresponding named `complexType`. For example: `<project>` is of type `project_t`.
  - The schema prescribes that some elements precede or follow other elements -- document creators should follow this strictly. However, applications should be written without expecting a particular element order, in case later versions of the schema change the order of elements. That is, be conservative in what you write, but liberal in what you accept.

    This is not to say that element order is not important -- the relative order of multiple elements with the same name should be preserved and can be relied upon by applications. So, if the schema specifies the content model of an element to have multiple `<A>` children and then multiple `<B>` children, applications should treat a dataset the same whether all the `<A>` children came before or after (or even interleaved with) the `<B>` elements, but all `<A>` elements should be presented to the application in the same order they appear in the document. The corollary is that the semantics of any XCEDE 2 element must not depend on the order in which differently-named children are presented in the document.

Document validation.

- The structure of an XCEDE 2 document can be validated using the XCEDE 2 Core schema. Several XML Schema validators are available to perform this task. Validating against the schema will ensure that generic XCEDE 2 parsers can read the document.

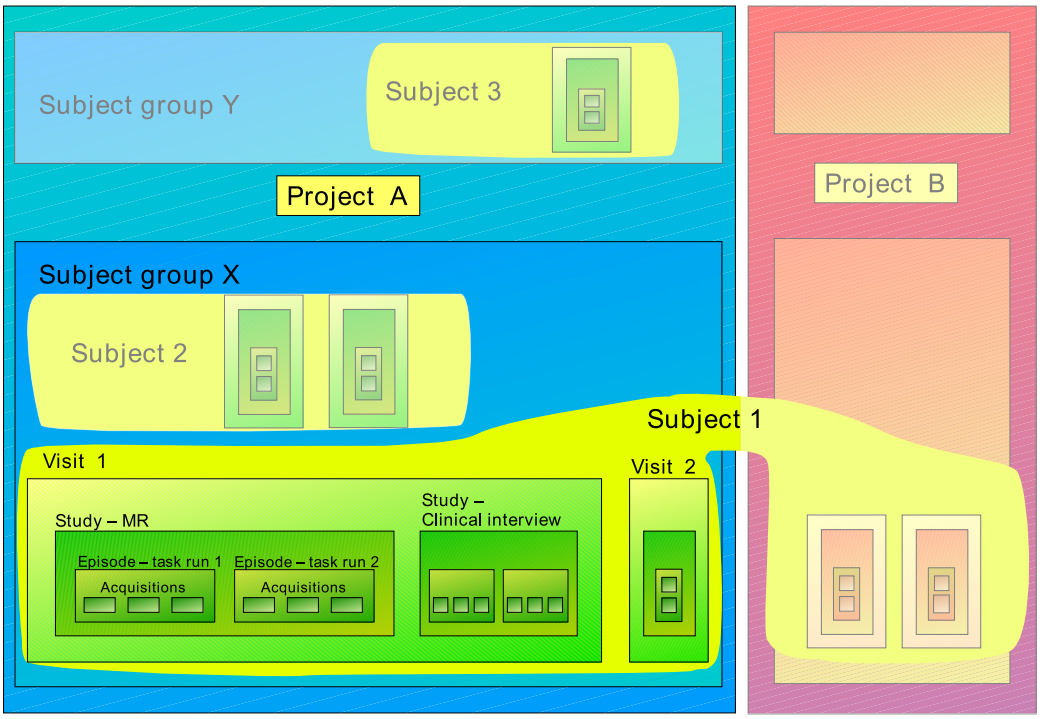
  Users may wish to perform *content-based* validation of an XCEDE document, for example, to make sure that "MR" image metadata stored in an XCEDE document have certain required fields such as TR, TE, magnet field strength, etc., and that their values fall within reasonable ranges. This kind of validation is not possible using XML Schema. However, other content-based schema languages exist for this purpose, most notably Schematron [Schematron]. As long as the XCEDE 2 Core schema is structurally sufficient to represent data for a given project or type of data, using a content-based schema language such as Schematron allows users to restrict or "specialize" the content of XCEDE documents without needing to modify or extend from the XCEDE 2 Core schema itself.

Chapter 2. Experiment Hierarchy

2.1. Overview

As illustrated in Figure 2.1, the XCEDE experiment hierarchy consists of several *levels* representing divisions of experiment data at various granularities. Elements at each level contain level-specific "info" elements, whose schema types may be derived to store experiment-specific or data modality-specific metadata. The linking mechanism between levels is flexible enough to support the omission of levels if the schema user finds them unnecessary.

Figure 2.1. XCEDE hierarchy



2.1.1. Hierarchy Levels

In the typical intended usage, a *project* is the top-level division of experiment data, and represents a research project which collects and analyzes data from one or more *subjects* which are divided (within the project) into *subject groups*. A subject may be a member of multiple research projects, and it is the subject group that maintains and distinguishes the mappings between subjects and research projects.

A *visit* may represent a subject's appearance at an experiment "site" (for collaborative projects, this could be the institution or lab at which the data is being collected or analyzed). A visit may be further subdivided into one or more *studies*, each of would consist of one or more data collection *episodes*.

Visit and study are more or less arbitrary divisions of the data that exist for convenience, and do not in themselves have any inherent meaning as far as the schema is concerned. However, an *episode* does have a meaning, and is intended to represent a unit of data collection by one or more instruments over a given time interval. Each set of data collected (perhaps by different instruments) over this time interval should be represented by an *acquisition*. Multiple acquisitions within an episode should be understood to occur simultaneously over the time interval represented by the episode. So, for example, an episode in an fMRI study may encapsulate the acquisition of a time-series of volume images from an MR scanner, as well as other acquisitions of behavioral or physiological data; all these (simultaneously collected) data would be stored as individual acquisitions and stored as part of the same episode.

At first thought, it might seem natural to represent the experiment hierarchy as described as a traditional XML hierarchy, where higher-level elements encapsulate lower-level elements as child elements. However, in XCEDE, all level elements (`<project>`, `<visit>`, etc.) are stored as children of the root `<XCEDE>` element. Links between levels are implicit in the level IDs assigned to each element and propagated to elements in lower levels. The great benefit of this approach is that applications are easier to write because all major elements are stored in the same place (under the XCEDE root element), and that XCEDE documents can be merged by merely concatenating the lists of top-level elements under a single XCEDE root element. Another advantage is to allow users of the schema to omit levels merely by omitting the unnecessary elements and IDs/links.

2.1.2. IDs and Linking

Specifying one or more *level IDs* allows one to explicitly define the hierarchical relations between XCEDE elements in different levels in the experiment hierarchy. Every level element has a set of these level IDs, composed of the element's own `ID` attribute, plus its "ancestor" ID attributes, indicating which higher-level elements have this element in their scope. For example, the `<visit>` element contains the ancestor ID attributes `subjectGroupID`, `subjectID`, and `projectID`. Level elements are linked to their ancestors in the hierarchy by sharing the appropriate subset of level IDs. In addition, any level element can linked from another XCEDE element that likewise specifies the appropriate level IDs.

For example, a link to a visit element may specify `visitID`, `subjectID`, `subjectGroupID`, and `projectID` attributes, and a `level` attribute with the value `visit`, indicating that this link is to a visit element. An application resolving this link will search for a visit element in the XCEDE 2 dataset whose attributes match those specified in the link (the `visitID` attribute in the link is matched against the `ID` attribute in the visit element).

Though the individual `ID` attributes are not required to be unique, the *set* of these level IDs applied to a level element must be unique. In addition, any link to a level element must provide enough level IDs to uniquely describe a single target level element. Level ID attributes not specified in the link are understood to match any value, but a link must specify enough of these IDs match at most one level element.

The elements that can link to level elements are `<catalog>`, `<resource>`, and `<data>` (children of the `<XCEDE>` root element), and `<inputRef>` and `<outputRef>` (children of the `<analysis>`

element). As previously described, level elements (except for <project> and <subject>, which are at the top of the experiment hierarchy), by virtue of specifying their own ancestor IDs, automatically incorporate links to their ancestor elements.

2.2. Examples

2.2.1. Metadata hierarchy

The metadata hierarchy illustrated in Figure 2.1 can be represented in XCEDE as shown in Figure 2.2 (only those elements/attributes relevant to linking are shown; the actual metadata contents of the elements are omitted for space). Note that subject “1” in the illustration is represented in two different projects, and so the corresponding <subject> element may be an ancestor for visits in both project “A” and “B”. However, the subject's role in each project is specified by the combination of the *project ID* and *subject group ID* (defined in the subject group lists in <project>), and these are used in every level element starting with <visit> and below.

Figure 2.2. Metadata hierarchy instance

```
<XCEDE xmlns="http://www.xcede.org/xcede-2">
  <project ID="A">
    <projectInfo>
      <subjectGroupList>
        <subjectGroup ID="X">
          <subjectID>1</subjectID>
          <subjectID>2</subjectID>
        </subjectGroup>
      </subjectGroupList>
    </projectInfo>
  </project>
  <project ID="B">
    <projectInfo>
      <subjectGroupList>
        <subjectGroup ID="Z">
          <subjectID>3</subjectID>
        </subjectGroup>
      </subjectGroupList>
    </projectInfo>
  </project>
  <subject ID="1" />
  <subject ID="2" />
  <subject ID="3" />
  <visit ID="1"
         projectID="A" subjectID="1" subjectGroupID="X" />
  <study ID="MR scan"
         projectID="A" subjectID="1" subjectGroupID="X" visitID="1" />
  <episode ID="task run 1"
         projectID="A" subjectID="1" subjectGroupID="X" visitID="1" studyID="MR" />
  <acquisition ID="MR image"
         projectID="A" subjectID="1" subjectGroupID="X" visitID="1" studyID="MR"
         episodeID="task run 1" />
  <acquisition ID="behavioral data"
         projectID="A" subjectID="1" subjectGroupID="X" visitID="1" studyID="MR"
         episodeID="task run 1" />
  <acquisition ID="heart rate"
         projectID="A" subjectID="1" subjectGroupID="X" visitID="1" studyID="MR"
         episodeID="task run 1" />
  <study ID="Clinical interview"
         projectID="A" subjectID="1" subjectGroupID="X" visitID="2" />
  <!-- ... etc. ... -->
</XCEDE>
```

Chapter 3. Binary Data Resources

3.1. Overview

The XCEDE 2 *Binary Data Resource* component is used to provide a generic interface to a binary data stream stored in one or more external files. Any of the binary data resource types described in this chapter can be used anywhere an `abstract_resource_t` is called for (with the appropriate `xsi:type` attribute); in the current XCEDE schema, these locations are the top-level `<resource>` element and the `<dataResource>` child element of `<acquisition>`.

XCEDE provides multiple layers of derived types to store more specialized information about the binary data. The base type and each of the derived types are described in turn.

`abstract_resource_t`. The abstract base type `abstract_resource_t` provides a few elements and attributes that are especially important for binary data resources. In particular, the `<uri>` element and its `offset` and `size` attributes point to a "chunk" of data stored in an external file. A series of `<uri>` elements define a stream of data that may be described in greater detail by the data types described below.

`binaryDataResource_t`. This type derives from `abstract_resource_t` and allows an application to interpret the data stream as a sequence of data items with a given data type (`<elementType>`) and byte order (`<byteOrder>`).

`dimensionedBinaryDataResource_t`. The data stream, until now, could only be interpreted as a one-dimensional sequence. This type provides `<dimension>` elements that allow the data stream to be interpreted as a multi-dimensional array of data items. Each dimension has a `<size>` and a `<label>`, as well as the ability to discard subsets of the data in the data stream (using the `outputSelect` attribute).

`mappedBinaryDataResource_t`. This type places the multi-dimensional array of data items represented by `dimensionedBinaryDataResource_t` into an arbitrary coordinate system.

3.2. Examples

Several examples of binary data are presented here, each showing the use of one of the different binary data types described in this chapter.

3.2.1. Basic data stream

The basic binary data resource type describes a sequence of data items. For example, consider a data file (`random_data_file.bin`) containing 2048 random 32-bit floating point numbers, stored in little-endian (least-significant-byte first) order. The `<dataResource>` describing this data is shown in Figure 3.1.

Figure 3.1. Simple `binaryDataResource_t` example

```
<dataResource xsi:type="binaryDataResource_t">
  <uri offset="0" size="8192">random_data_file.bin</uri>
  <elementType>float32</elementType>
  <byteOrder>lsbfirst</byteOrder>
</dataResource>
```

Note the `xsi:type` specifying that this `<dataResource>` element is of type `binaryDataResource_t`. (The `xsi:` prefix should have already been declared previously in the XML file using something similar to `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`)

The `<elementType>` element is restricted to one of several pre-defined strings (see the schema for details). The `<byteOrder>` element must be `lsbfirst` for little-endian data or `msbfirst` for big-endian data.

If the `<compression>` element is specified, it specifies that the file(s) pointed to by the `<uri>` elements are compressed. The content of the element should specify which type of compression (the only compression method specifically recognized by this specification is `gzip`). The `size` and `offset` attributes in the `<uri>` element always refer to the *uncompressed* data. An example of this is shown in Figure 3.2.

Figure 3.2. `binaryDataResource_t` with compression

```
<dataResource xsi:type="binaryDataResource_t">
  <uri offset="0" size="8192">random_data_file.bin.gz</uri>
  <elementType>float32</elementType>
  <byteOrder>lsbfirst</byteOrder>
  <compression>gzip</compression>
</dataResource>
```

As a special case, if the application does not find the file pointed to by a URI, and the `<compression>` element is *not* present, it may search for the same file with an appended `.gz` suffix, and if it exists, treat it as implicitly gzip-compressed data. Figure 3.3 shows how the same data in Figure 3.2 could be expressed using this alternative method. Pointing to the uncompressed version of the file (even when only the compressed version exists) allows the user to decompress or compress the data file at will, without affecting the ability of the application to read the data using the same `binaryDataResource_t`. Note that the `<uri>` and `<compression>` elements must be internally consistent. It would be an error to reference the uncompressed file `random_data_file.bin` and yet say that it was compressed using `<compression>gzip</compression>`. Likewise, explicit references to the compressed file (especially files that do not have the `.gz` suffix) must specify the compression method explicitly using the `compression` element.

Figure 3.3. `binaryDataResource_t` with implicit compression

```
<dataResource xsi:type="binaryDataResource_t">
  <uri offset="0" size="8192">random_data_file.bin</uri>
  <elementType>float32</elementType>
  <byteOrder>lsbfirst</byteOrder>
</dataResource>
```

3.2.2. Dimensioned data

Consider a camera that acquires an image using a 256x256 matrix of big-endian 32-bit signed integer voxels. This data has two spatial dimensions, which, by convention, we label x, and y (and z if a third spatial dimension is needed, and t if there is a time dimension). Figure 3.4 shows how this data might be represented.

Figure 3.4. `dimensionedBinaryDataResource_t` example

```
<dataResource xsi:type="dimensionedBinaryDataResource_t">
  <uri offset="0" size="262144">rawdata.img</uri>
  <elementType>int32</elementType>
  <byteOrder>msbfirst</byteOrder>
  <dimension label="x">
    <size>256</size>
  </dimension>
  <dimension label="y">
    <size>256</size>
  </dimension>
</dataResource>
```

Dimensions are ordered from fastest-moving to slowest-moving. So in the above example, the x dimension index changes on each consecutive data item, but the y dimension changes every 256 elements.

3.2.3. Mapped data

A “mapped” binary data resource is a (perhaps multidimensional) array of values, the matrix indices of which can be converted into a location in a given coordinate system. The location of the bounding box of the data in this space is given by specifying a location (in target-space coordinates) for the the first data item, and two things for each dimension: a unit-length direction vector (in the target-space coordinate system) and the spacing between successive data items in that dimension. The transformation matrix for a three-dimensional coordinate system has the form shown in Figure 3.5. This transformation matrix converts from matrix indices `(x,y,z)` to a coordinate location `(a,b,c)`. Figure 3.6 shows how the components of a transformation of MR image data into scanner RAS (Right/Anterior/Superior) coordinates are represented in a `mappedBinaryDataResource_t`. The unit vectors for each dimension are (X_A X_B X_C) = (1 0 0), (Y_A Y_B Y_C) = (0 1 0), and (Z_A Z_B Z_C) = (0 0 1), and are placed in the `<direction>` elements in each `<dimension>` element. The spacing values (S_X S_Y S_Z) = (3.75mm 3.75mm 4mm) are put in the `<spacing>` element in each `<dimension>`. The coordinates of the first voxel in the data are given by (O_A O_B O_C) = (-120 -120 -52).

Figure 3.5. Transformation matrix

$$\begin{pmatrix} X_A & Y_A & Z_A & O_A \\ X_B & Y_B & Z_B & O_B \\ X_C & Y_C & Z_C & O_C \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} S_A & 0 & 0 & 0 \\ 0 & S_B & 0 & 0 \\ 0 & 0 & S_C & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}$$

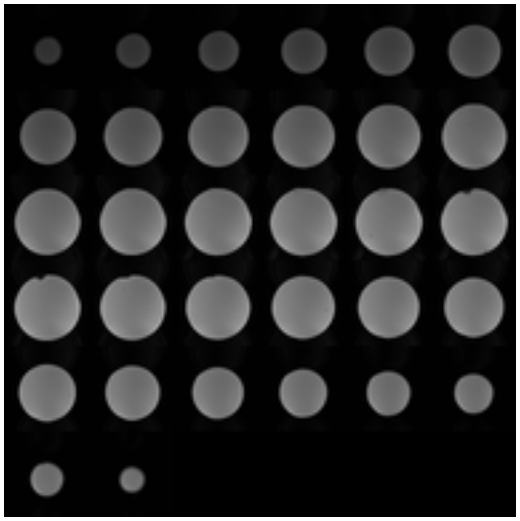
Figure 3.6. `mappedBinaryDataResource_t` example

```
<dataResource xsi:type="mappedBinaryDataResource_t">
  <uri offset="0" size="442368">V0001.img</uri>
  <uri offset="0" size="442368">V0002.img</uri>
  <uri offset="0" size="442368">V0003.img</uri>
  <uri offset="0" size="442368">V0004.img</uri>
  <uri offset="0" size="442368">V0005.img</uri>
  <!-- ... 135 more <uri> elements omitted for space ... -->
  <elementType>int32</elementType>
  <byteOrder>msbfirst</byteOrder>
  <dimension label="x">
    <size>64</size>
    <spacing>3.75</spacing>
    <gap>0</gap>
    <direction>1 0 0</direction>
    <units>mm</units>
  </dimension>
  <dimension label="y">
    <size>64</size>
    <spacing>3.75</spacing>
    <gap>0</gap>
    <direction>0 1 0</direction>
    <units>mm</units>
  </dimension>
  <dimension label="z">
    <size>27</size>
    <spacing>4</spacing>
    <gap>1</gap>
    <direction>0 0 1</direction>
    <units>mm</units>
  </dimension>
  <dimension label="t">
    <size>140</size>
    <spacing>2</spacing>
    <gap>0</gap>
    <datapoints>0 2 4 6 8</datapoints>
    <units>sec</units>
  </dimension>
  <originCoords>-120 -120 -52</originCoords>
</dataResource>
```

3.2.4. Advanced topic: split dimensions and outputSelect

A more complicated example is given by data generated by a Siemens MR scanner. In this case, the data represents a three-dimensional 64x64x32 image, stored in DICOM format. However, because the earlier versions of the DICOM format did not support three-dimensional data in one file, Siemens came upon the clever idea to “tile” the 32 two-dimensional slices across an NxN two-dimensional grid (Figure 3.7).

Figure 3.7. A “tiled”image



Applications may naturally want to express this data as a three-dimensional block, with columns, rows, and slices. In a conventionally-stored three-dimensional X×Y×Z image, the first X voxels compose the first row in the first slice, and then the next X voxels are the second row in the first slice; likewise the first X*Y voxels are the first slice, and the next X*Y voxels are the second slice, and so on. However, in the “tiled” image, though the first X voxels are again the first row in the first slice, the next X voxels are the first row in the second slice! At first it would seem that the dimension order has merely been switched, and specifying the labels of the dimensions as x, z, and y would fix things. However, we only hit six slices' first rows before hitting going to the second row of the same six slices. Only after going through all the rows in this fashion in the first six slices do we go on to the next six slices.

The end result is that the dimension that we are calling the z dimension has been split in two. The two components of the z dimension are interleaved with the x and y dimensions like so: x, z_1, y, z_2. The two components of the z dimension are distinguished with the `splitRank` attribute, as shown in Figure 3.8.

Figure 3.8. Split dimension example

```
<dataResource xsi:type="binaryDataResource_t">
  <uri offset="9240" size="589824">img0001.dcm</uri>
  <elementType>uint32</elementType>
  <byteOrder>lsbfirst</byteOrder>
  <dimension label="x">
    <size>64</size>
  </dimension>
  <dimension label="z" splitRank="1">
    <size>6</size>
  </dimension>
  <dimension label="y">
    <size>64</size>
  </dimension>
  <dimension label="z" splitRank="2">
    <size>6</size>
  </dimension>
</dataResource>
```

Applications should read this data as if it were four dimensions, and then permute the data to bring the two z dimensions together (in the order specified by splitRank) in the position of the highest-ranked split dimension, and the two dimensions can then be merged into one. The size of the new z dimension is the product of the sizes of the component split dimensions, so 6 * 6 = 36.

You may recall that the original data was acquired as a 64x64x32 volume, but the NxN tiling representation requires that the number of tiles be the square of an integer N. One more mechanism has been added to the `<dimension>` element to accomodate the presence of data that should be disregarded: the `outputSelect` attribute (see Figure 3.9).

Figure 3.9. `outputSelect` example

```
<dataResource xsi:type="binaryDataResource_t">
  <uri offset="0" size="589824">img0001.dcm</uri>
  <elementType>uint32</elementType>
  <byteOrder>lsbfirst</byteOrder>
  <dimension label="x">
    <size>64</size>
  </dimension>
  <dimension label="z" splitRank="1">
    <size>6</size>
  </dimension>
  <dimension label="y">
    <size>64</size>
  </dimension>
  <dimension label="z" splitRank="2" outputSelect="0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31">
    <size>6</size>
  </dimension>
</dataResource>
```

The `outputSelect` attribute specifies a list of indices along the given dimension (or combined dimension if it occurs on the highest-ranked component of a split dimension) that should be regarded as valid data. Data in the other indices should be ignored.

Chapter 4. Catalogs

4.1. Overview

Catalogs in XCEDE 2 are containers of related resources, resource references, and data references. They are recursive in that catalogs can contain a list of catalogs and catalog references.

Catalogs are useful in a number of contexts. A catalog could be created, for example, to represent all of the acquisition resources associated with an episode. These catalogs could be contained within a parent catalog that represents each of the episodes in an MR study. Catalogs could also be used to represent the various resources generated as part of an analysis or uploaded and tagged by users.

A benefit of using catalogs, as opposed to linking resources directly in analysis and acquisition elements, is that the catalogs can be sent independent of the parent content to client applications that choose not to support the full XCEDE specification. Additionally, catalog documents can be quite large when they contain thousands of entries (for example, when pointing to individual DICOM files for an acquisition), so separating the catalogs from their parents can be more efficient.

Catalogs are represented by the `catalog_t` complex type, derived from `abstract_tagged_entity_t`. The catalog child element of the root XCEDE element is of type catalog_t. The resources included in the catalog can be pointed to from by resource references in a number of places in XCEDE, including analysis_t and acquisition_t.

The abstract base type `abstract_tagged_entity_t` provides an unbounded set of `tags` that users and applications can populate to attach ad-hoc documentation and labels to the catalog and its contents. catalog_t also includes an unbounded list of child subcatalogs (or references to child catalogs) and an unbounded list of entries. Each entry is either a resource, a refernce to a resource, or a reference to an XCEDE data element (an analysis, for example).

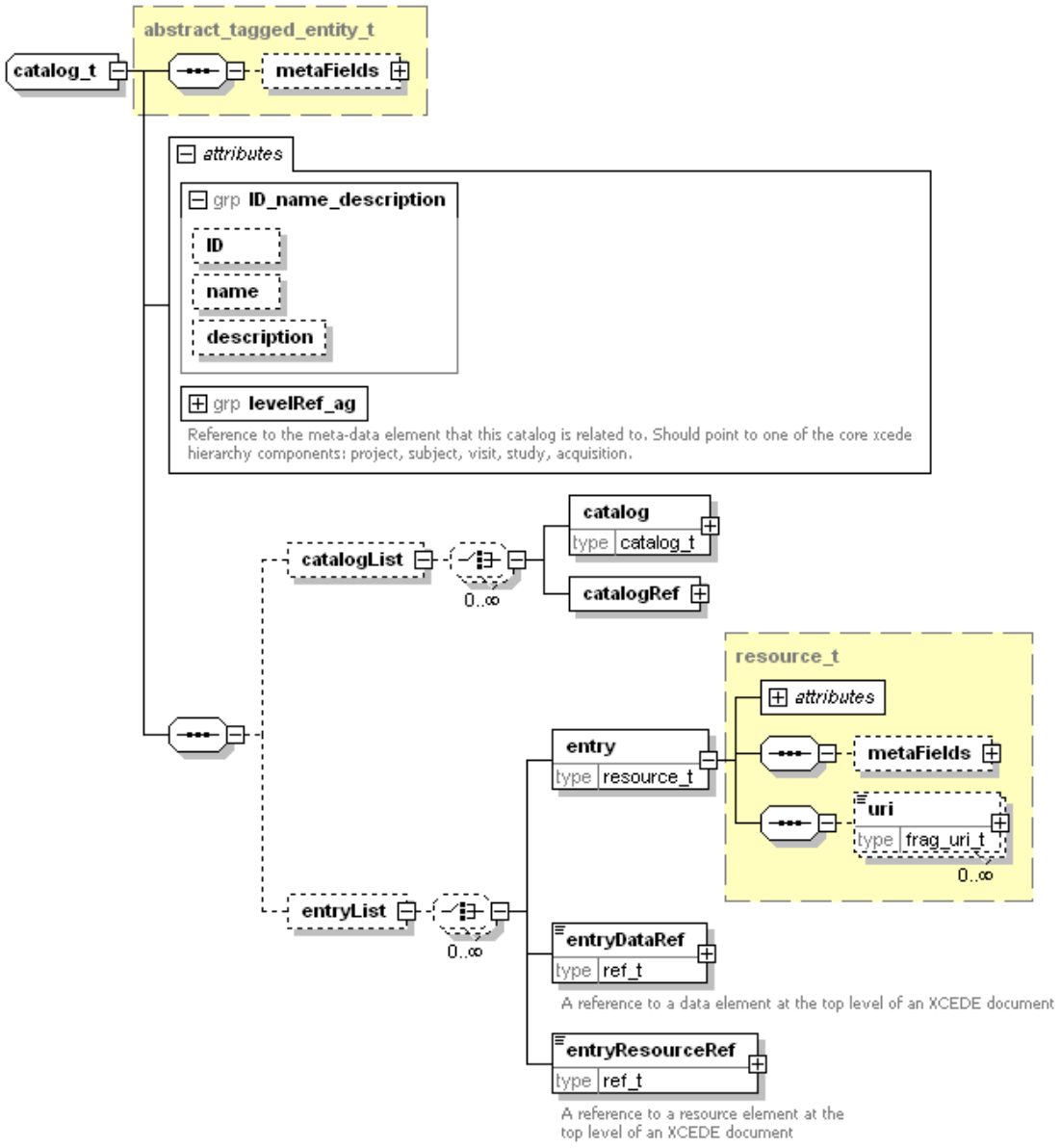
4.2. Examples

Figure 4.1. Catalog instance

```
<XCEDE>
 <catalog ID="ID0">
  <catalogList>
   <catalog ID="ID1">
    <entryList>
     <entry ID="ID2" name="lh.pial" description="pial surface of left
 hemisphere" format="FreeSurfer:surface-1" content="lh.pial" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/fbph2-000648622547/surf/lh.pial"/>
     <entry ID="ID3" name="brain" description="extracted brain
 mri" format="FreeSurfer:mgz-1" content="brain" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/fbph2-000648622547/mri/brain.mgz"/>
     <entry ID="ID4" name="zstat8" description="8th zstatisic
 contrast" format="nifti:nii-1" content="zstat8" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/sirp-hp65-stc-to7-gam.feat/stats/zstat8.nii"/>
     <entry ID="ID5" name="aparc+aseg" description="parcellation and segmentation
 label map" format="FreeSurfer:mgz-1" content="aparc+aseg" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/fbph2-000648622547/mri/aparc+aseg.mgz"/>
    </entryList>
   </catalog>
   <catalog ID="ID1">
    <entryList>
     <entry ID="ID2" name="lh.pial" description="pial surface of left
 hemisphere" format="FreeSurfer:surface-1" content="lh.pial" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/fbph2-000648622547/surf/lh.pial"/>
     <entry ID="ID3" name="brain" description="extracted brain
 mri" format="FreeSurfer:mgz-1" content="brain" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/fbph2-000648622547/mri/brain.mgz"/>
     <entry ID="ID4" name="zstat8" description="8th zstatisic
 contrast" format="nifti:nii-1" content="zstat8" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/sirp-hp65-stc-to7-gam.feat/stats/zstat8.nii"/>
     <entry ID="ID5" name="aparc+aseg" description="parcellation and segmentation
 label map" format="FreeSurfer:mgz-1" content="aparc+aseg" cachePath=""
 uri="file://c:/data/fBIRN-AHM2006/fbph2-000648622547/mri/aparc+aseg.mgz"/>
    </entryList>
   </catalog>
  </catalogList>
 </catalog>
</XCEDE>
```

4.3. Reference

Figure 4.2. catalog_t



Chapter 5. Provenance

5.1. Overview

Provenance is used to record in detail the sequence of processing and analysis steps that have been executed to create a derived image and/or measures. These details include the name and type of machine on which the executable was run, the name of the executable, and input parameters to the exectuable.

XCEDE-represented studies will be populated with large numbers of medical image volumes which researchers will wish to access and run various processing tools on in order to obtain derived data. This derived data, such as volumes of various brain structures, cortical thickness etc., will then be placed back into the database, linked to the original volume from which it was produced. If a researcher would like to recreate the processing steps, in order to apply it to a new data set, they will be able to obtain the information needed to download or reconstruct the tools that were used on the original data set. Information about the processing tools needs to be made available by the authors of the tools, is collected at run time by the processing scripts, and written into XCEDE.

`provenance_t.` This type contains an ordered sequence of processing steps (`processStep_t`) that were executed to create the data that this element is contained within, which can be `dataResource_t` or an `analysis_t.`

`processStep_t.` This type contains two attributes, ID and parent, indicating a unique ID for the step and its parent step, respectively. It includes a number of child elements that describe the conditions under which an executable was run. All of the child elements are formally optional. The following are strongly recommended:

program	The name of the executable. The version attribute is required, build number is optional. See instructions below for inserting version control content. Alternately, you can define it as a user string.
programArguments	The arguments passed into the executable. The attributes are optional, they are used to break down the full list of arguments into input and output arguments.
timeStamp	The time at which the program started to run, milliseconds are optional until cross platform code can be found.
user	Who was logged in and running this executable?
hostName	The unique id of the computer running this executable. Replaces Machine from old schema.
architecture	This is the type of hardware that the processing step is run on, sun, x86, etc. platform: This is the operating system under which the processing step is being run.

These are optional:

cvs	Revision control repository information. Use the CVS keyword Id, inserted into your source code file as $Id: chap_provenance.xml,v 1.8 2007/08/13 19:53:05 gadde Exp $ and it will be replaced with the file name, version, date of change and who changed it. Also works with SVN, but see SVN notes for enabling it.
compiler	What was used to compile this executable? Name and version.

library	The name and version of a linked library. Can be repeated as many times as necessary. This is dependent on the library in question, examples are given for VTK and Tcl and Tk.
buildTimeStamp	The time stamp showing when the executable was built.
package	This tag gives the overall package version, if the executable is part of a larger unit, for example, mri_convert is part of FreeSurfer, and the whole FreeSurfer package may have a different version number than the mri_convert executable that might change at a different rate.
repository	Where can the software be found? Link to a code repository or a web page. Can be extracted automatically if using SVN, via the HeadURL keyword.

5.2. Examples

Provenance elements are optional children of dataReource and analysis types and describe the sequence of processing steps that were executed to generate the data represented by the parent element. In the example below,

Figure 5.1. Simple provenance example

```
<dataResource xsi:type="binaryDataResource_t">
  <uri offset="0" size="8192">random_generated_data_file.bin</uri>
  <provenance ID="1">
 <processStep parent="1" ID="1">
  <program build="1.0" version="1.0">filter1</program>
  <programArguments inputs="-in analyze" outputs="-out minc">-size 10 -reps 100 -v -in
 minc -out nifti</programArguments>
  <timeStamp>11:20:37</timestamp>
  <user>xnatmaster</xnat>
  <hostName>lablin1</hostname>
  <architecture/>
  <platform version="String">String</platform>
 </processStep>
 <processStep parent="1" ID="2">
  <program build="String" version="1.2">filter2</program>
  <programArguments inputs="-in minc" outputs="-out nifti">-size 20 -reps 200 -v -in
 minc -out nifti</programArguments>
  <timeStamp>11:50:21</timestamp>
  <user>xnatmaster</user>
  <hostName>lablin1</hostname>
  <architecture>x86</architecture>
  <platform version="6">fedora</platform>
 </processStep>
  </provenance>
  <elementType>float32</elementType>
  <byteOrder>lsbfirst</byteOrder>

</dataResource>
```

5.3. Generating provenance XML from applications

The figure below illustrates a C code snippet (written by Nicole Aucoin and implemented in 3DSlicer) that generates provenance XML from an application. [Note: This may not be completely compliant with XCEDE2.]

```
void printAllInfo(int argc, char **argv)
{
  int i;
  struct tm * timeInfo;
  time_t rawtime;
  // yyyy/mm/dd-hh-mm-ss-ms-TZ
  // plus one for 3 char time zone
  char timeStr[27];

  fprintf(stdout, "<ProcessStep>\n");
  fprintf(stdout, "<Program version=\"$Revision: 1.8 $\">%s</ProgramName>\n", argv[0]);
  fprintf(stdout, "<ProgramArguments>");
  for (i = 1; i < argc; i++)
    {
    fprintf(stdout, " %s", argv[i]);
    }
  fprintf(stdout, "</ProgramArguments>\n");
  fprintf(stdout, "<CVS>$Id: chap_provenance.xml,v 1.8 2007/08/13 19:53:05 gadde Exp
 $</CVS> <TimeStamp>");
  time ( &rawtime );
  timeInfo = localtime (&rawtime);
  strftime (timeStr, 27, "%Y/%m/%d-%H-%M-%S-00-%Z", timeInfo);
  fprintf(stdout, "%s</TimeStamp>\n", timeStr);
  fprintf(stdout, "<User>%s</User>\n", getenv("USER"));

  fprintf(stdout, "<HostName>");
  if (getenv("HOSTNAME") == NULL)
    {
    if (getenv("HOST") == NULL)
      {
      fprintf(stdout, "(unknown)");
      }
    else
      {
      fprintf(stdout, "%s", getenv("HOST"));
      }
    }
  else
    {
    fprintf(stdout, "%s", getenv("HOSTNAME"));
    }
  fprintf(stdout, "</HostName><Platform version=\"");
#if defined(sun) || defined(__sun)
#if defined(__SunOS_5_7)
  fprintf(stdout, "2.7");
#endif
#if defined(__SunOS_5_8)
  fprintf(stdout, "8");
#endif
#endif
#if defined(linux) || defined(__linux)
  fprintf(stdout, "unknown");
#endif

  fprintf(stdout, "\">");
  // now the platform name
#if defined(linux) || defined(__linux)
  fprintf(stdout, "Linux");
#endif
#if defined(macintosh) || defined(Macintosh)
  fprintf(stdout, "MAC OS 9");
#endif
#ifdef __MACOSX__
  fprintf(stdout, "MAC OS X");
#endif
#if defined(sun) || defined(__sun)
```

```
# if defined(__SVR4) || defined(__svr4__)
  fprintf(stdout, "Solaris");
# else
  fprintf(stdout, "SunOS");
# endif
#endif
#if defined(_WIN32) || defined(__WIN32__)
  fprintf(stdout, "Windows");
#endif
  fprintf(stdout, "</Platform>\n");

  if (getenv("MACHTYPE") != NULL)
    {
    fprintf(stdout, "<Architecture>%s</Architecture>\n", getenv("MACHTYPE"));
    }

  fprintf(stdout, "<Compiler version=\"");
  #if defined(__GNUC__)
#if defined(__GNU_PATCHLEVEL__)
  fprintf(stdout, "%d", (__GNUC__ * 10000 + __GNUC_MINOR__ * 100 +
 __GNUC_PATCHLEVEL__));
#else
  fprintf(stdout, "%d", (__GNUC__ * 10000 + __GNUC_MINOR__ * 100));
#endif
#endif
#if defined(_MSC_VER)
  fprintf(stdout, "%d", (_MSC_VER));
#endif
  // now the compiler name
  fprintf(stdout, ">");
#if defined(__GNUC__)
  fprintf(stdout, "GCC");
#else
#if defined(_MSC_VER)
  fprintf(stdout, "MSC");
#else
  fprintf(stdout, "UKNOWN");
#endif
#endif
  fprintf(stdout, "</Compiler>\n");


  fprintf(stdout, "<Library version=\"%s\">VTK</Librarary><Library
 version=\"unknown\">ITK</Library><Library version=\"%s\">KWWidgets</Library>\n",
 VTK_VERSION, KWWidgets_VERSION);
  int major, minor, patchLevel;
  Tcl_GetVersion(&major, &minor, &patchLevel, NULL);
  fprintf(stdout, "<Library version=\"%d.%d.%d\">TCL</Library>\n", major, minor,
 patchLevel);
  //fprintf(stdout, "<Library version=\"%d.%d.%d\">TK</Library>\n", major,
  //minor, patchLevel);
#ifdef USE_PYTHON
  fprintf(stdout, "<Library version=\"%s\">Python</Library>\n", PY_VERSION);
#endif
  fprintf(stdout, "<Repository>$HeadURL:
 http://www.na-mic.org/svn/Slicer3/trunk/Applications/GUI/Slicer3.cxx
 $</Repository>\n");
  fprintf(stdout, "<ProcessStep>\n");
  fprintf(stdout, "\n");
}
```

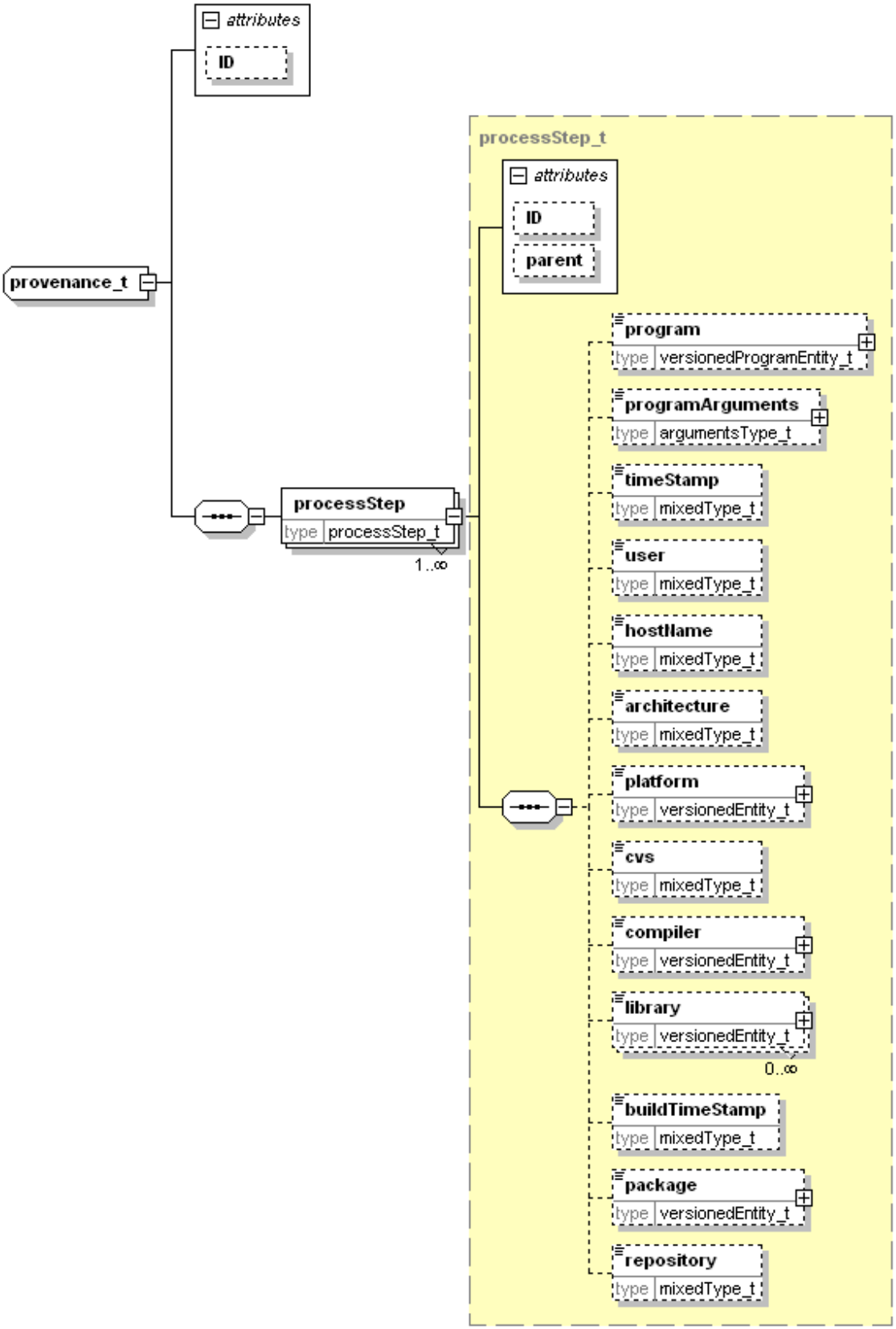
For the version, you can use the CVS keyword Name, inserted into your source code file as $Name: $ and it will be replaced with a the version tag when you check out the code with a tag:

```
cvs -r tag co [modulename]</programlisting>
```

If you are using SVN, in order to use CVS style keywords and have them be replaced with the appropriate values, you must enable the SVN repository's properties on a file by file (and keyword by keyword) basis by issuing the following command:

```
svn propset svn:keywords "Revision Id Date HeadURL" [filename.ext]
```

5.4. Reference



Chapter 6. Events

6.1. Overview

Events in XCEDE are merely time intervals annotated with arbitrary metadata. This component can be used to represent several types of behavioral data, statistics calculated on time series data, or any other metadata whose proper interpretation requires that it be associated with a particular time interval.

An XCEDE event consists of the following:

onset	The onset (in seconds) of the time interval.
duration	The duration (in seconds) of the time interval.
type	Usage of this field is user-specified
name	Usage of this field is user-specified
units	The units of the onset and duration fields. This field is optional, and it is recommended that users of the schema prescribe an implicit unit of measurement and use it consistently. In that case, this field may be considered informational only.
values	A *value* adds named metadata to this event.

The following instance shows how each of these fields may be populated.

```
<event type="visual" name="event#1" units="sec">
  <onset>0</onset>
  <duration>2</duration>
  <value name="shape">square</value>
  <value name="shapecolor">red</value>
</event>
```

Event elements are stored within the `<data>` element of an `<acquisition>`. The `<data>` element should be of type `events_t` (using `xsi:type` — see examples below).

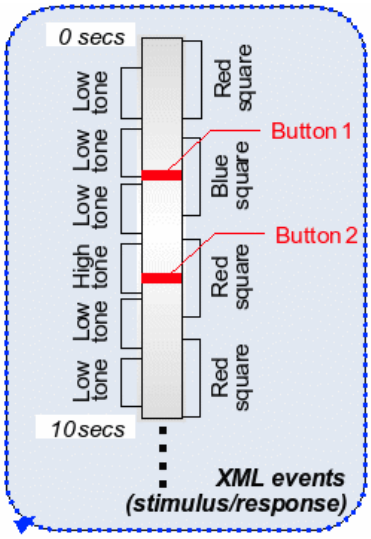
All onsets are relative to an arbitrary time reference. Typically, time 0 (zero) could mean the start of data acquisition. An event list may be interpreted as concurrent with data in other `<acquisition>` elements (which could be other event lists). If so, the same time reference should be used in all concurrent acquisition data.

There is no ordering constraint on events in a list. Applications should depend on using the `<onset>` elements to order the events chronologically if they so desire.

An optional `<params>` element may precede the first event in a list, and this element stores arbitrary metadata (using the same `<value>` element used above) that apply to all events in the list.

6.2. Examples

Figure 6.1. An event timeline



Consider the timeline shown in Figure 6.1, representing stimuli and responses in a neuroimaging study. We show in Figure 6.2 how the first 5 seconds' worth of the events might be represented in XCEDE.

Figure 6.2. XCEDE Events example - stimulus/response data

```
<XCEDE xmlns="http://www.xcede.org/xcede-2"
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
  <acquisition ID="my_stimulus_response_data">
    <dataRef ID="my_events" />
  </acquisition>
  <data ID="my_events" xsi:type="events_t">
    <event type="visual">
      <onset>0</onset>
      <duration>2</duration>
      <value name="shape">square</value>
      <value name="shapecolor">red</value>
    </event>
    <event type="visual">
      <onset>2.5</onset>
      <duration>2</duration>
      <value name="shape">square</value>
      <value name="shapecolor">blue</value>
    </event>
    <event type="audio">
      <onset>0.3</onset>
      <duration>1.4</onset>
      <value name="frequency">low</value>
    </event>
    <event type="audio">
      <onset>2.0</onset>
      <duration>1.4</onset>
      <value name="frequency">low</value>
    </event>
    <event type="audio">
      <onset>3.5</onset>
      <duration>1.4</onset>
      <value name="frequency">low</value>
    </event>
    <event type="response">
      <onset>3.4</onset>
      <value name="button">1</value>
    </event>
  </data>
</XCEDE>
```

Each stimulus and each response are stored as separate event elements. Note that all the visual events appear first in the XCEDE file, then the audio events, and then the response event. This ordering is arbitrary, and the events could easily have been presented in chronological (or random!) order. The semantic interpretation of the events within an event list must not depend on their document order.

Stimulus and response data are not the only appropriate content to represent in XCEDE events. Figure 6.3 shows how quality assurance (QA) statistics for each volume/timepoint in an fMRI scan can be stored as events. Note that the time reference for the onsets is arbitrary, but if the `acquisition` containing the event data is contained within the same `episode` as other time-locked data, it should be assumed that the time reference is the same for all `acquisitions` within the `episode`, unless otherwise explicitly specified. So, for example, this QA data might be associated with MR image data in the same `episode`, and time 0 (zero) would by default be assumed to have the same meaning in both sets of data.

Figure 6.3. XCEDE Events example - QA data

```
<XCEDE xmlns="http://www.xcede.org/xcede-2"
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
  <acquisition ID="my_stimulus_response_data">
    <dataRef ID="my_events" />
  </acquisition>
  <data ID="my_events" xsi:type="events_t">
    <event>
      <onset>0</onset>
      <duration>2</duration>
      <value name="volmean">759.218</value>
      <value name="cmassx">106.781</value>
      <value name="cmassy">118.279</value>
      <value name="cmassz">66.9694</value>
    </event>
    <event>
      <onset>2</onset>
      <duration>2</duration>
      <value name="volmean">759.218</value>
      <value name="cmassx">106.801</value>
      <value name="cmassy">118.242</value>
      <value name="cmassz">67.1636</value>
    </event>
    <!-- ... etc. ... -->
  </data>
</XCEDE>
```

Chapter 7. Protocols

7.1. Overview

Specifications of experimental protocols are stored in XCEDE using the `protocol_t` tag structures.

Protocols in XCEDE consists of a hierarchial organization of steps defining the protocol and items within the steps defining particular

`protocol_t`. The `protocol_t` tag extends the `abstract_protocol_t` tag to include steps and items making up a protocol description.

`abstract_protocol_t`. The `abstract_protocol_t` abstract tag provides basis information related to a protocol such as protocol time offset, min and max occurences, concept linkages, and more described belowThe `protocol_offset_t` tag is used to specify timing of this particular protocol relative to other protocols or steps within an experiment. The `protocolTimeRef` and `preferrredTimeOffset` tags are used to describe the offset

7.2. Examples

7.3. Reference

Chapter 8. Assessments

8.1. Overview

Complete data storage and assessment specifications and descriptions are stored in XCEDE using the `assessment_t` and `assessmentDescItem_t` tag structures.

In XCEDE the formal definition of an assessment and the cataloging of the actual data values collected for that assessment are specified in different parts of the schema. The formal description of the assessment questions and possible answer choices are specified in the protocol section `protocol_t` of the schema whereas the actual assessment data for an acquisition are stored in the `assessment_t` tags

`assessment_t`. The `assessment_t` tag extends the `abstract_data_t` tag to include information about the acquired assessment.The *name* element contains the assessment name which is unique within the document and links the acquired listing of assessment data back to the formal assessment definition optionally contained within the protocol descriptionThe *dataInstance* element specifies the state of the acquired data. Specifically whether it is first entry, second entry, validated, unvalidated, etc.

`assessmentInfo_t`. The `assessmentInfo_t` tag is derived from the `abstract_info_t` element and is used to store a description of the assessment

`assessmentItem_t`. The `assessmentItem_t` tag is used to store the actual data captured by the assessment. The `assessmentItem_t` tag is composed of a `<valueStatus>` element which contains information about whether the subject declined to answer the question or why the data value might be missing. The `<value>` element stores the actual value of the assessment item as captured on the assessment iteself. The `<normValue>` is used to store the normalized value for the element. The `<reconciliationNote>` is used to document whether the values have been reconcilied if this data is part of multiple data entries. The `<annotation>` is used to annotate the assessment item.The `terminology_ag` reference is used to give contextual meaning to the question and link it with known ontologies and concepts.

8.2. Examples

The assessment example below shows both a formal assessment description stored in the `<protocol>` block and also the actual acquired assessment data

8.2.1. Formal Assessment Description

The formal description of each question for the assessment and possible answer choices

Figure 8.1.

```
<xcede:step ID="SES" name="Socio-Economic Scale" minOccurences="1" maxOccurences="1"
               required="true">
                  <xcede:items>
                   <xcede:item ID="ses_education_subject">
                       <xcede:itemText>
                           <xcede:textLabel location="leadText"
                               value="What is the highest level of education or
professional training
                               that you have achieved?"/>
                           <xcede:textLabel location="trailText"
                               value="Any trailing text might go here"/>
                       </xcede:itemText>
                       <xcede:itemChoice itemCode="1"
                           itemValue="professional or graduate training (received
degree)"/>
                       <xcede:itemChoice itemCode="2" itemValue="college graduate"/>
                       <xcede:itemChoice itemCode="3" itemValue="some college (at
least one year)"/>
                   </xcede:item>
                   <xcede:item ID="ses_education_p_caretaker_prior_18">
                       <xcede:itemText>
                           <xcede:textLabel location="leadText"
                               value="What is the highest level of education or
professional training that your primary caretaker until you were 18 years old has
achieved?"/>
                       </xcede:itemText>
                       <xcede:itemChoice itemCode="1"
                           itemValue="professional or graduate training (received
degree)"/>
                       <xcede:itemChoice itemCode="2" itemValue="college graduate"/>
                       <xcede:itemChoice itemCode="3" itemValue="some college (at
least one year)"/>
                   </xcede:item>
               </xcede:items>
           </xcede:step>
```

8.2.2. Actual Acquired Assessment Data

An instance of actual assessment data acquired on a subject during a protocol collection

Figure 8.2.

```
<xcede:data xsi:type="xcede:assessment_t" subjectID="00301882920">
        <xcede:name>Socio-Economic Status</xcede:name>
        <xcede:dataInstance validated="true">
            <xcede:assessmentInfo>
                <xcede:description>This is the socio-economic scale. I'm sure there is
 more
                    intersting things to say about it but I have no idea
 what</xcede:description>
            </xcede:assessmentInfo>
            <xcede:assessmentItem ID="ses_education_subject">
                <xcede:value>1</xcede:value>
            </xcede:assessmentItem>
            <xcede:assessmentItem ID="ses_education_p_caretaker_prior_18">
                <xcede:value>2</xcede:value>
            </xcede:assessmentItem>
            <xcede:assessmentItem ID="ses_education_p_caretaker_lifetime">
                <xcede:value>2</xcede:value>
            </xcede:assessmentItem>
            <xcede:assessmentItem ID="ses_education_s_caretaker_prior18">
                <xcede:value>1</xcede:value>
            </xcede:assessmentItem>
        </xcede:dataInstance>
    </xcede:data>
```

8.3. Reference

Chapter 9. Analysis

The analysis_t type and its corresponding elements are used to document the collected output and/or results from an analysis or processing of data.

9.1. Overview

An analysis is composed of the "inputs" (i.e. the files and paramenters used in an analysis or processing of data), the application(s) or method(s) used and the resultant data (i.e. values and output files):

Analysis Constituents

`provenance`	The record of the origin and transformations applied to source data that produced this analysis. More information on this element can be found in the provenance chapter.
`inputRef`	???
`outputRef`	???
`measurementGroup`	A measurementGroup contains information and data related to the outcome of an analysis. For example, this could be a statistic (e.g. from a Statistical Parametric Map) or a measurement (e.g. the volume of the hippocmapus).

9.2. Measurement Group

A measurementGroup contains information and data related to the outcome of an analysis and is based on the measurementGroup_t type. Extending abstract container, this allows for the annotation and documentation of the measurements being described. The measurement being described by this element are described by two specific elements:

Measurement Group Constituents

`entity`	In the context of a measurement Group, is the entity being measured. For example, the hippocampus. In more complicated instances this could be an entity described by multiple entries, for example, when doing a region of interest analysis where multiple or multiple partial regions are involved.
`observation`	In the context of a measurement Group, are the observations that are made concerning the entity. For example, following hte hippocmapus example, might be the total volume, surface area, etc.

The `entity` element is based on the `abstract_entity_t` type that allows for the definition of specific entity classes. Currently two such classes are defined:

Entity Classes

`Anatomical Entity`	Defines an anatomical entity through a set of terms that name this entity. This set can contain a collection of synonymous terms. In addition to the anatomical region itself a laterality and tissue type can be specified to narrow the focus (e.g. left hippocampal grey metter).

`Atlas Entity`	Defines an atlas based region through the definition of geometries. These are defined on the `abstract_geometry_t` type.

9.3. Examples

9.4. Reference

Figure 9.1. terminology_ag

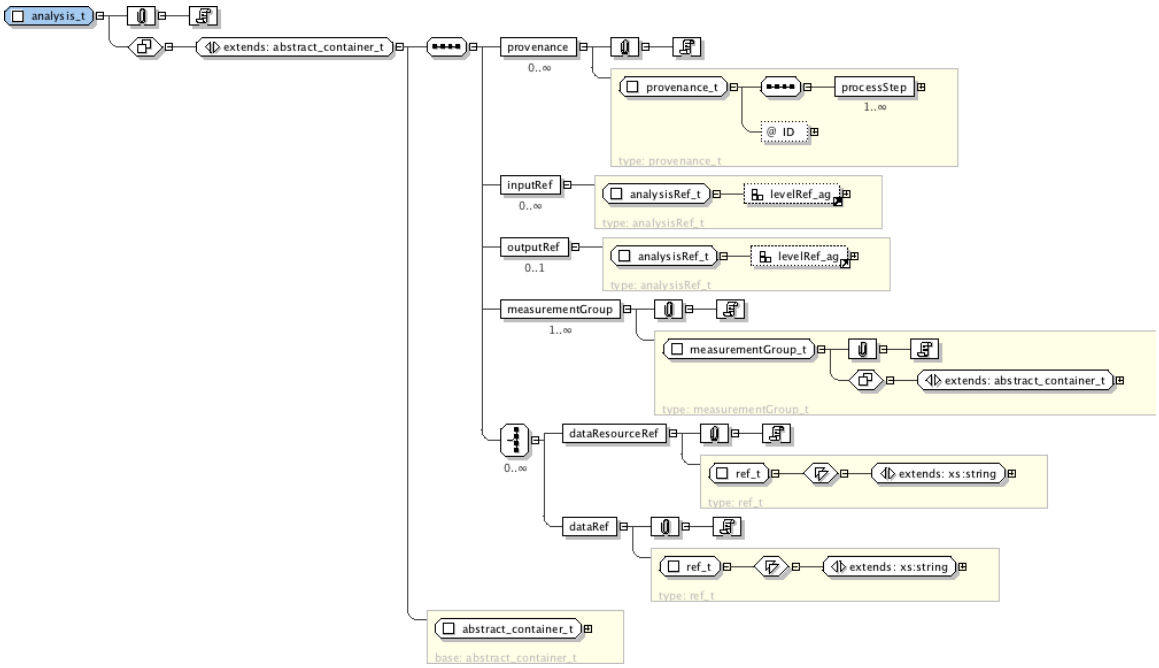


Figure 9.2. terminology_ag

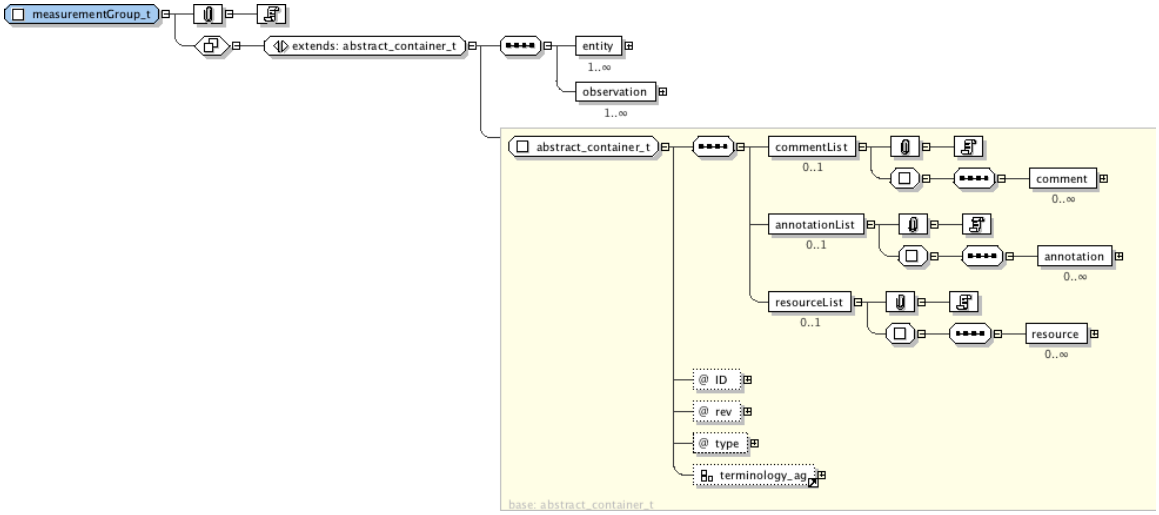


Figure 9.3. terminology_ag

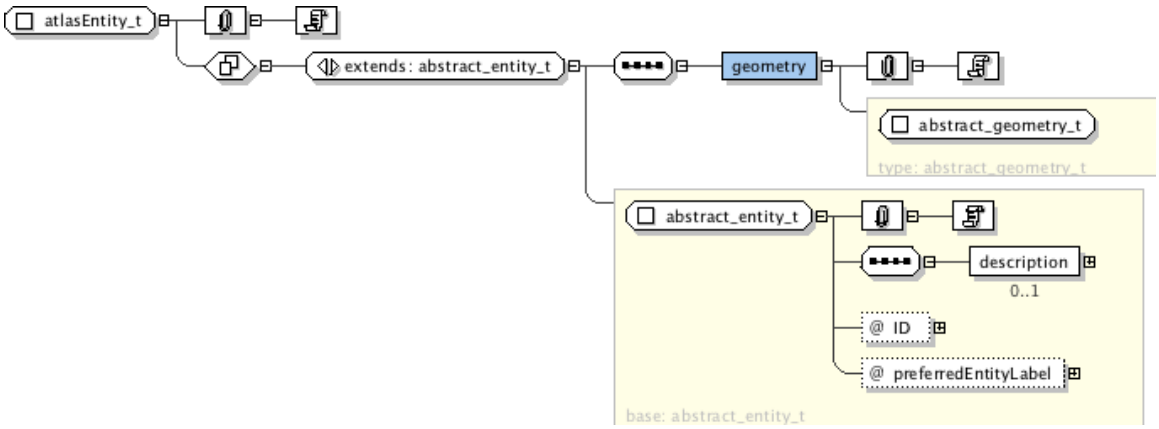


Figure 9.4. terminology_ag

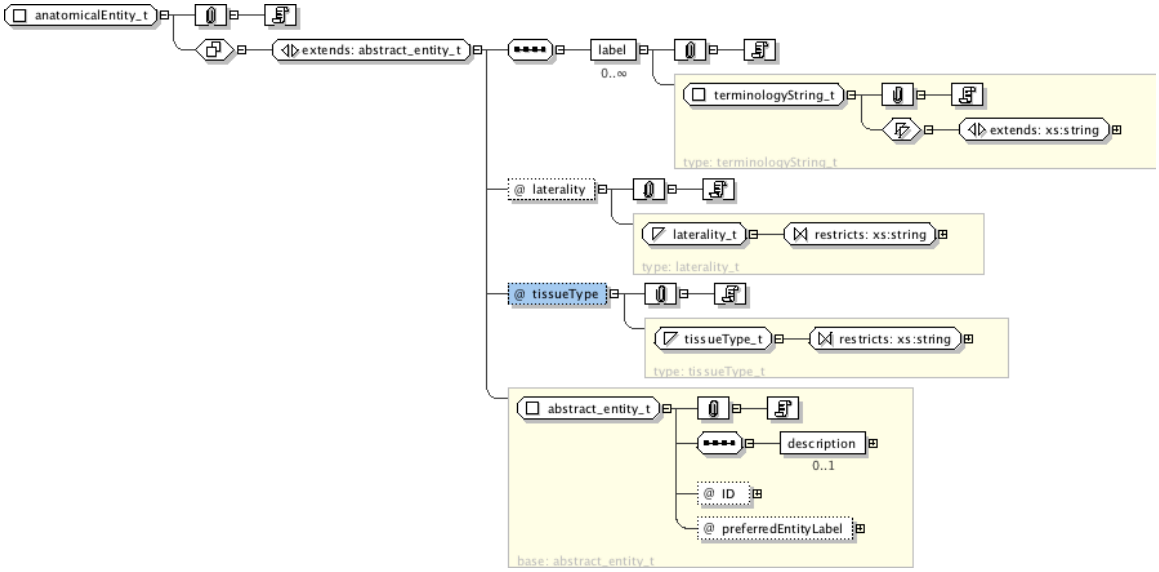
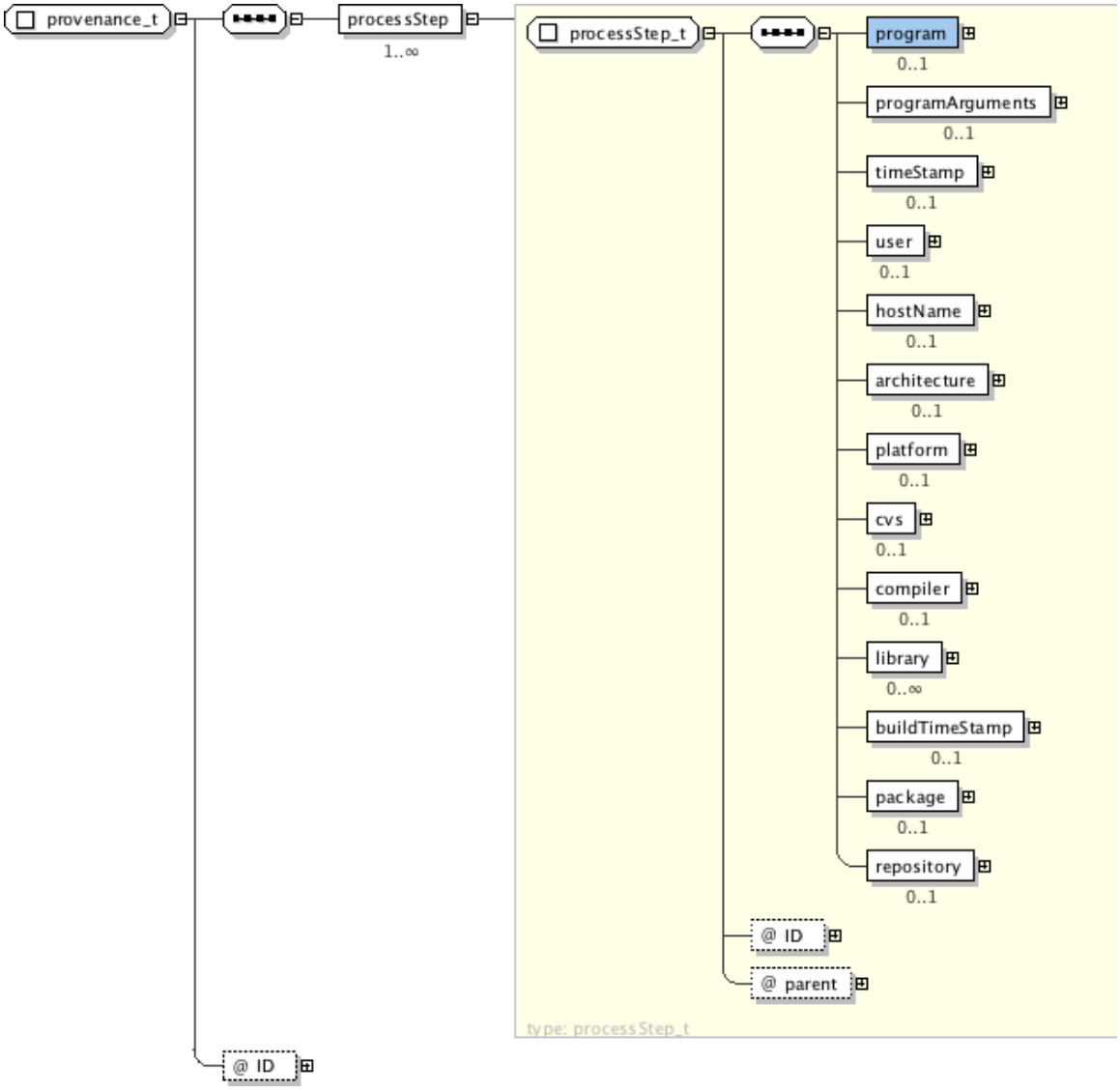


Figure 9.5. terminology_ag



Chapter 10. Terminology

10.1. Overview

The use of terminologies (and ontologies) are critical in the sharing of data. Ontologies specify the terms used to describe and represent areas of knowledge. They are commonly used by people, databases, and applications that have a requirement to share information about a subject. For example, the subject areas of medicine, real estate, and automobile repair all have specific ways of representing and communicating knowledge. Ontologies include computer-usable definitions of the basic terms describing the domain and the relationships between concepts. They encode knowledge within a domain and also knowledge that spans multiple domains. In this way, they make that knowledge reusable. For comprehensive information on ontologies and their use in biomedical science, please visit the National Center for Biomedical Ontology (NCBiO; http://www.bioontology.org/).

In the context of XCEDE, the use of terminologies is critical for data exchange. By specifying the exact terminological correspondences for data within XCEDE a user of the data is clear about what the data represents.

In order to accomplish this within XCEDE, two constructs are primarily used: terminology_ag and terminologyString_t. The terminology_ag attribute group defines the necessary information required to annotate a data item with its terminology correspondence. The terminologyString_t utilizes this attribute group and provides a basic type that can be utilized in the schema for data items that should have an associated terminology reference.

10.2. Examples

The use of terminologies is embedded within the XCEDE schema itself. For example, in regards to subject information, terminology strings are utilized to define the sex and species.

Figure 10.1. Terminology String Instance

```
<xs:complexType name="subjectInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t">
   <xs:sequence>
    <xs:element name="sex" type="terminologyString_t" minOccurs="0"/>
    <xs:element name="species" type="terminologyString_t" minOccurs="0"/>
    <xs:element name="birthdate" type="terminologyString_t" minOccurs="0"/>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

As an example of the use of terminology strings, in the context of a subject detailed above, the following example shows how sex and species are represented for a human female subject.

Figure 10.2. Terminology String Instance

```
 <sex preferredLabel="Female" abbreviation="F" nomenclature="UMLS"
 termID="C0015780" termPath="SNOMED CT Concept:Observable entity:General clinical
 state:Gender:Female">f</sex>
 <species preferredLabel="Human" abbreviation="human" nomenclature="BIRNLex"
 termID="birn_org_anat:birnOrgAnatomy_366"
 termPath="Thing:entity:continuant:independent_continuant:object:Biomaterial_object:Organismal_object:O
species
```

10.3. Reference

Figure 10.3. terminology_ag

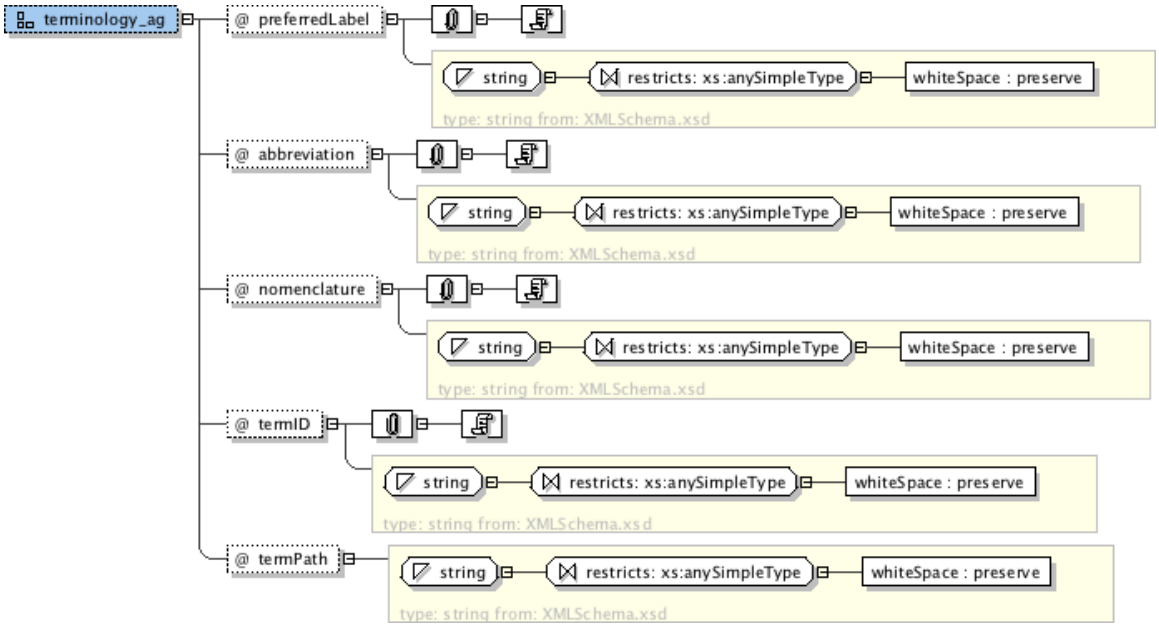
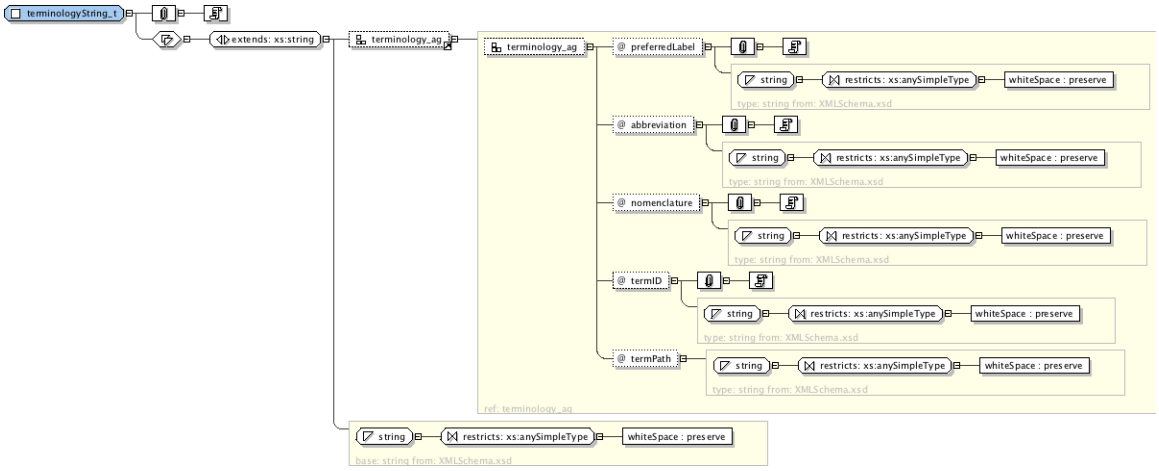


Figure 10.4. terminologyString_t



Part II. Using and Extending XCEDE 2

This section of the XCEDE 2.0 manual provides practical lessons on using the schema, as well as discussions on how to use W3C XML Schema to extend the core XCEDE 2 datatypes and how to integrate it into applications, databases and web services.

This part of the manual is not yet written.

Chapter 11. A Use Case

11.1. Overview

11.2. Examples

11.3. Reference

Chapter 12. Extending the Schema

12.1. Overview

12.2. Examples

12.3. Reference

Chapter 13. Integration With Applications and Services

13.1. Overview

13.2. Examples

13.3. Reference

Appendix A. XML Schema for XCEDE 2.0

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.xcede.org/xcede-2"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:sch="http://purl.oclc.org/dsdl/schematron"
 xmlns:gml="http://www.opengis.net/gml" targetNamespace="http://www.xcede.org/xcede-2"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:annotation>
  <xs:documentation>The XCEDE schema provides an extensive metadata hierarchy for
 describing and documenting research and clinical studies. The schema organizes
 information into five general hierarchical levels: a complete project; studies within
 a project; subjects involved in the studies; visits for each of the subjects; the full
 description of the subject's participation during each visit</xs:documentation>
  <xs:documentation>Each of these sub-schemas is composed of information relevant to
 that aspect of an experiment and can be stored in separate XML files or spliced into
 one large file allowing for the XML data to be stored in a hierarchical directory
 structure along with the primary data. Each sub-schema also allows for the storage
 of data provenance information allowing for a traceable record of processing and/or
 changes to the underlying data. Additionally, the sub-schemas contain support for
 derived statistical data in the form of human imaging activation maps and simple
 statistical value lists.</xs:documentation>
  <xs:documentation>XCEDE was originally designed in the context of neuroimaging
 studies and complements the Biomedical Informatics Research Network (BIRN) Human
 Imaging Database, an extensible database and intuitive web-based user interface for
 the management, discovery, retrieval, and analysis of clinical and brain imaging data.
 This close coupling allows for an interchangeable source-sink relationship between the
 database and the XML files, which can be used for the import/export of data to/from
 the database, the standardized transport and interchange of experimental data, the
 local storage of experimental information within data collections, and human and
 machine readable description of the actual data.</xs:documentation>
 </xs:annotation>
 <xs:element name="XCEDE">
  <xs:complexType>
   <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="annotationList" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"
 maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="revisionList" minOccurs="0">
     <xs:annotation>
      <xs:documentation>container for document revision history</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="revision" type="revision_t" minOccurs="0"
 maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="project" type="project_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="subject" type="subject_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="visit" type="visit_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="study" type="study_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="episode" type="episode_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="acquisition" type="acquisition_t" minOccurs="0"
 maxOccurs="unbounded"/>
    <xs:element name="catalog" type="catalog_t" minOccurs="0" maxOccurs="unbounded"/>
```

```
   <xs:element name="analysis" type="analysis_t" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="resource" type="resource_t" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="protocol" type="protocol_t" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="data" type="abstract_data_t" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:choice>
  <xs:attribute name="version" type="xs:string"/>
 </xs:complexType>
</xs:element>
<!--*********           Top-level containers           *********-->
<xs:complexType name="project_t">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="projectInfo" type="projectInfo_t" minOccurs="0"/>
    <xs:element name="contributorList" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="contributor" type="person_t" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="subjectGroup_t">
 <xs:annotation>
  <xs:documentation>There should be one of these elements for each subject group in
this project (e.g. control, patient).</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="subjectID" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="ID"/>
 <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
<xs:complexType name="subject_t">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="subjectInfo" type="subjectInfo_t" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="visit_t">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="visitInfo" type="visitInfo_t" minOccurs="0"/>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attributeGroup ref="visitExternalIDs_ag"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="study_t">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="studyInfo" type="studyInfo_t" minOccurs="0"/>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
```

```
   <xs:attributeGroup ref="studyExternalIDs_ag"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="episode_t">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="episodeInfo" type="episodeInfo_t" minOccurs="0"/>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attributeGroup ref="episodeExternalIDs_ag"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="acquisition_t">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="acquisitionInfo" type="acquisitionInfo_t" minOccurs="0"/>
    <xs:choice minOccurs="0">
     <xs:element name="dataResourceRef" type="ref_t">
      <xs:annotation>
       <xs:documentation>A reference to a resource as described above. The resource
could be part of a catalog, a root level resource, etc.</xs:documentation>
      </xs:annotation>
     </xs:element>
     <xs:element name="dataRef" type="ref_t">
      <xs:annotation>
       <xs:documentation>A refrence to a container that the actual acquisition data
goes into (as opposed to being in an external non-XCEDE format)</xs:documentation>
      </xs:annotation>
     </xs:element>
    </xs:choice>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="acquisitionProtocol" use="optional"/>
   <xs:attributeGroup ref="acquisitionExternalIDs_ag"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="analysis_t">
 <xs:annotation>
  <xs:documentation>A collection of output from ana analysis of
data.</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="provenance" type="provenance_t" minOccurs="0"
maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>The record of the origin and transformations applied to source
data that produced this analysis</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="inputRef" type="ref_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="outputRef" type="ref_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="measurementGroup" type="measurementGroup_t"
maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>A measurementGroup contains information and data related
to the outcome of an analysis.  For example, this could be a statistic (e.g.
```

```
from a Statistical Parametric Map) or a measurement (e.g. the volume of the
hippocmapus).</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="observation_t">
 <xs:annotation>
  <xs:documentation>Observations that are made concerning an entity (e.g. in a
measurementGroup).  For example, following hte hippocmapus example, might be the total
volume, surface area, etc.</xs:documentation>
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="name" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>The name (preferred label) of the observation
value</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="units" type="xs:string" use="optional">
    <xs:annotation>
     <xs:documentation>The measurement units for the returned value of the
observation</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="type" type="valueTypes_t" use="optional">
    <xs:annotation>
     <xs:documentation>The declared data type for the returned value of the
observation</xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="protocol_t">
 <xs:complexContent>
  <xs:extension base="abstract_protocol_t">
   <xs:sequence>
    <xs:element name="steps" minOccurs="0">
     <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element name="step" type="protocol_t"/>
       <xs:element name="stepRef" type="ref_t"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="items" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="item" type="protocolItem_t" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="catalog_t">
 <xs:complexContent>
  <xs:extension base="abstract_tagged_entity_t">
   <xs:sequence>
    <xs:element name="catalogList" minOccurs="0">
     <xs:complexType>
```

```
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element name="catalog" type="catalog_t"/>
       <xs:element name="catalogRef">
        <xs:complexType>
         <xs:attribute name="catalogID" type="xs:string"/>
        </xs:complexType>
       </xs:element>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="entryList" minOccurs="0">
     <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element name="entry" type="resource_t"/>
       <xs:element name="entryDataRef" type="ref_t">
        <xs:annotation>
         <xs:documentation>A reference to a data element at the top level of an XCEDE
document</xs:documentation>
        </xs:annotation>
       </xs:element>
       <xs:element name="entryResourceRef" type="ref_t">
        <xs:annotation>
         <xs:documentation>A reference to a resource element at the top level of an
XCEDE document</xs:documentation>
        </xs:annotation>
       </xs:element>
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attributeGroup ref="ID_name_description"/>
   <xs:attributeGroup ref="levelRef_ag"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="resource_t">
 <xs:annotation>
  <xs:documentation>A resource is something that we haven't agreed on
  yet.</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="abstract_tagged_entity_t">
   <xs:sequence>
    <xs:element name="uri" type="frag_uri_t" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attributeGroup ref="ID_name_description"/>
   <xs:attributeGroup ref="levelRef_ag"/>
   <xs:attribute name="format" type="xs:string" use="optional">
    <xs:annotation>
     <xs:documentation>Format of file. E.g. DICOM, Analyze, 4dfp</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="content" type="xs:string" use="optional">
    <xs:annotation>
     <xs:documentation>Code indicating the contents of the image. E.g. GFC,
T88</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="cachePath" use="optional">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:maxLength value="255"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:extension>
 </xs:complexContent>
```

```
</xs:complexType>
<!--*********      Top-level ID attribute groups      *********-->
<xs:attributeGroup name="visitExternalIDs_ag">
 <xs:attribute name="projectID" type="xs:string">
  <xs:annotation>
   <xs:documentation>The content of this attribute should match the ID attribute of
the the project_t to which this element is associated.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="projectURI" type="xs:string">
  <xs:annotation>
   <xs:documentation>This is the location of a document where to find an element
matching the projectID (see projectID attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="subjectID" type="xs:string">
  <xs:annotation>
   <xs:documentation>The content of this attribute should match the ID attribute of
the the subject_t to which this element is associated.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="subjectURI" type="xs:string">
  <xs:annotation>
   <xs:documentation>This is the location of a document where to find an element
matching the subjectID (see subjectID attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="subjectGroupID" type="xs:string">
  <xs:annotation>
   <xs:documentation>content of this attribute should match the ID of one of the
subject groups listed in the project_t associated with this element (see projectID and
project_t).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:attributeGroup>
<xs:attributeGroup name="studyExternalIDs_ag">
 <xs:attributeGroup ref="visitExternalIDs_ag"/>
 <xs:attribute name="visitID" type="xs:string">
  <xs:annotation>
   <xs:documentation>The content of this attribute should match the ID attribute of
the the visit_t to which this element is associated.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="visitURI" type="xs:string">
  <xs:annotation>
   <xs:documentation>This is the location of a document where to find an element
matching the visitID (see visitID attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:attributeGroup>
<xs:attributeGroup name="episodeExternalIDs_ag">
 <xs:attributeGroup ref="studyExternalIDs_ag"/>
 <xs:attribute name="studyID" type="xs:string">
  <xs:annotation>
   <xs:documentation>The content of this attribute should match the ID attribute of
the the study_t to which this element is associated.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="studyURI" type="xs:string">
  <xs:annotation>
   <xs:documentation>This is the location of a document where to find an element
matching the studyID (see studyID attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:attributeGroup>
<xs:attributeGroup name="acquisitionExternalIDs_ag">
 <xs:attributeGroup ref="episodeExternalIDs_ag"/>
```

```
 <xs:attribute name="episodeID" type="xs:string">
  <xs:annotation>
   <xs:documentation>The content of this attribute should match the ID attribute of
the episode_t to which this element is associated.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="episodeURI" type="xs:string">
  <xs:annotation>
   <xs:documentation>This is the location of a document where to find an element
matching the episodeID (see episodeID attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:attributeGroup>
<xs:attributeGroup name="allLevelExternalIDs_ag">
 <xs:attributeGroup ref="acquisitionExternalIDs_ag"/>
 <xs:attribute name="acquisitionID" type="xs:string">
  <xs:annotation>
   <xs:documentation>The content of this attribute should match the ID attribute of
the acquisition_t to which this element is associated.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="acquisitionURI" type="xs:string">
  <xs:annotation>
   <xs:documentation>This is the location of a document where to find an element
matching the acquisitionID (see acquisitionID attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:attributeGroup>
<xs:attributeGroup name="levelRef_ag">
 <xs:annotation>
  <xs:documentation>Reference to the meta-data element that this catalog is related
to. Should point to one of the core xcede hierarchy components: project, subject,
visit, study, acquisition.</xs:documentation>
 </xs:annotation>
 <xs:attribute name="level" type="levelDescriptor_t"/>
 <xs:attributeGroup ref="allLevelExternalIDs_ag"/>
</xs:attributeGroup>
<!--*********            Abstract types            *********-->
<xs:complexType name="abstract_data_t" abstract="true">
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:attributeGroup ref="levelRef_ag"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="abstract_container_t" abstract="true">
 <xs:sequence>
  <xs:element name="commentList" minOccurs="0">
   <xs:annotation>
    <xs:documentation>A comment is a description of the entity refered to by
the current element.  For example, a description of a particular subject or
series.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
    <xs:sequence>
     <xs:element name="comment" type="authoredText_t" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="annotationList" minOccurs="0">
   <xs:annotation>
    <xs:documentation>An annotation is a description related to the xml document and
the current element.  Fore example, some processing that ahs edited content in the
element itself.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
```

```
    <xs:sequence>
     <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="resourceList" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Informational resources related to the
container</xs:documentation>
   </xs:annotation>
   <xs:complexType>
    <xs:sequence>
     <xs:element name="resource" type="informationResource_t" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="ID" type="xs:string"/>
 <xs:attribute name="rev" type="xs:string">
  <xs:annotation>
   <xs:documentation>Revision number, should correspond with an appropriate revision
ID in the XCEDE/history element</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="type" type="xs:string">
  <xs:annotation>
   <xs:documentation>Attribute for creating categories within a container set. For
example, within study, types might include 'PET' or 'MR'. One could also create
sub-classes using colon notation: "MR:STRUCT"</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attributeGroup ref="terminology_ag"/>
</xs:complexType>
<xs:complexType name="abstract_entity_t" abstract="true">
 <xs:annotation>
  <xs:documentation>entity elements are used to describe the resultant scope of an
analysis (e.g. anatomical entity, atlas entity)</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="description" type="xs:string" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="ID" type="xs:string"/>
 <xs:attribute name="preferredEntityLabel" type="xs:string" use="required">
  <xs:annotation>
   <xs:documentation>Preferred Name for the Entity</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:complexType>
<xs:complexType name="abstract_info_t" abstract="true">
 <xs:annotation>
  <xs:documentation>info elements are present in each of the hierarchy levels. these
can be extended to capture instance specific content (following recommendation
  5)</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="description" type="xs:string" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="abstract_protocol_t" abstract="true">
 <xs:sequence>
  <xs:element name="protocolOffset" type="protocolOffset_t" minOccurs="0"
maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attributeGroup ref="ID_name_description"/>
 <xs:attributeGroup ref="terminology_ag"/>
```

```
 <xs:attribute name="level" type="levelDescriptor_t">
  <xs:annotation>
   <xs:documentation>Describes the level of the XCEDE hierarchy that this protocol
    instance should be validated against</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="required" type="xs:boolean"/>
 <xs:attribute name="minOccurences" type="xs:integer"/>
 <xs:attribute name="maxOccurences" type="xs:integer">
  <xs:annotation>
   <xs:documentation>Are these occurences within a step (i.e. during a single time
point)? How to refer to repeats across steps?</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="minTimeFromStart" type="xs:string">
  <xs:annotation>
   <xs:documentation>Absolute time from start of overall protocol</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="maxTimeFromStart" type="xs:string"/>
</xs:complexType>
<!--*********            Level information types            *********-->
<xs:complexType name="projectInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t">
   <xs:sequence>
    <xs:element name="exptDesignList" minOccurs="0">
     <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element name="exptDesign"/>
       <xs:element name="exptDesignRef" type="ref_t"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="subjectGroupList" minOccurs="0">
     <xs:annotation>
      <xs:documentation>This provides a mapping of subjects to subject groups
within projects. A subject can be a member of subject groups in multiple
projects.</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="subjectGroup" type="subjectGroup_t" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
     </xs:complexType>
    </xs:element>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="subjectInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t">
   <xs:sequence>
    <xs:element name="sex" type="terminologyString_t" minOccurs="0"/>
    <xs:element name="species" type="terminologyString_t" minOccurs="0"/>
    <xs:element name="birthdate" type="terminologyString_t" minOccurs="0"/>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="studyInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="visitInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t">
   <xs:sequence>
    <xs:element name="timeStamp" type="xs:dateTime" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="episodeInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="acquisitionInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t"/>
 </xs:complexContent>
</xs:complexType>
<!--*********            Resource types            *********-->
<xs:complexType name="informationResource_t">
 <xs:complexContent>
  <xs:extension base="resource_t"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="dcResource_t">
 <xs:complexContent>
  <xs:extension base="informationResource_t">
   <xs:sequence>
    <xs:element name="title" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="creator" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="subject" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="publisher" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="contributor" type="orderedString_t" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="date" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="type" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="format" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="identifier" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="source" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="language" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="relation" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="coverage" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="rights" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="dataResource_t">
 <xs:complexContent>
  <xs:extension base="resource_t">
   <xs:sequence>
    <xs:element name="provenance" type="provenance_t" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="binaryDataResource_t">
```

```
 <xs:annotation>
  <xs:documentation>The parent type (abstract_resource_t) can describe a stream of
data. The extensions in this derived type (binaryDataResource_t) tell you that this
data stream is composed of a (one-dimensional) sequence of data items of a given data
type and byte order.</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="dataResource_t">
   <xs:sequence>
    <xs:element name="elementType" minOccurs="0">
     <xs:annotation>
      <xs:documentation>This element describes the type of individual data elements in
the data record. For numeric data types, this indicates whether the element type is
a signed integer ("int"), unsigned integer ("uint"), or floating-point ("float"), as
well as the number of bits allocated to each element.</xs:documentation>
     </xs:annotation>
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="int8"/>
       <xs:enumeration value="uint8"/>
       <xs:enumeration value="int16"/>
       <xs:enumeration value="uint16"/>
       <xs:enumeration value="int32"/>
       <xs:enumeration value="uint32"/>
       <xs:enumeration value="int64"/>
       <xs:enumeration value="uint64"/>
       <xs:enumeration value="float32"/>
       <xs:enumeration value="float64"/>
       <xs:enumeration value="ascii"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="byteOrder" minOccurs="0">
     <xs:annotation>
      <xs:documentation>This element describes whether the individual data elements
in the data record are stored with the most-significant-byte first (msbfirst) or
least-significant-byte first (lsbfirst). This element is required if the the data type
given by the "elementtype" element has a size larger than one byte.</xs:documentation>
     </xs:annotation>
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="lsbfirst"/>
       <xs:enumeration value="msbfirst"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="compression" type="xs:string" minOccurs="0">
     <xs:annotation>
      <xs:documentation>If this element is present, the files pointed to by the uri
elements are compressed data files. The only compression method specifically named by
this specification is "gzip". As a special case for binaryDataResource_t and derived
types, files compressed with gzip and containing a .gz suffix can be referenced in
the uri element without the suffix. If a file pointed to by the URI does not exist,
the application should search for the same file with the .gz suffix appended -- if it
exists, use that file and act as if the compression element had been specified with
the value "gzip". This allows the referenced files to be compressed or uncompressed
at will (as long as the .gz suffix is appropriately added/removed from the filename),
without needing to change the URI's in this element.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="dimensionedBinaryDataResource_t">
 <xs:annotation>
```

```
  <xs:documentation xml:lang="en">This type adds multi-dimensionality to the
(uni-dimensional) data stream represented by
  binaryDataResource_t.</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="binaryDataResource_t">
   <xs:sequence>
    <xs:element name="dimension" type="binaryDataDimension_t" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="mappedBinaryDataResource_t">
 <xs:annotation>
  <xs:documentation xml:lang="en">This type places the multi-dimensional
data array (say a 3-dimensional cube) into a coordinate space (say MR scanner
coordinates).</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="binaryDataResource_t">
   <xs:sequence>
    <xs:element name="dimension" type="mappedBinaryDataDimension_t"
maxOccurs="unbounded"/>
    <xs:element name="originCoords" type="xs:string" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">This is a coordinate tuple giving the location
of the first item in the data. For example, if this is an MR volume, this could
be a triple giving the location in RAS coordinates of the first voxel in the
data.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="binaryDataDimension_t">
 <xs:annotation>
  <xs:documentation xml:lang="en">This element stores information about one of the
N dimensions in the data record. Multiple instances of this element are ordered from
fastest-moving to slowest-moving. These elements provide information to describe the
size (in data elements) of the N-dimensional bounding box for the data, and in some
cases to describe the mapping of indexes within this bounding box to 'real-world'
coordinates.</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="size" type="xs:int">
   <xs:annotation>
    <xs:documentation xml:lang="en">The number of elements in the data along one
traversal of this dimension.</xs:documentation>
   </xs:annotation>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="label" type="xs:string">
  <xs:annotation>
   <xs:documentation xml:lang="en">This is a label for the dimension. The first three
spatial dimensions (or however many exist) must be labeled, in order, 'x', 'y', and
'z'. The first temporal dimension must be labeled 't'.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="splitRank" type="xs:string">
  <xs:annotation>
   <xs:documentation>If this attribute exists, this dimension is a "split" dimension,
and this dimension must be "merged" with one or more other dimensions (with the same
label) before presenting the data to the application. This is useful, for example,
if the data is stored in Siemens' Mosaic DICOM format, where slices of a 3-D volume
are arranged to look like they are tiled onto a square 2-D area. In this case, what
would normally be called the 'z' dimension has two forks, one that occurs before the
```

```
'y' dimension (the first row in the data covers the first row of several slices),
and one that occurs after the 'y' dimension. If, as in this case, there are two or
more dimensions that should be merged into one, both component dimensions should have
the label 'z', but have splitRank attributes "1" and "2", which specifies the order
in which all 'split' dimensions of the same label will be merged. After merging, the
resultant 'z' dimension element should contain the same children of the highest-ranked
split 'z' dimension, except for the 'size' element, which will be the product of the
sizes of all 'z' split dimensions. The position of the resultant dimension should be
the position of the highest-ranked 'z' split dimension. The data itself should also be
reordered to reflect the new dimension structure.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="outputSelect" type="xs:string">
  <xs:annotation>
   <xs:documentation xml:lang="en">In the same way that the 'splitRank' attribute
allows you to specify dimensions that should be merged before presenting the data to
an application, this attribute specifies a data filter along this dimension. If this
attribute exists, it should contain a whitespace-separated list of indices (indexed
starting at 0). Only data points along this dimension that occur in the index list
should be presented to the application. Likewise, the 'size' of the dimension, after
selection, should be updated to reflect the new size of this dimension (which should
be the number of indices in the content of this attribute).</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:complexType>
<xs:complexType name="mappedBinaryDataDimension_t">
 <xs:complexContent>
  <xs:extension base="binaryDataDimension_t">
   <xs:sequence>
    <xs:element name="origin" type="xs:float" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">A value assigned to the first data element
along this dimension. For example, if this dimension corresponds to "time",
this element could store the time corresponding to the first data element.
If this is a two-dimensional projection of the surface of the Earth, and this
dimension takes you around the Earth parallel to the equator, this value could
be the degrees longitude. For MRI data, this is the single coordinate on the
Left-to-Right, Posterior-to-Anterior, or Inferior-to-Superior axis to which this
dimension most closely matches (see 'direction' element and 'rasOrigin' element in
'mrImageDataResource_t').</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="spacing" type="xs:float" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">This is the average distance between consecutive
data elements in this dimension. If the spacing is not regular, then it may be
possible to calculate the actual distance between any two data elements in this
dimension using the 'datapoints' element.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="gap" type="xs:float" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">This is the length of the unsampled space
between consecutive data elements in this dimension, i.e. the distance between the end
of one data element and the beginning of the next. For MRI data, this can be used to
specify the gap between two collected slices &#x02015; the actual width of each slice
can be calculated as 'spacing' minus 'gap'.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="datapoints" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">The content of this element is either (1) a
whitespace-separated list of values, or (2) a list of 'value' elements, that can be
used as a label for each data point along this dimension. The values can be numbers
representing points on an axis (this is the typical case), text strings, coordinate
tuples, etc. Any datapoint label that includes whitespace (coordinate tuples included)
must be encapsulated within a child 'value' element. If this element is missing, it
```

```
is assumed that labels can be calculated using information in other fields (such as
'origin', 'spacing', etc.). This element is particularly useful for dimensions with
irregular spacing.</xs:documentation>
     </xs:annotation>
     <xs:complexType mixed="true">
      <xs:sequence>
       <xs:element name="value" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="direction" type="listoffloats_t" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">This element contains a vector (represented as
a whitespace-separated list of floating-point values in the appropriate coordinate
system) that is parallel to this dimension's edge of the bounding box. The vector
starts at the first element in the data and points towards subsequent elements along
this dimension. For MRI data, this should be a unit vector in (R,A,S) coordinates
(positive values are Right, Anterior, or Superior respectively) &#x02015; for 'x' and
'y' dimensions, this corresponds to the two vectors in the ImagePatientOrientation
field in DICOM.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="units" type="xs:string" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">This stores the units used for all
       numeric values in this dimension element. In MRI data, this should
       be 'mm' for all spatial dimensions ('x', 'y', 'z') and 'ms' for the
       temporal dimension 't'.</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="measurementframe" minOccurs="0">
     <xs:annotation>
      <xs:documentation xml:lang="en">The mapping (if any) between the values
expressed in &lt;datapoints&gt; and the coordinate system used by this datarec. For
example, in DTI data, this is useful for mapping gradient direction vectors to the RAS
coordinate space used in the &lt;direction&gt; vectors.</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="vector" type="listoffloats_t" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="frag_uri_t">
 <xs:annotation>
  <xs:documentation>The external data pointed to by this uri is a "fragment", where a
"fragment" is defined as a stream of data contiguously stored in the same file offset
by 'offset' bytes and of 'size' bytes.</xs:documentation>
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:anyURI">
   <xs:attribute name="offset" type="xs:unsignedLong">
    <xs:annotation>
     <xs:documentation>The data for this fragment will start at this byte position in
the resource specified by the 'uri' element. If this attribute does not exist or is
empty, it is assumed to be zero.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="size" type="xs:unsignedLong">
    <xs:annotation>
     <xs:documentation>This specifies the size of this block (in bytes) in the
      resource specified by the 'uri' element. If this attribute does not
```

```
      exist or is empty, it is calculated using the dimension and elementtype
      element.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="listoffloats_t">
 <xs:list itemType="xs:float"/>
</xs:simpleType>
<xs:complexType name="format_t">
 <xs:annotation>
  <xs:documentation>Container for describing imaging formats and file name extensions
   (currently underimplemented)</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="description" type="xs:string" minOccurs="0"/>
  <xs:element name="documentationList" minOccurs="0">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="documentation" type="informationResource_t" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="extensionList" minOccurs="0">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="extension" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="name"/>
</xs:complexType>
<!--*********            Provenance types            *********-->
<xs:complexType name="processStep_t">
 <xs:sequence>
  <xs:element name="program" type="versionedProgramEntity_t" minOccurs="0"/>
  <xs:element name="programArguments" type="argumentsType_t" minOccurs="0"/>
  <xs:element name="timeStamp" type="xs:dateTime" minOccurs="0"/>
  <xs:element name="user" type="xs:string" minOccurs="0"/>
  <xs:element name="hostName" type="xs:dateTime" minOccurs="0"/>
  <xs:element name="architecture" type="xs:string" minOccurs="0"/>
  <xs:element name="platform" type="versionedEntity_t" minOccurs="0"/>
  <xs:element name="cvs" type="xs:string" minOccurs="0"/>
  <xs:element name="compiler" type="versionedEntity_t" minOccurs="0"/>
  <xs:element name="library" type="versionedEntity_t" minOccurs="0"
maxOccurs="unbounded"/>
  <xs:element name="buildTimeStamp" type="xs:dateTime" minOccurs="0"/>
  <xs:element name="package" type="versionedEntity_t" minOccurs="0"/>
  <xs:element name="repository" type="xs:string" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="ID" type="xs:string"/>
 <xs:attribute name="parent" type="xs:string"/>
</xs:complexType>
<xs:complexType name="provenance_t">
 <xs:sequence>
  <xs:element name="processStep" type="processStep_t" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="ID" type="xs:string"/>
</xs:complexType>
<xs:complexType name="argumentsType_t">
 <xs:annotation>
  <xs:documentation>input and output arguments of the processing
tool</xs:documentation>
```

```
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="inputs" type="xs:string"/>
   <xs:attribute name="outputs" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="versionedEntity_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="version" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="versionedProgramEntity_t">
 <xs:annotation>
  <xs:documentation>version and build type of the processing tool</xs:documentation>
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="version" type="xs:string"/>
   <xs:attribute name="build" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<!--*********           Event types            *********-->
<xs:complexType name="events_t">
 <xs:complexContent>
  <xs:extension base="abstract_data_t">
   <xs:sequence>
    <xs:element name="params" type="eventParams_t" minOccurs="0"/>
    <xs:element name="event" type="event_t" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string" minOccurs="0"/>
    <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"
maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="event_t">
 <xs:annotation>
  <xs:documentation>This element represents an interval of time, with arbitrary
metadata
   (in the value element).</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="onset" type="xs:float" minOccurs="0"/>
  <xs:element name="duration" type="xs:float" minOccurs="0"/>
  <xs:element name="value" type="eventValue_t" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"
maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="type" type="xs:string" use="optional"/>
 <xs:attribute name="units" type="xs:string" use="optional">
  <xs:annotation>
   <xs:documentation>This attribute is optional, but an group using this schema should
    agree on, use, and enforce measurement units consistently, to avoid the need for
    unit conversion in an application.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:complexType>
<xs:complexType name="eventValue_t">
 <xs:annotation>
  <xs:documentation>User-specified metadata associated with an
event.</xs:documentation>
```

```
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="name" type="xs:string"/>
   <xs:anyAttribute processContents="lax"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="eventParams_t">
 <xs:annotation>
  <xs:documentation>These value elements apply to all events in the parent event
  list.</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="value" type="eventValue_t" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="abstract_tagged_entity_t">
 <xs:sequence>
  <xs:element name="metaFields" minOccurs="0">
   <xs:complexType>
    <xs:sequence minOccurs="0">
     <xs:element name="metaField" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
       <xs:simpleContent>
        <xs:extension base="xs:string">
         <xs:attribute name="name" type="xs:string"/>
        </xs:extension>
       </xs:simpleContent>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:annotation>
 <xs:documentation>*********            Protocol types
*********</xs:documentation>
</xs:annotation>
<xs:complexType name="protocolItem_t">
 <xs:sequence>
  <xs:element name="itemText" minOccurs="0">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="textLabel" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
       <xs:attribute name="location" use="required"/>
       <xs:attribute name="value" type="xs:string" use="required"/>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:choice>
   <xs:element name="itemRange" type="protocolItemRange_t" minOccurs="0"
maxOccurs="unbounded"/>
   <xs:element name="itemChoice" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:attribute name="itemCode" type="xs:string"/>
     <xs:attribute name="itemValue" type="xs:string"/>
     <xs:attribute name="ID" type="xs:string"/>
     <xs:attribute name="units" type="xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:sequence>
```

```
 <xs:attribute name="name" type="xs:string"/>
 <xs:attribute name="ID" type="xs:string"/>
 <xs:attribute name="required" type="xs:boolean"/>
</xs:complexType>
<xs:complexType name="protocolOffset_t">
 <xs:sequence>
  <xs:element name="protocolTimeRef" type="unitString_t" minOccurs="0"/>
  <xs:element name="preferedTimeOffset" type="unitString_t" minOccurs="0"/>
  <xs:element name="minTimeOffset" type="unitString_t" minOccurs="0"/>
  <xs:element name="maxTimeOffset" type="unitString_t" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="protocolItemChoice_t">
 <xs:sequence>
  <xs:element name="value" type="xs:string" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="units" type="xs:string"/>
</xs:complexType>
<xs:complexType name="protocolItemRange_t">
 <xs:attribute name="min" type="xs:string">
  <xs:annotation>
   <xs:documentation>Minimum value for item (inclusive)</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="max" type="xs:string">
  <xs:annotation>
   <xs:documentation>Maximum value for item (inclusive)</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="units" type="xs:string"/>
</xs:complexType>
<xs:annotation>
 <xs:documentation>*********            Assessment types
*********</xs:documentation>
</xs:annotation>
<xs:complexType name="assessmentInfo_t">
 <xs:complexContent>
  <xs:extension base="abstract_info_t"/>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="assessment_t">
 <xs:complexContent>
  <xs:extension base="abstract_data_t">
   <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="dataInstance" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="assessmentInfo" type="assessmentInfo_t" minOccurs="0">
        <xs:annotation>
         <xs:documentation>Block for describing things like
          informant, clinical rater, assessment date,
         etc</xs:documentation>
        </xs:annotation>
       </xs:element>
       <xs:element name="assessmentItem" type="assessmentItem_t" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="validated" type="xs:boolean">
       <xs:annotation>
        <xs:documentation>Indicates whether the instance has been
         validated (e.g. by reconciling double-entry instances).
         There should be only one validated instance per
        assessment.</xs:documentation>
       </xs:annotation>
      </xs:attribute>
     </xs:complexType>
```

```
    </xs:element>
    <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"
maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="assessmentDescItem_t">
 <xs:complexContent>
  <xs:extension base="protocolItem_t">
   <xs:attribute name="version" type="xs:string"/>
   <xs:attribute name="formRef" type="xs:string"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="assessmentItem_t">
 <xs:sequence>
  <xs:element name="valueStatus" type="xs:string" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Information on the status of a value (e.g. subject refused to
     answer)</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="value" type="value_t">
   <xs:annotation>
    <xs:documentation>Actual value of the assessment item as recorded on the
    form</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="normValue" type="value_t" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Normalized or scaled value of the assessment
    item</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="reconciliationNote" type="textAnnotation_t" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Normalized or scaled value of the assessment
    item</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"/>
 </xs:sequence>
 <xs:attributeGroup ref="terminology_ag"/>
 <xs:attribute name="ID"/>
 <xs:attribute name="name"/>
</xs:complexType>
<xs:annotation>
 <xs:documentation>*********            Analysis types
*********</xs:documentation>
</xs:annotation>
<xs:complexType name="analysisRef_t">
 <xs:attributeGroup ref="levelRef_ag"/>
</xs:complexType>
<xs:complexType name="measurementGroup_t">
 <xs:annotation>
  <xs:documentation>A measurementGroup contains information and data related
to the outcome of an analysis.  For example, this could be a statistic (e.g.
from a Statistical Parametric Map) or a measurement (e.g. the volume of the
hippocmapus).</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="abstract_container_t">
   <xs:sequence>
    <xs:element name="entity" type="abstract_entity_t" maxOccurs="unbounded">
     <xs:annotation>
```

```
      <xs:documentation>In the context of a measurement Group, is the entity being
measured.  For example, the hippocampus. </xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="observation" type="observation_t" maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>In the context of a measurement Group, are the observations
that are made concerning the entity.  For example, following hte hippocmapus example,
might be the total volume, surface area, etc.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:annotation>
 <xs:documentation>*********           Terminology types
*********</xs:documentation>
</xs:annotation>
<xs:attributeGroup name="terminology_ag">
 <xs:attribute name="preferredLabel" type="xs:string" use="required">
  <xs:annotation>
   <xs:documentation>The preferred label of this term. This can be different than the
term used by the applications (e.g. CBLM versus cerebellum)</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="abbreviation" type="xs:string" use="optional">
  <xs:annotation>
   <xs:documentation>The preferred abbreviation for the term</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="nomenclature" type="xs:string" use="required">
  <xs:annotation>
   <xs:documentation>The source ontology of this terminology/ontology
term</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="termID" type="xs:string" use="required">
  <xs:annotation>
   <xs:documentation>Applications will likely want to constrain what are valid IDs
    within the context of their application (for example, allowing only
   LSID's)</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="termPath" type="xs:string" use="optional"/>
</xs:attributeGroup>
<xs:complexType name="terminologyString_t">
 <xs:annotation>
  <xs:documentation>A simple representation of terms for use within the
schema.</xs:documentation>
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attributeGroup ref="terminology_ag"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="nsTermAnnotation_t">
 <xs:complexContent>
  <xs:extension base="abstract_annotation_t">
   <xs:sequence>
    <xs:element name="ontologyClass" type="xs:string" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="nsOntologyAnnotation_t">
```

```
 <xs:complexContent>
  <xs:extension base="abstract_annotation_t">
   <xs:sequence>
    <xs:element name="term" type="xs:string" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="nomenclature_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="abbreviation" type="xs:string"/>
   <xs:attribute name="nomenclature" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="atlasEntity_t">
 <xs:annotation>
  <xs:documentation>entity elements are used to describe the resultant scope of an
analysis</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="abstract_entity_t">
   <xs:sequence>
    <xs:element name="geometry" type="abstract_geometry_t">
     <xs:annotation>
      <xs:documentation>Atlas entity defined through the use of GML base
geometries.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="abstract_geometry_t" abstract="true"/>
<xs:complexType name="anatomicalEntity_t">
 <xs:annotation>
  <xs:documentation>Entity elements are used to describe the resultant scope of an
analysis</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="abstract_entity_t">
   <xs:sequence>
    <xs:element name="label" type="terminologyString_t" minOccurs="0"
maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>The set of terms that name this entity.  This set can contain
a collection of synonymous terms</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="laterality" type="laterality_t" use="optional">
    <xs:annotation>
     <xs:documentation>Laterality of region (or none) </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="tissueType" type="tissueType_t" use="optional">
    <xs:annotation>
     <xs:documentation>Tissue type of region (or none) </xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:annotation>
 <xs:documentation>*********            Misc. types
*********</xs:documentation>
```

```
</xs:annotation>
<xs:complexType name="nameValue_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="name" type="xs:string"/>
   <xs:anyAttribute processContents="lax"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="metadataList_t">
 <xs:sequence>
  <xs:element name="value" type="nameValue_t" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ref_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="ID" type="xs:string"/>
   <xs:attribute name="URI" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="authoredText_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attributeGroup ref="authoredText_ag"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:attributeGroup name="authoredText_ag">
 <xs:attribute name="author" type="xs:string"/>
 <xs:attribute name="timestamp" type="xs:dateTime"/>
</xs:attributeGroup>
<xs:complexType name="abstract_annotation_t" abstract="true">
 <xs:attributeGroup ref="authoredText_ag"/>
</xs:complexType>
<xs:complexType name="textAnnotation_t">
 <xs:complexContent>
  <xs:extension base="abstract_annotation_t">
   <xs:sequence>
    <xs:element name="text" type="xs:string" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="generator_t">
 <xs:sequence>
  <xs:element name="application" type="versionedEntity_t">
   <xs:annotation>
    <xs:documentation>Program used to generate document</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="invocation" type="xs:string">
   <xs:annotation>
    <xs:documentation>Application input required to generate this document. Should be
explicit such that this document can be re-generated from this info</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="dataSource" type="xs:string" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Description of data source with version numbers and/or timestamp
of data</xs:documentation>
   </xs:annotation>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="person_t">
```

```
 <xs:annotation>
  <xs:documentation>Add additional fields (address, email, etc)</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="salutation" type="xs:string" minOccurs="0">
   <xs:annotation>
    <xs:documentation>e.g. Dr., Mr., Mrs.</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="givenName" type="xs:string" minOccurs="0"/>
  <xs:element name="middleName" type="xs:string" minOccurs="0"/>
  <xs:element name="surname" type="xs:string" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Used for last name or only name (e.g.
    Prince)</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="academicTitles" type="xs:string" minOccurs="0"/>
  <xs:element name="institution" type="xs:string" minOccurs="0"/>
  <xs:element name="department" type="xs:string" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="ID" type="xs:string"/>
 <xs:attribute name="role" type="xs:string"/>
</xs:complexType>
<xs:simpleType name="levelDescriptor_t">
 <xs:restriction base="xs:string">
  <xs:enumeration value="project"/>
  <xs:enumeration value="subject"/>
  <xs:enumeration value="visit"/>
  <xs:enumeration value="study"/>
  <xs:enumeration value="episode"/>
  <xs:enumeration value="acquisition"/>
 </xs:restriction>
</xs:simpleType>
<xs:attributeGroup name="ID_name_description">
 <xs:attribute name="ID" type="xs:string"/>
 <xs:attribute name="name" type="xs:string"/>
 <xs:attribute name="description" type="xs:string"/>
</xs:attributeGroup>
<xs:complexType name="unitString_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="units" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="revision_t">
 <xs:sequence>
  <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
  <xs:element name="generator" type="generator_t" minOccurs="0"/>
  <xs:element name="annotation" type="textAnnotation_t" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="ID" type="xs:string"/>
</xs:complexType>
<xs:complexType name="orderedString_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="order" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="value_t">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="units" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
```

```
 </xs:complexType>
 <xs:simpleType name="valueTypes_t">
  <xs:restriction base="xs:string">
   <xs:enumeration value="float"/>
   <xs:enumeration value="boolean"/>
   <xs:enumeration value="varchar"/>
   <xs:enumeration value="integer"/>
   <xs:enumeration value="URI"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="laterality_t">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
 <xs:simpleType name="tissueType_t">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
</xs:schema>
```

Bibliography

[BIRN] *Biomedical Informatics Research Network*. URL:<http://www.nbirn.net/>.

[CostelloSchemaVersioning] *Impact of XML Schema Versioning on System Design*. URL:<http://www.xfront.com/SchemaVersioning.html>.

[Schematron] Rick Jelliffe. URL:<http://www.schematron.com/>. ISO specification for Schematron available as ISO/IEC 19757-3:2006.

[XML10] *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. W3C. URL:<http://www.w3.org/TR/xml/>.

[XMLSchema10] *XML Schema 1.0*. Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. W3C. Primer: <http://www.w3.org/TR/xmlschema-0/>, Structures: <http://www.w3.org/TR/xmlschema-1/>, Datatypes: <http://www.w3.org/TR/xmlschema-2/>.