Configuration and Initialization of the XAL High-Level Application Development Environment[*]

Second Edition

Christopher K. Allen
Los Alamos National Laboratory
Los Alamos, New Mexico 87545 USA
ckallen@lanl.gov

C. M. Chu, John D. Galambos, Thomas A. Pelaia, and Andrei Shishlo
Spallation Neutron Source, Oak Ridge National Laboratory
Oak Ridge, Tennessee, 37831 USA

July, 2006

[*] Work supported by US Department of Energy

Abstract

This document describes the basic configuration mechanism of the high-level accelerator-application development environment known as XAL. Specifically, we describe how to configure the XAL system to a given particle accelerator site. The primary purpose of XAL is the rapid development of high-level applications needed for control and operation of charged-particle beam accelerators and accelerator systems. We begin with an overview of XAL, discussing the basic design philosophy and architecture. To utilize the XAL environment, it must be configured to the specific accelerator site, after which, applications may be written in a more general fashion, in particular, without regard to the specific machine to which it applies. There are four main XML files used to configure XAL to a given accelerator site: the main file, the optics file, the timing file, and the model-parameters file. There is also a legacy mechanism for entering particle beam data (used in simulation) known as the probe file. We discuss all of these files in detail.

Table of Contents

1 Introduction

The XAL software system is a programming environment specifically designed for developing high-level control applications for charged-particle accelerator systems. In its initial inception, it began as a support tool for the Spallation Neutron Source (SNS) in Oak Ridge, Tennessee, which continues to be the primary source of development activity. However, collaborative development efforts have existed at Los Alamos National Laboratory, Brookhaven National Laboratory, and, most recently Stanford Linear Accelerator. XAL attempts to provide a uniform standard for building high-level control applications for accelerator systems. It provides a common hardware-independent platform for building such applications. In this manner different accelerator sites can share software, leverage off existing development, and participate in collaborations where common interests occur.

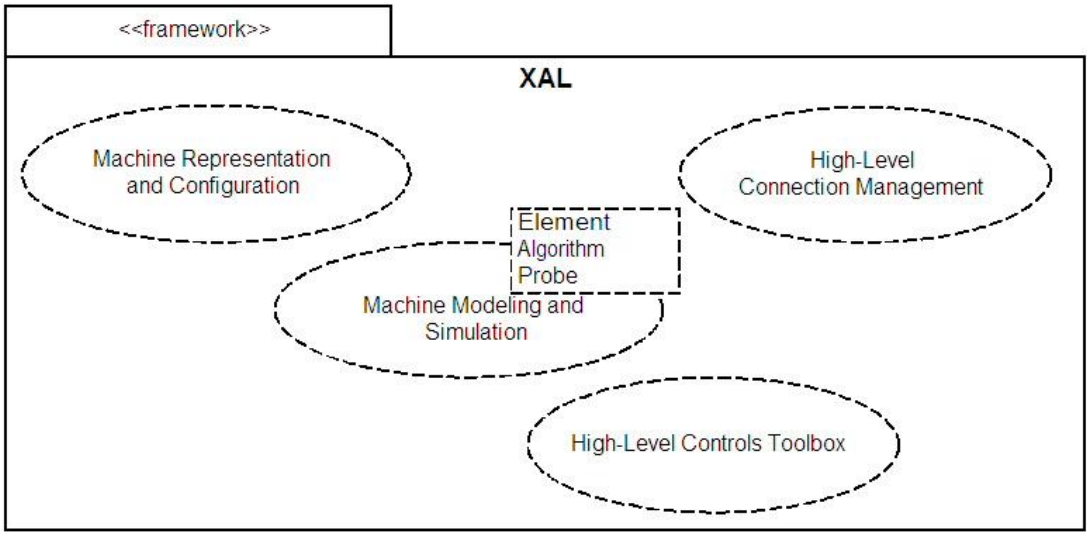
<<framework>>
XAL
Machine Representation and Configuration
High-Level Connection Management
Element
Algorithm
Probe
Machine Modeling and Simulation
High-Level Controls Toolbox

Figure 1: XAL mechanisms

This presentation first provides a brief introduction to XAL, then discusses in detail the XAL configuration mechanism. Specifically, we explain how to configure XAL to a particular accelerator complex. This activity primarily consists of describing the hardware arrangement of an accelerator complex and informing XAL of all the corresponding control signals (process variables) of the given hardware. As we shall see, this task is accomplished through the proper construction of a set of formatted XML (eXtended Markup Language [15]) files. Once these files are built, we may develop control applications using the XAL environment, and use many of the existing applications that ship with XAL.

1.1 XAL Overview

The basic purpose of XAL is to provide a high-level programming interface for charged-particle accelerator systems. That is, we write programs that communicate with the machine hardware from a high-level "physics perspective", rather than a lower-level "engineering perspective" (e.g., EPICS programming). XAL also attempts to provide uniformity in programming practice and in the developed applications. For example, XAL applications all present a uniform *look and feel*. Moreover, MKS units are typically used, the notable exceptions being described here. XAL also contains many "off-the-shelf" tools for building applications, such as an optimization package, a signal processing package, and a GUI framework. Typically, these tools are "boxed" according to a standard programming interface so that different tools within the same package may be swapped in and out. An UML mechanism diagram of the XAL system is shown in Figure 1. In the diagram we can see the four main mechanisms of XAL: a physics-centric representation of the hardware, an online simulator for model-based control applications, high-level connection management including synchronization between the hardware and the model, and a suite of common tools available to all XAL users. For a more comprehensive overview of the XAL environment one may consult references [6], [7], [8], and [14].

The XAL environment can be thought of as middleware between the applications programmer and the lower-level EPICS (Experimental Physics and Industrial Control System) software. Or as an alternative view, XAL provides an additional layer of abstraction from the EPICS communication layer. Where EPICS is essentially "flat", that is, all signals look alike; XAL presents the machine structure to the application developer. This concept is depicted in Figure 2 were we show the interdependencies of various parts in an accelerator control system. Notice in the figure that the application developer may ignore the GUI mechanism of XAL, or XAL altogether. XAL also communicates directly with Matlab [11] and the Jython [10] implementation of the Python scripting language. Thus, prototype applications may be quickly built from these scripting environments for testing and debugging.

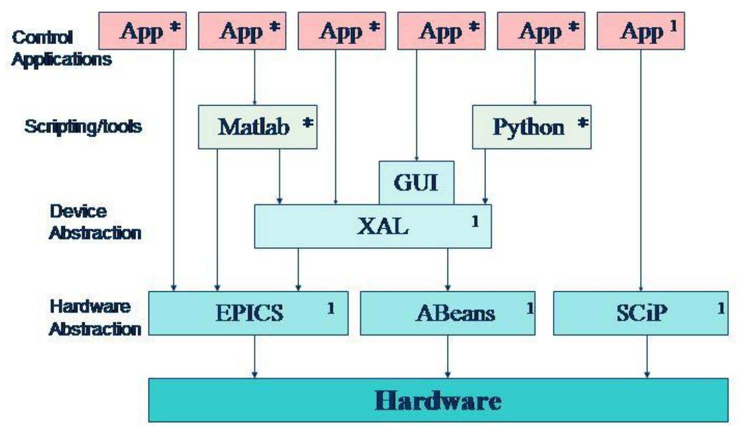

Figure 2: XAL in the control system hierarchy

Note also in the figure that XAL can support multiple communication protocols, in particular the EPICS channel access (CA) or Cosylab's Abeans [8]. The notion of a communication channel is abstracted in XAL. Although the application developer will probably seldom interact with channels directly, XAL treats every channel object in a uniform manner. The details of implementing a concrete channel are left to the underlying communications protocol. So far, EPICS has been the primary communications support protocol.

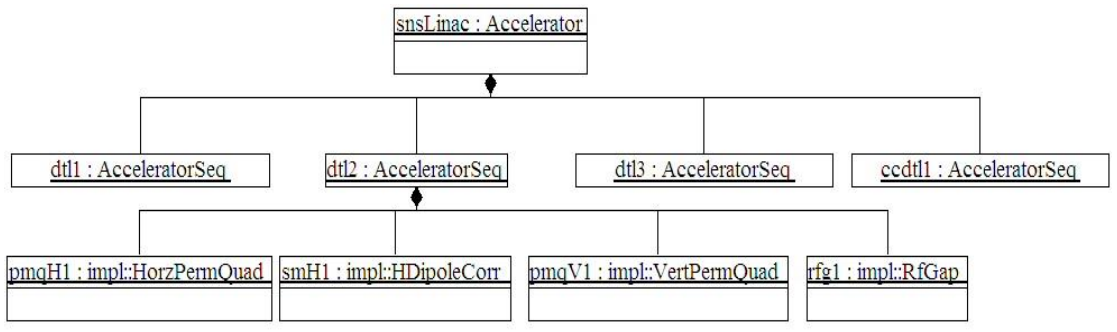

Figure 3: the XAL hardware datagraph

To support its physics-centric view of the accelerator, the XAL applications programming interface (API) is object-oriented where hardware objects such as beam position monitors (BPM), steering magnets, and quadrupole magnets are encapsulated by software objects. XAL provides a multi-layered view of the entire accelerator, which is configured dynamically (i.e., at run time). The developer may inspect the hardware configuration by traversing through a tree-like data structure of software objects. An UML instance diagram of this XAL "datagraph" is shown in Figure 3 for section of an example accelerator. Using this capability, the application developer is able write applications that are not "hard-coded" to a particular machine configuration. The practice of building "one-off" control applications that function only for a specific accelerator site can be eliminated with proper programming in XAL.

1.2 Basic Architecture of XAL

The architecture of XAL is based upon modern software engineering principles. In particular, XAL is implemented with a component-based architecture. The modern software engineering philosophy prescribes that very large software systems should be composed of modular components that fit together with well-defined software interfaces. This situation is exactly analogous to modern hardware design methodology where large systems are assembled from smaller components. For example, a computer is built from integrated circuits, circuit boards, hard drives, memory cards, and peripherals which all communicate via well-defined hardware interfaces (e.g., VMI, SCSI, PCI, USB, etc.). As long as the interfaces remain unchanged, hardware components which understand these interfaces may be swapped in or out of the system. Likewise, the situation exists with modern software systems. Thus, the difficult part is designing an interface that will accommodate all the requirements of the system components, including potential upgrades. Figure 4 is an UML component diagram showing some of the major components of XAL. There we see that the connection mechanism is a separate component of XAL, where either JCA (Java channel access), CAJ (channel access in Java), or Abeans may be used for the actual communication. Also shown in the figure is the configuration mechanism. This feature is accommodated (ideally) using a master database containing the machine hardware description from which a set of XML configuration files is instantiated. The XML files have a specific format and were intended to provide independence between database configuration and XAL development.

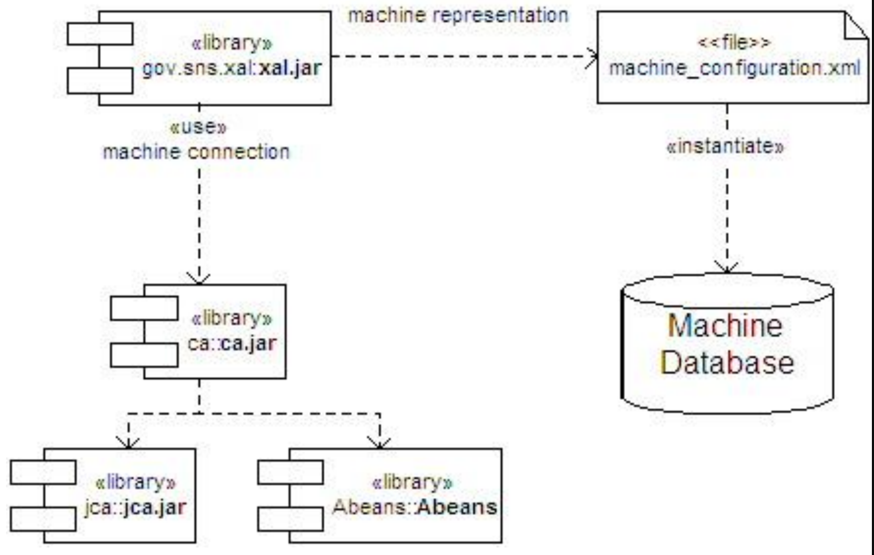

Figure 4: UML component diagram of XAL

1.3 XAL Configuration Mechanism

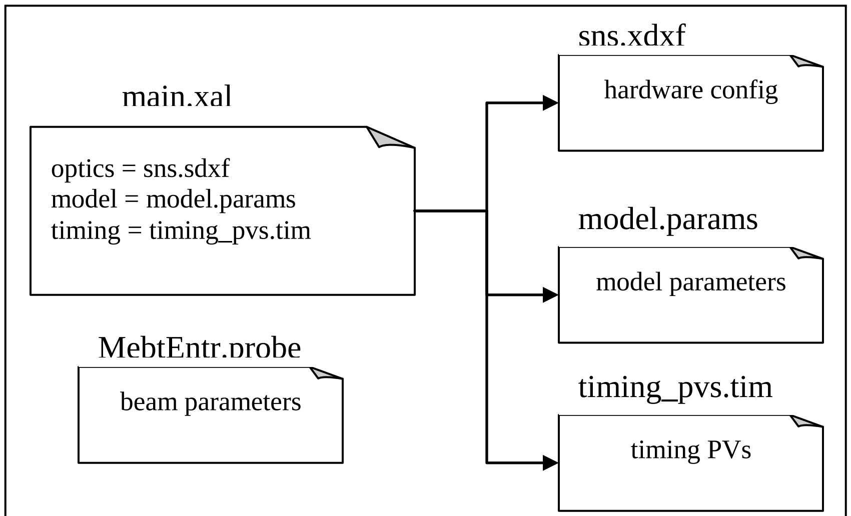

Figure 5: XAL configuration files

XAL is built on the principle of dynamic configuration. That is, XAL adapts to the current configuration of the underlying accelerator hardware, rather than building applications which a particular machine configuration. The component that drives this mechanism is a set of XML configuration files, as shown in Figure 5. The root file in the hierarchy is `main.xal`; it contains the names and locations of the other files in the XAL configuration mechanism. Currently there are three main files to which the root file `main.xal` points: one describes the hardware configuration of the accelerator (`sns.xdxf` in Figure 5), another describes the parameters required for beam simulation or "model parameters" of the machine (`model.params` in Figure 5), and the last describes the set of process variables (PVs) used for timing purposes (`timing_pvs.tim` in Figure 5). There also may exist separate "probe" files, Figure 5 depicts such a file called `MebtEntr.probe`. The data in such a file describe beam parameters necessary to instantiate a "beam probe" for machine simulation. This probe-file mechanism has more or less been deprecated by the model parameters file, whose data includes that of a probe file. However, such files may still be convenient in some circumstances.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE sources SYSTEM "xdxf.dtd">
<sources>
      <optics_source name="optics" url="sns.xdxf"/>
      <timing_source name="timing" url="timing_pvs.tim"/>
      <tablegroup_source name="modelparams" url="model.params"/>
</sources>
```

Excerpt 1: contents of main.xal

Excerpt 1 shows the contents of `main.xal` for the situation depicted in Figure 5. The first line is standard XML protocol specifying the character encoding used within the document (in this case UTF-8, or Unicode Transformed Format 8). The second line is important. It indicates the Document Type Definition file, or DTD file, specifying the data format in the current XML files; for XAL the DTD file is `xdxf.dtd`. Essentially, then, the `xdxf` format is the major topic of the presentation. DTD files are a means of not only specifying the format of an XML file, but enforcing it as well. However, DTDs are essentially being replaced by XML Schema Definitions, or XSDs. These format specifications are more flexible and can maintain tighter control of the XML layout. Currently there is no XSD within the XAL configuration system, which continues to use the DTD file `xdxf.dtd`. A listing of `xdxf.dtd` is provided in Appendix A. For those familiar with DTD files, this listing may help in the understanding of the XAL configuration mechanism. The sequel is devoted towards the explanation of the `xdxf` format and how to use it in order to describe the hardware configuration of an accelerator system.

The rest of `main.xal` contains a listing of all the configuration files used by XAL. These files are listed under the XML element tag `<sources>`. The element `<optics_source>` contains the URL of the file defining the hardware configuration of the accelerator (in the `xdxf` format); in this case `sns.xdxf`. The `xdxf` format specifies that there can be only one `<optics_source>` element, that is, it is a singleton. There are two attributes in the `<optics_source>` element, `name` and `url`. Since <optics_source> is a singleton element, the value of the `name` attribute is the identifier string of the XAL data graph. Since this is a singleton the value of its identifier is arbitrary, the user should pick a value that makes sense in context of the machine. The second attribute, `url`, points to the location of the actual file containing the configuration data (i.e., it is the URL of the configuration file). The next element, `<timing_source>`, also a singleton, points to a file which lists all the general timing signals used by XAL. Such signals are used by applications that trigger on certain events, like an RF pulse, or monitor certain aspects of the machine, such as the orbit. It, too, has the attributes `name` and `url`, and they are used in exactly the same way as for an `<optics_source>` element. Finally, the last element, `<tablegroup_source>`, is an example of what is called a *table group* in XAL.

Table groups contain arbitrary user data, stored in a schema-based table format, which is not otherwise part of the XAL specification, but available within the XAL framework. They are XAL's mechanism for providing dynamic data to applications and are discussed in detail within Chapter 4. There can be many `<tablegroup_source>` elements in `main.xal` and, thus, many table groups within the XAL framework. The attributes of a <tablegroup_source> element are, again, `name` and `url`. They are used just as the previous source elements, with the exception that the `name` attribute be unique. This requirement is necessary since there can be multiple <tablegroup_source> element; the unique value of the `name` attribute is used to refer to the particular table group within XAL. There is one special case of a table group within XAL, that having the name attribute with value "modelparams". This is the case shown in Excerpt 1. This particular table group is called the "model parameters" table group. This table group has a pre-defined format recognized by XAL to contain all the modeling data not directly associated with the machine hardware. In particular, all the beam parameters are contained in this file, as well as numeric parameters for doing simulations. The format of this special table group is discussed in Section 4.2.

1.4 XAL Persistent Data

Within XAL the various XML configuration files are managed by the class `gov.sns.xal.smf.data.XMLDataManager`. Instances of this class are the primary method for providing access to XAL's persistent data (i.e., data shared between XAL applications and maintained over time). This class is capable of not only loading XAL data but storing it as well. Thus, the optics file and any table groups may be modified using this class. The application developer would typically create an instance of `XMLDataManager`, point it to the URL of the `main.xal` file, then use it to built the XAL data graph representing the accelerator hardware. Of course, there are already mechanisms available in the application framework within XAL providing such services, yet, we wish to point out the nature of the `XMLDataManager` class since it is central to the XAL configuration mechanism.

Once an instance of `XMLDataManager` is pointed to the `main.xal` XAL file, the output is a populated instance of the data structure held in the `Accelerator` class. An instance of `Accelerator` is a representation of all the machine hardware and some supporting systems according to the configuration described in the optics file, timing file, and any miscellaneous data from the table groups. The `XMLDataManager` class employs two support classes from the same package, `TimingDataManager` and `OpticsSwitcher`. The `TimingDataManager` class is responsible for reading the timing file (described in Section 3) and creating a `TimingCenter` object, which is attached to the `Accelerator` object. The `XMLDataManager` also contains an internal class, `TableManager`, for reading and storing any miscellaneous table group data identified by `<tablegroup_source>` elements in the main configuration file. This auxiliary data is accessible from a singleton `EditContent` object, which is also attached to the `Accelerator` object. The `EditContext` object is like a miniature relational database, it is the method by which XAL programmatically stores and retrieves table group data that does not otherwise fit into the current architecture. It is part of the `gov.sns.tools.data` package, which it shares with many of its component classes. Again, for a full description of this data mechanism see Chapter 4.

2 Accelerator Optics: The <optics_source> File

The optics configuration file, or *optics file* in XAL terminology, is tagged `<optics_source>` in `main.xal`. It contains the hardware components of the accelerator and their current configuration. Obviously XAL is not concerned with every hardware component in the machine, only those pertinent to high-level control. So there are many components of the machine that are not in the optics file (e.g., gimbals, rheostats, cryostats, etc.). XAL presents to the high-level applications programmer only hardware objects to that are listed in the optics file. Thus, if a piece of hardware is irrelevant to high-level operations, then it is simply omitted from this file. The hardware configuration file also provides binding between a hardware component, other related components, and the process variables which control them. For example, a quadrupole magnet may be listed in the configuration file, along with its bulk power supply (i.e., a power supply connected to several quadrupole magnets) its trim coil, the process variables (signals) that control them, and the read back signals for all.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE xdxf SYSTEM "xdxf.dtd">
<xdxf system="sns" ver="2.0.0" date="Fri Jan 27 14:08:32 EST 2006">
  <comboseq id="MEBT-DTL">
         <sequence id="MEBT"/>
         <sequence id="DTL1"/>
         <sequence id="DTL2"/>
         <sequence id="DTL3"/>
         <sequence id="DTL4"/>
         <sequence id="DTL5"/>
         <sequence id="DTL6"/>
  </comboseq>
```

Excerpt 2: example XAL optics file

The optics file is composed primarily of the XML elements `<node>`, `<sequence>`, `<comboseq>`, `<channelsuite>`, and `<powersupplies>`. A `<node>` element represents a basic hardware object, such as a wire scanner, a beam position monitor (BPM), or a steering magnet. A `<sequence>` element contains a linear sequence of `<node> elements` representing a sector of the accelerator system. A `<comboseq>` element describes the various ways that sequences may be pasted together to form viable beam paths within the accelerator system. This feature is essential for accelerator complexes having multiple beamlines, that is, systems where the beam may propagate down several different paths. The user of an XAL application may select one of these possible beam paths for analysis. With the use of `<comboseq>` objects his or her choice is restricted to the viable beamlines within the complex. The `<channelsuite>` element is the method by which XAL associates hardware objects to their control signals. Each hardware object typically has a “suite” of appropriate control signals by which the user may interact. For example, a steering magnet typically has signals for setting the dipole current and observing the actual current value seen at the magnet location. Finally, the `<powersupplies>` element contains a listing of all the power supplies relevant to the XAL accelerator description. Although this element is not contained in the `xdxf` specification (it was added to XAL later), it is an extremely important component of the optics file and XAL will not function correctly without it. There should be only one `<powersupplies>` entry and typically it occurs at the end of the optics file. We consider each of these XML elements in detail.

Attribute	**Description**
`system`	accelerator system name
`ver`	File version stamp
`date`	File build date

Table 1: <xdxf> attributes

Excerpt 2 shows the initial section of a typical XAL optics file. The first two lines are similar to that of the `main.xal` file, they are XML requirements specifying the character encoding and any XML format specifications (e.g., the `xdxf` DTD specification). The next line is the first XML element in the file, the `<xdxf>` element. There is only one `<xdxf>` element in the optics file and it contains a description of the all the accelerator hardware considered by XAL. That is, the `<xdxf>` element is the root of the accelerator hierarchy, containing all the accelerator hardware information in the optics file. As seen in the `xdxf.dtd` file, there are three required XML attributes of the `<xdxf>` element, `system`, `ver`, and `date`. These attributes are listed in Table 1. The `system` attribute is the name of the accelerator system that is described in the `<xdxf>` element. In Excerpt 2, its value is “sns” indicating that the optics file describes the Spallation Neutron Source. The `ver` attribute contains any versioning information associated with the file. This feature allows users to monitor changes in the XAL configuration with a version stamp, facilitating more accurate comparisons of archived data. The final attribute, `date`, contains the date in which the optics file was last built. The `date` value can be expressed in the ISO 8601 standard [12], however, any date format recognized by the Java class `DateFormat` [9] is valid.

```
<node type="QH" id="MEBT_Mag:QH01" pos=".128" len=".061" status="true">
  <attributes>
    <magnet len=".061" polarity="-1" dfltMagFld="-34.636"/>
      <align x="0.0" y="0.0" z="0.0" pitch="0" yaw="0" roll="0"/>
      <aperture shape="0" x=".016"/>
   </attributes>
  <ps main="MEBT_Mag:PS_QH01"/>
  <channelsuite name="magnetsuite">
     <channel handle="fieldRB" signal="MEBT_Mag:QH01:B"
        settable="false"/>
  </channelsuite>
</node>
```

Excerpt 3: example 6<node> element for a horizontal quadrupole magnet

2.1 Basic Hardware Objects: The <node> Element

All basic hardware elements are described by `<node>` element entries within the optics file. Excerpt 3 is an example of a typical `<node>` entry; in this case the node represents a quadrupole magnet connected to a bulk power supply and oriented for focusing in the horizontal direction (type code "QH"). There are three required XML attributes of every `<node>` element and two optional attributes. All these attributes are listed in Table 3. The required attributes are `type`, `id`, and `pos`. The `type` attribute is the hardware type identifier string; these identifiers are listed in Table 2. The `id` attribute contains the unique identifier string of the hardware object which the <node> represents. This attribute is generally the same identifier given to the device in the global database, or machine schematics. The `pos` attribute is the offset (in meters) of the hardware object from the *beginning* of its container `<sequence>`. Thus, the value of `pos` is actually the position of the `<node>` within the accelerator sector described by the parent `<sequence>` element. The `pos` attribute always refers to the position of the `<node>` center location. The optional attributes of a <node> element are `len` and `status`. The `len` attribute represents the length of the element, taken to be zero if absent. The `status` attribute is basically now a legacy feature of XAL meant to indicate whether or not a hardware component was on line. If present it should always be set to "yes".

Attribute	Description
`type`	Type identifier (see Table 2)
`id`	Unique identifier string
`pos`	Position within sector (m)
`len`	Length of hardware node (m)
`status`	Legacy parameter (="yes")

Table 3: <node> element attributes

2.1.1 Node Attribute Buckets

The first child element of the <node> element we consider is `<attributes>`. According to the current `xdxf` format, this element is required. The `<attributes>` entry is meant to contain design parameters for the `<node>`, called "attribute buckets" in XAL terminology. The `<attributes>` entry may optionally contain three child elements called `<align>`, `<magnet>`, and `<rfgap>`. There are also other attribute buckets not listed in the `xdxf` format that may be contained in this section. These buckets contain parameters for specialized nodes that were added to XAL later in the development, such as parameters for the specialized ring

XAL Accelerator Node Types		
Type Id	**XAL Class**	**Hardware Object**
BCM	`CurrentMonitor`	Beam current monitor
Bnch	`ReBuncher`	Re-buncher cavity
BPM	`BPM`	Beam position monitor
DCH	`HDipoleCorr`	Dipole corrector, hor.
DCV	`VDipoleCorr`	Dipole corrector, vert.
QH	`Quadrupole`	Quadrupole magnet, hor.
QV	`Quadrupole`	Quadrupole magnet, vert.
PMQH	`PermQuadrupole`	Permanent mag. quad. hor.
PMQV	`PermQuadrupole`	Permanent mag. quad. ver.
RG	`RfGap`	Radio frequency gap
WS	`ProfileMonitor`	Wire scanner
DH	`Bend`	Dipole bend magnet, hor.
DV	`Bend`	Dipole bend magnet, vert.
RBPM	`RingBPM`	Ring BPM

beam position monitors. We discuss these specialized buckets below. We shall see that the `<sequence>` element may also contain attribute buckets needed to specify collective properties of the accelerator sector which it describes. We cover sequence attribute buckets when we discuss the properties of `<sequence>` elements. Finally, we point out that, as of this writing, no attribute bucket may contain child elements. They are XML elements whose XML attributes contain design parameters of the device they are intended to describe.

The `<align>` attribute bucket contains parameters describing the measured misalignments for the `<node>` element. These attributes are listed in Table 4. As seen in the table, these parameters are `x`, `y`, `z`, the horizontal, vertical, and longitudinal offsets, respectively, and `pitch`, `yaw`, `roll`, the angles describing the rotational offset of the `<node>`. The `pitch`, `yaw`, and `roll` angles are the same as that in aeronautics, where we take the beam axis as the eigenaxis. Specifically the `pitch` angle is the angle of inclination with respect to the horizontal (rotation about the *x* axis), the `yaw` angle is the horizontal deflection angle (rotation about the *y* axis), and the `roll` angle is the rotation about the beam axis (rotation about the *z* axis). These angle are depicted graphically in Figure 6. As of the current writing, the alignment parameters have not been utilized in the XAL model. However, they are accessible from within the framework.

Attribute	Description
`x`	Horizontal offset (m)
`y`	Vertical offset (m)
`z`	Longitudinal offset (m)
`pitch`	Rotat. about x axis (deg)
`yaw`	Rotat. about y axis (deg)
`roll`	Rotat. about z axis (deg)

Table 4: align attribute bucket attributes

The `<magnet>` attribute bucket contains the design attributes relevant to a `<node>` representing some type of magnet device. The parameters of a <magnet> attribute bucket are listed in Table 5. The first three parameters represent general characteristics of a magnet device. Their interpretation depends upon the type of magnet being described in the `<node>` entry. They are `len`, specifying the effective length of the magnet (in meters), `dfltMagFld`, specifying the design field strength of the magnet, and `polarity`, specifying the field polarity. For a dipole magnet the value of `dfltMagFld` is actually the field strength in Telsa. However, for a quadrupole magnet this value represents the field *gradient*, in Tesla per meter. The value of polarity is somewhat complicated, thus, we discuss it in the next paragraph. The next two parameters, `multFieldNorm` and `multFieldSkew`, are the field components in the normal direction and the tangential direction, respectively. For the sake of clarity, it is unfortunate that within the XAL classes, the `multFieldSkew` value is actually referenced using the identifier "`Tang`" rather than "`Skew`". For example, you must use `getTangField()` rather than `getSkewField()` in the `MagnetBucket` class. There is only one required attribute of a `<magnet>` attribute bucket, the `len` attribute.

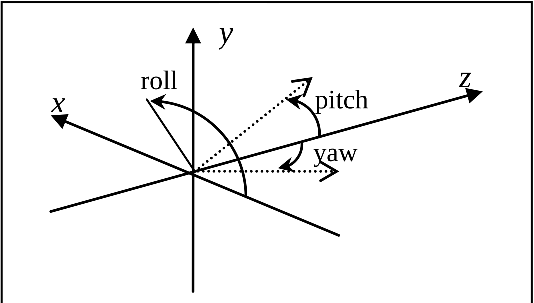

Figure 6: alignment angles

The interpretation of the `polarity` attribute is somewhat convoluted so we cover it in some depth. The attribute can have the values "–1" or "+1", meaning negative or positive polarity, respectively. Although `polarity` does determine the field direction with respect to the design direction, unfortunately, it actually refers specifically to the electric current direction with respect to its design direction. In electromagnets the field polarities are actually *opposite* to those of the current. So a `polarity` value of "–1"

Attribute	Description
`len`	Effective length (m)
`dfltMagFld`	Design field strength
`polarity`	Field direction (-1,+1)
`multFieldNorm`	Normal field component
`multFieldSkew`	Tangential field component
`bendAngle`	Dipole deflection angle (deg)
`dipoleEntrRotAngle`	Entrance pole face angle (deg)
`dipoleExitRotAngle`	Exit pole face angle (deg)
`dipoleQuadComponent`	Quadru. component of dipole
`pathLength`	Magnet path length (m) ???

Table 5: <magnet〉 attribute bucket parameters

means *negative* current, but *positive* field direction. In the case of a bending magnet, `polarity` indicates the bend direction while facing downstream, "–1" indicating a right bend and "+1" indicating a left bend. To make this situation easier to remember, consider the specific situation of a horizontal bending magnet. Then a value of "–1" indicates a bend in the *negative x* direction (the dipole fields point in the positive *y* direction) while a value of "+1" indicates a bend in the *positive x* direction (the dipole fields point in the negative *y* direction). When the `<node>` entry describes a quadrupole magnetic, a `polarity` value of "–1" indicates a (horizontally) focusing quadrupole while a value of "+1" indicates a (horizontally) defocusing quadrupole. Here, according to convention, focusing refers to focusing in the horizontal plane and *negative* current means *positive* field producing a *focusing* effect. Apparently, for a dipole corrector magnet, (i.e., a steering magnet) `polarity` indicates the direction of the correction. It is assumable that the situation is the same as that for a bending dipole, for a horizontal corrector a value "–1" means that positive current deflects in the negative *x* direction while a value "+1" deflects in the positive *x* direction.

The next five parameters of the `<magnet>` attribute bucket are particular to a dipole bending magnet. They are `bendAngle` (in degrees), `dipoleEntrRotAngle` (in degrees), `dipoleExitRotAngle` (in degrees), `dipoleQuadComponent` (in 1/meters^2), and `pathLength` (in meters). The `bendAngle` attribute is the deflection of in the design trajectory caused by the bend. A negative bend angle indicates a bend to the left (positive *x* direction for a horizontal dipole) while a positive angle indicates a bend to the right (negative *x* direction for a horizontal dipole). The `dipoleEntrRotAngle` is the angle of the magnet pole face respect to the design trajectory at the magnet entrance, while `dipoleExitRotAngle` is analogous angle at the exit of the magnet. A pictorial representation of these parameters is shown in Figure 7. Note that for a horizontal bend the angle is take positive if the pole face normal lies in the positive *x* bending plane. The attribute `dipoleQuadComponent` specifies the quadrupole field component of the dipole magnet. Clearly for an ideal dipole this value is zero. Typically, however, a field gradient is introduced to provide transverse focusing within the magnet. A commonly used measure of this gradient at the design radius R_0 is the field index $n = (R_0/B_y(R_0))(dB_y(R_0)/dR)$. In terms of n, the `dipoleQuadComponent` value is $-n/R_0^2$. The `pathLength` attribute is the total path length of the design trajectory through the dipole magnet. In the case of a dipole the `len` attribute then refers to the *physical length* of the magnet, not the path length of the design trajectory.

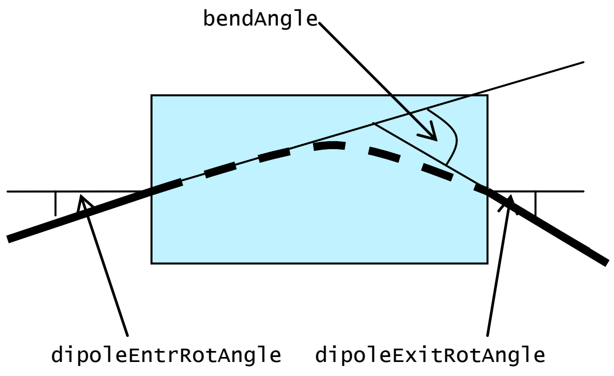

Figure 7: dipole bending magnet parameters

Attribute	Description
`x`	Horizontal dimension (m)
`y`	Vertical dimension (m)
`shape`	Aperture geometry 0 Unknown 1 Ellipse 2 Rectangle 3 Diamond 11 Irregular

Table 6: <aperture> attribute bucket parameters

In the future it may be wise to create a separate attribute bucket for each magnet type, especially since the sextupole magnet modeling element is implemented. For example we might have attribute buckets `<dipolemag>`, `<quadrupolemag>`, `<sextupolemag>`, etc., containing parameters specific to each device type. However, recall that the only required attribute of a <magnet> attribute bucket is `len`. Since the last four attributes are not used for a quadrupole magnet it is not necessary to specify them in the optics file.

The `<aperture>` attribute bucket specifies the `<node>` aperture shape and size. The `<aperture>` element has three attributes, `x`, `y`, and `shape`. Attributes `x` and `y` state the horizontal and vertical dimensions of the aperture (in meters), respectively, while `shape` is an enumerated value indicating the shape of the aperture. The enumerations for `shape` attribute are listed in Table 6, along with a description of the other attributes in the `<aperture>` attribute bucket. Note to specify a circular aperture of radius *r*, we would set the following attributes `x` = *r*, `y` = *r*, and `shape` = 1.

The last attribute bucket in the `xdxf` specification is the `<rfgap>` bucket. Obviously, this set of parameters describes the properties of a radio frequency gap. The important attributes of the `<rfgap>` attribute bucket are `length`, `phaseFactor`, `ampFactor`, `TTF`, `endcell`, and `gapOffset`. There are additional attributes but they are either now legacy attributes or primarily used for testing. The `length` attribute specifies the length of the gap in meters. The `TTF` attribute is the transit time factor of the RF gap, which is a parameter determined by the geometry of the gap. The remaining attributes are all related to the parent `<sequence>` object to which the `<node>` belongs. Since an RF gap is assumed to be part of a larger accelerating structure, such as a drift tube linac or elliptical cavity, a `<node>` representing an RF gap is contained within a `<sequence>` representing such a structure. Thus, this `<sequence>` is some type of electromagnetic cavity driven by a radio frequency source. The attribute `phaseFactor` is the ratio of the RF phase at the first gap of the `<sequence>` to the RF phase at the current gap. The attribute `ampFactor` is the ratio of the RF amplitude at the first gap to the RF amplitude at the current gap (typically "1"). The attribute `endcell` is a flag to indicate whether or not the current RF gap is the final gap in the parent `<sequence>` ("0" if not, "1" if so). The final attribute `gapOffset`, specifies the offset of the gap center relative to its specified position within the `<sequence>` (given by the `<node>`'s `pos` attribute).

Attribute	**Description**
`length`	Gap length (m)
`phaseFactor`	Ratio of phase at gap #1 to this gap
`ampFactor`	Ratio of RF amp at gap #1 to this gap
`TTF`	Transit time factor
`endCell`	Last gap flag (0 or 1)
`gapOffset`	Gap center offset (m)

Table 7: <rfgap> attribute bucket parameters

Finally, we discuss two `<node>` attribute buckets that are not in the `xdxf` specification, `<bpm>` and `<twiss>`. The `<bpm>` attribute bucket naturally applies only to beam position monitors. It has three attributes, `length`, `frequency`, and `orientation`, which are listed in Table 8. The `length` attribute specifies the length of the stripline in the beam position monitor (in meters). The `frequency` attribute is the phase frequency of the beam position monitor, which is unfortunately in *megahertz* rather than hertz. The final attribute, `orientation`, is an enumeration indicating the direction of the leads, "-1" indicating that the leads come into the BPM from the downstream direction, "+1" indicating the upstream direction.

Attribute	**Description**
`length`	BPM stripline length (m)
`frequency`	Phase frequency (MHz)
`Orientation`	Lead orientation (-1, +1)

Table 8: <bpm> attribute bucket parameters

The `<twiss>` attribute bucket lists the design Twiss (or Courant-Snyder) parameters of the *beam* at the current `<node>`. Since the machine parameters and the beam parameters are now separately described within the XAL framework, this attribute bucket is now essentially a legacy feature and is not seen in the current configuration files. However, for completeness we list its attributes which should be self explanatory: `x, y, ax, bx, ex, ay, by, ey, az, bz, ez, etx, etpx, ety, etpy, mux, muy`.

2.1.2 Power Supplies: The <ps> Element

Whenever a hardware object is attached to a power supply we see a `<ps>` (for "power supply") element entry under the `<node>` entry. This element is not currently listed in the `xdxf` format, but it is an important part of the configuration process, XAL will not function correctly without it. The `<ps>` entry is used to indicate any power supply connected to the current `<node>` object, this includes differing types of power supplies such as those for a dipole corrector magnet or a quadrupole focusing magnet. If an application developer wishes to change the field strength of a magnet, he must do so using the power supply object referenced by the `<ps>` entry. Of course, doing so will also change the field strength of any other magnet connected to the supply. Consequently, we see that this mechanism allows XAL to handle

Attribute	**Description**
`main`	UID of a main power supply
`trim`	UID of a trim power supply

Table 9: <ps> element attributes

bulk supplies. The XML attributes of a <ps> element when seen under the context of a <node> element are main and trim, as shown in Table 9, both optional but relevant only if at least one exists. In Excerpt 3 we see that only the main attribute is present. Either attribute specifies the unique identifier string of a power supply described in the <powersupplies> entry of the optics file. The <ps> element also has another context, as a child of the <powersupplies> element. There, the <ps> element has different attributes. For a full description of the XAL <powersupplies> element see Section 2.4, however, for the sake of completeness we offer a brief description of the power supply configuration mechanism below.

In the singleton <powersupplies> entry, each power supply in the accelerator is described by a <ps> element entry. In that context the <ps> element has two attributes, id and type (see Table 16). The value of id is the unique identifier string of the power supply while the value of type is currently either "main" or "trim". If type equals "main" then the Java class type representing it in XAL is MainMagnetSupply. If type equals "trim" then the object representing the supply is of class type MagnetTrimSupply. The remaining part of the <ps> entry describes all the process variables connected to the power supply. Returning to the context of a <node> element, the main attribute of <ps> points to a power supply feeding the current <node>, which must be of type "main". The trim attribute also points to a connected power supply, but of type "trim". Either of these two attributes may be present. However, the most likely scenario is that either the main attribute appears alone, or in conjunction with a trim attribute. The later case describes a node that is connected to both a main supply and a trim supply. There we assume the main supply is a bulk supply feeding many nodes, but the operating point of the current node can be varied around the quiescent point using the trim supply. Typically there are also a readback signals directly listed in the <node> element's channelsuite to verify the magnet strength (see Section 2.1.3).

2.1.3 Node Connectivity: The <channelsuite> and <channel> Elements

The last section of a <node> entry is the <channelsuite> element. The <channelsuite> element contains a listing of all the signals *specifically* associated with that <node>. A <channelsuite> element has an optional attribute name, which contains the identifier of the channelsuite. This is *not* a unique identifier. It is primarily present as a description of the signal set. For example, in Excerpt 3 we see that the name attribute has the value "magnetsuite", implying that this is the set of channels for a magnet.

Attribute	Description
handle	Internal XAL channel handle
signal	Channel name
settable	Bidirectional channel flag

Table 10: <channel> element attributes

Appearing under the <channelsuite> element is a listing of <channel> elements connected to the parent <node>. The <channel> element is the mechanism by which XAL and the underlying communications protocol (e.g., EPICS) are bound. Each <channel> element describes a control signal connected to the parent <node>. The <channel> element uses three attributes describing this signal, handle, signal, and settable. These attributes are listed in Table 10. The handle attribute specifies the name binding of the channel used internally within XAL. This attribute is very important. It indicates the type of communications channel and binds the channel to the XAL device field. For example, in Excerpt 3 we see that the handle attribute of the <channel> entry in the <channelsuite> has the value "fieldRB". Thus, XAL knows that this is a readback signal for a horizontal quadrupole magnet and, thus, binds the channel to the getField() method of the Quadrupole object representing that <node>. We discus this topic further in the next paragraph. The signal attribute is the channel name of process variable represented by the <channel>. When EPICS is used as the communications protocol, the value of signal is the EPICS channel name. The last attribute, settable, indicates whether or not the channel is bi-directional. That is, can the user modify the value or state of the <channel> object. If the <channel> is a readback process variable, this value is "0", otherwise it is "1" indicating that it is "settable".

Within XAL itself, a channel is represented by an instance of the class Channel. Seldom should it be necessary to interact directly with a Channel object, these objects are usually encapsulated by the methods of classes representing hardware devices. However, if it is absolutely necessary to work with a Channel object (this typically results in less portable code), there are means to acquire references to them.

In XAL's current state, the channel handles (i.e., the handle attribute of a <channel> entry) are hard-coded in the classes representing the hardware objects. All of these classes are children of the class gov.sns.xal.AcceleratorNode and belong to the Java package gov.sns.xal.smf. In the future it may be wise to bind the channel handles with a properties file so they exist in a common location. A convenient method for determining the channel handles is to consult the Javadoc for XAL. Regrettably however, much of the Javadoc for the gov.sns.xal.smf package is currently incomplete, so it is quite probable that one may not find the desired value there. Thus, the safest method is unfortunately also the most inconvenient, direct inspection of the source code for the node type under consideration. To help mitigate the inconvenience we list the set of current channel handles in Table 23 of Appendix B, with the caveat that these values may change in the future.

```
<sequence type="Bnch" id="MEBT_RF:Bnch01" pos=".528" len=".13" status="true">
   <attributes>
      <align x="0.0" y="0.0" z="0.0" pitch="0" yaw="0" roll="0"/>
      <sequence predecessor="null"/>
      <rfcavity                amp="1.3"
             TTFCoefs=".445, 0, 0"
             TTFPrimeCoefs="0, 0, 0"
             STFCoefs="0, 0, 0"
             STFPrimeCoefs="0, 0, 0"
             TTF_endCoefs=".445, 0, 0"
             TTFPrime_EndCoefs="0, 0, 0"
             STF_endCoefs="0, 0, 0"
             STFPrime_endCoefs="0, 0, 0"
             structureMode="0"
             phase="-90" freq="402.5"/>
   </attributes>
   <channelsuite name="rfsuite">
     <channel handle="cavAmpSet" signal="MEBT_LLRF:FCM1:CtlAmpSet" settable="true"/>
     <channel handle="cavPhaseSet" signal="MEBT_LLRF:FCM1:CtlPhaseSet" settable="true"/>
     <channel handle="cavAmpAvg" signal="MEBT_LLRF:FCM1:cavV" settable="false"/>
     <channel handle="peakErr" signal="MEBT_LLRF:FCM1:PeakErr" settable="false"/>
     <channel handle="regErr" signal="MEBT_LLRF:FCM1:RegErr" settable="false"/>
     <channel handle="resErrAvg" signal="MEBT_LLRF:ResCtrl1:ResErr_Avg" settable="false"/>
     <channel handle="cavPhaseAvg" signal="MEBT_LLRF:FCM1:cavPhaseAvg" settable="false"/>
     <channel handle="deltaTRFStart" signal="MEBT_LLRF:FCM1:deltaTRFStart" settable="true"/>
     <channel handle="deltaTRFEnd" signal="MEBT_LLRF:FCM1:deltaTRFEnd" settable="true"/>
     <channel handle="tDelay" signal="MEBT_LLRF:FCM1:tDelay" settable="true"/>
   </channelsuite>
   <node type="RG" id="MEBT_RF:Bnch01:Rg01" pos="0">
     <attributes>
        <rfgap length="0.13" phaseFactor="0" ampFactor="1" TTF="0.445" endCell="0"
               gapOffset="0"/>
     </attributes>
   </node>
</sequence>
```

Excerpt 4: example <sequence> listing

2.2 Accelerator Sectors: The <sequence> Element

Accelerator sectors are described with the <sequence> element in the xdxf format. Typically a <sequence> element contains a linear listing of <node> elements, each describing a particular hardware object in the beamline. However, <sequence> objects can also contain other elements, specifically <attributes>, <channelsuite>, and even other <sequence> objects. In the later case a <sequence> element is nested in another <sequence> element in order to described composite accelerator components. There are no restrictions on the depth of this nesting. If a <channelsuite> element appears under a <sequence> listing, it is assumed that the sequence describes some type of macro device with collective properties, such as an RF cavity. Typically, then, the <channelsuite> element will contain a listing of all the control process variables associated with that macro device (e.g., RF amplitude, RF phase, etc.). Finally, as with a <node> element, the <sequence> element also has a set of attributes. These parameters listed under the <attributes> child element and describing collective properties of the sequence.

Excerpt 4 shows an example <sequence> element entry in the XAL optics file. This particular sequence represents a rebuncher cavity and contains only one <node> object, an RF gap. The rest of the <sequence> entry contains the process variables associated with the rebuncher, under the <channelsuite> listing, and the attribute buckets of the <sequence>, under the <attributes> element. Note also the XML attributes of the `<sequence>` element, these are also essential for XAL operation. We cover all of these points in this subsection.

2.2.1 Sequences Representing RF Cavities

Since XAL very often uses sequence elements to represent RF accelerating structures, we digress briefly to review RF cavities in general and elaborate specifically on how they are modeled in XAL. Consider Figure 8 where we depict a model of an RF cavity. Figure 8a) shows a four-cell superconducting elliptical cavity while Figure 8b) shows our generalized model for all RF cavity structures. The elliptical cavity is included to demonstrate how most accelerating structure geometries can be considered under the current model. In the figure each gap has an index $n = 1,2,\ldots,N$; the index $n = 0$ is reserved for the entrance to the cavity. We denote by W_n the energy of the beam *before* entering gap n and W_{n+1} the energy of the beam *after* gap n. For example, W_0 is the energy of the beam entering the cavity and W_1 is the beam energy after the first gap. Likewise, denote the particle phase at the center of each gap as ϕ_n, reserving ϕ_0 for the particle phase at the moment it enters the cavity. Thus, ϕ_0 actually represents a process variable, the phase of the driving RF klystron.

Now consider a single beam particle propagating on-axis through the RF cavity. Its location is described completely by the path length parameter s. Within an RF cavity there is a longitudinal electric field E_z which accelerates our beam particle. This field E_z has the time varying form $E_z(s;t) = E_z(s)\cos(\omega t+\phi_0)$ where ω is the angular frequency of the RF and ϕ_0 is the phase offset. Letting $\phi(s)$ represent the phase of the particle (with respect to the RF) at axial position $s = s(t)$, then the field seen by the particle is given by $E_z(s)\cos[\phi(s)]$. Accordingly, the energy $W(s)$ of this particle is given by its initial energy W_0 (upon entering the cavity) plus the work done on it by the axial field E_z. We have

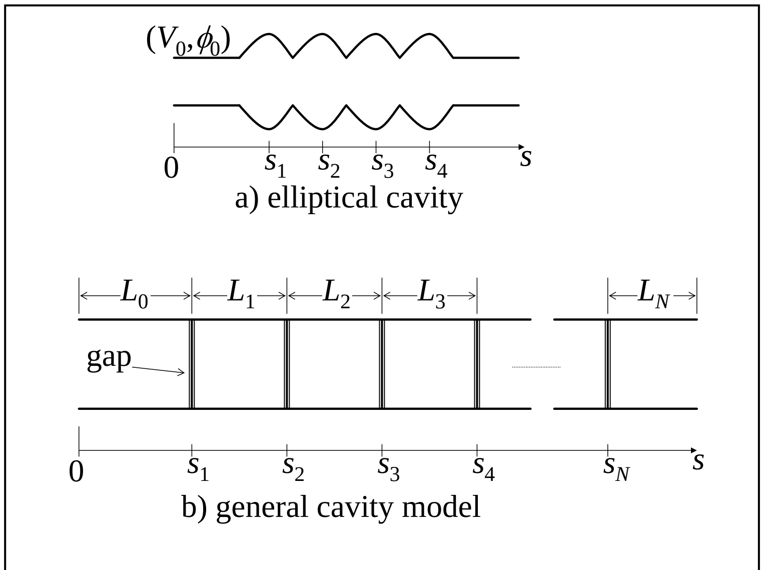

Figure 8: RF cavity model

(1)
$$W(s) = W_0 + qZ\int_0^s E_z(\sigma)\cos(\phi(\sigma))d\sigma\,,$$
$$\phi(s) = \phi_0 + \int_0^s k(\sigma)\,d\sigma\,,$$

where q is the unit charge, Z is the particle charge-state number, and $k(s)$ is the wave number at axial location s. Although presented as a function of path length s, the wave number k is more specifically a function of the particle normalized velocity β, which is in turn a function of s (due to the accelerating fields). (Note $\beta \equiv \dot{s}/c$, where c is the speed of light.) We have

(2)
$$k(s) \equiv \frac{2\pi}{\beta(s)\lambda}\,,$$

where λ is the RF wavelength in free space. Eqs. (1) are more or less exact, however, they are not terribly useful for analysis, or even computation. The difficulty arises in that these integral equations are coupled due to the particle velocity's implicit dependence on particle energy, that is, $\beta(s) = \beta(W(s))$. There are some simplifying assumptions that reduce Eqs. (1) to a more manageable set.

When the particle velocity β is constant, say through a drift, the wave number k is also constant. In this situation the equations in (1) decouple and have a straightforward solution. Now consider when the change in velocity β is small within a gap. We can approximate the particle phase $\phi(s)$ through gap n as $\phi(s) \approx \phi_n + \bar{k}_n s$ where ϕ_n is the *mid-gap* particle phase at n and $\bar{k}_n \equiv 2\pi / \bar{\beta}_n \lambda$ is the *mid-gap* wave number using the mid-gap velocity $\bar{\beta}_n$ of gap n. Using $\cos(\phi_n + \bar{k}_n s) = \cos(\bar{k}_n s)\cos(\phi_n) - \sin(\bar{k}_n s)\sin(\phi_n)$ for each gap n yields the familiar result for the energy gain involving the generalized transit time factors S and T. Specifically, the energy gain ΔW_n imparted to the beam by cavity gap n is given by the formula [16][24]

$$\Delta W_n = qZV_0\left[T_n(\bar{k}_n)\cos\phi_n - S_n(\bar{k}_n)\sin\phi_n\right] \tag{3}$$

where V_0 is the voltage across the gap, that is, $V_0 \equiv \int E_z(s)ds$, $\bar{k}_n$ is the mid-gap wave number of gap n, and ϕ_n is the particle phase at the gap center. The quantities $T_n(k)$ and $S_n(k)$ are the transit-time factors for the longitudinal field $E_{z,n}(s)$ generated by gap n. Mathematically these quantities are the Fourier cosine and sine transforms, respectively, of $E_{z,n}(s) \equiv E_z(s{-}s_n)$. They are defined as follows:

$$\begin{aligned} T_n(k) &\equiv \frac{1}{V_0}\int_{s_n-l_n/2}^{s_n+l_n/2} E_z(s-s_n)\cos k(s-s_n)\,ds, \\ S_n(k) &\equiv \frac{1}{V_0}\int_{s_n-l_n/2}^{s_n+l_n/2} E_z(s-s_n)\sin k(s-s_n)\,ds, \end{aligned} \tag{4}$$

where l_n is the length of gap n (this may be somewhat arbitrary depending upon cell geometry). The functions $T_n(k) = T_n(\beta)$ and $S_n(k) = S_n(\beta)$ can be numerically computed for particular cavity geometries using an electro-dynamics code such as SUPERFISH [3]. They essentially define the dynamics of each gap n.

In the above model the energy increase ΔW_n in a beam particle is applied impulsively at each gap n. In the longitudinal phase space (ϕ,W) any such change in energy ΔW must be accompanied by a corresponding change in phase $\Delta\phi$ to preserve the phase space volume (i.e., Liouville's theorem). This phase change $\Delta\phi_n$ for gap n is given by

$$\Delta\phi_n = \left[\phi(s) - \bar{k}_n s\right]_{s=s_n-l_n/2}^{s=s_n+l_n/2} = \int_{s_n-l_n/2}^{s_n+l_n/2}\left[k(s) - \bar{k}_n\right]ds \approx \frac{\partial k(W_n)}{\partial W}\int_{s-l_n/2}^{s+l_n/2}\Delta W(s)\,ds\,, \tag{5}$$

According to our initial approximation for ΔW_n, the expression for $\Delta W(s)$ in the interval $s \in [s_n{-}l_n/2, s_n{+}l_n/2]$ is given by

$$\Delta W(s) \approx qZ\int_{s_n-l_n/2}^{s} E_{z,n}(\sigma)\cos\left(\bar{k}_n\sigma + \phi_n\right)d\sigma\,. \tag{6}$$

Inserting this expression into that for $\Delta\phi_n$ and integrating by parts yields the following approximation for change in phase at the gap center:

$$\begin{aligned} \Delta\phi_n &\approx -\frac{\partial k(W_n)}{\partial W}qZV_0\left[S_n'(\bar{k}_n)\cos\phi_n + T_n'(\bar{k}_n)\sin\phi_n\right], \\ &= \frac{2\pi}{\lambda}\frac{qZV_0}{\beta_n^3\gamma_n^3 Amc^2}\left[S_n'(\bar{k}_n)\cos\phi_n + T_n'(\bar{k}_n)\sin\phi_n\right], \end{aligned} \tag{7}$$

where A is the atomic mass number of the beam particle, and we have used the chain rule $\partial k/\partial W = (\partial k/\partial\beta)(\partial\beta/\partial W)$ and the assumption that $E_{z,n}(s_n-l_n/2) = E_{z,n}(s_n+l_n/2) = 0$. These results are essentially a generalization of those seen in references [16] and [24].

A few remarks are in order concerning the longitudinal dynamics model we have just developed. For an "ideal gap" and a coordinate system with origin at the gap center, the ideal field $E_z(s)$ is an even function of s and thus $S_n(k) = 0$. There we recognize $T_n(k)$ as the usual transit time factor for an RF gap seen in the literature [24]. Note that we have implicitly assumed that the total axial field $E_z(z)$ can be decomposed into components $E_{z,n}(z)$ generated by each gap n, this decomposition may be somewhat arbitrary for some accelerating structures. We have also said nothing about how to compute any of the mid-gap values, such as the $\{\bar{k}_n\}$. Determining these mid-gap values can be nontrivial and techniques for doing so are described in reference [16]. The accuracy of Eqs. (3) and (7) becomes questionable when the velocity β does change significantly while the particle traverses the gap, for example, in the case of a low-energy electron beam. Clearly then the formulae are most accurate for heavier beam particles. For lighter beam particle species we may require a more accurate description of the longitudinal dynamics, such as that presented in reference [19].

Finally we point out that the maximum energy gain in gap n possible for a given klystron voltage V_0 is found by maximizing ΔW_n with respect to ϕ_n. This maximizing value of ϕ_n, say $\phi_{n,c}$ solves the equation $\partial\Delta W_n/\partial\phi_n = 0$ and is given by

$$\phi_{n,c} = -\arctan \frac{S_n(\bar{k}_n)}{T_n(\bar{k}_n)} . \tag{8}$$

Moreover, the energy gains are also functions of the mid-gap wave numbers $\{\bar{k}_n\}$ which are dependent upon the klystron phase ϕ_0 and amplitude V_0.

2.2.2 Sequence Attributes

Attribute	Description
`id`	Unique identifier string
`type`	Type identifier (see Table 11)
`pos`	Position within parent (m)
`len`	Length of sector (m)

Table 12: <sequence> element attributes

XAL Sequence Types		
Type Id	**XAL Class**	**Hardware Object**
CCL	`CLL`	Coupled-cavity linac
DTLTank	`DTLTank`	Drift-tube linac cavity
Bnch	`ReBuncher`	Re-buncher cavity
RF	`RfCavity`	Generic RF Cavity
SCLCavity	`SCLcavity`	Super-conducting linac

Table 11: sequence type identifier strings

Just as with the `<node>` element, a `<sequence>` element has XML attributes. These attributes describe the collective properties of the sequence object. There is one required XML attribute of every `<sequence>` element and three optional attributes. All these attributes are listed in Table 12. The required attribute is `id`, the unique identifier string for the `<sequence>`. Typically the value of this attribute is the accelerator sector that it represents, for example DTL1, DTL2, CCDTL1, etc. The optional attributes are `type`, `pos`, and `len`. The `type` attribute is the hardware type identifier string; this attribute is similar to the type attribute of the `<node>` element. If present, the type attribute informs XAL that the sequence represents a recognized composite element and, accordingly, XAL creates a special software object to represent it. The recognized type identifiers are listed in Table 11. The `pos` attribute is the offset (in meters) of the `<sequence>` from the *beginning* of its parent `<sequence>` object. If the current `<sequence>` object is a direct child of the top-level `<xdxf>` element then the `pos` attribute is the offset of within *entire* beamline described in the optics file. Thus, the value of `pos` is the absolute position of the `<sequence>` along the beamline. As with the `<node>` element, `pos` is the position of the center of the `<sequence>` object. The `len` attribute represents the total length of the `<sequence>`. This attribute together with the `pos` attribute are necessary to describe the drift regions between sequences. (The length of this drift is given by the difference between the `pos` attribute and the sum of the `pos` and `len` attributes of the previous `<sequence>`.)

In retrospect these attributes should have been required attributes since they are taken as zero if absent, potentially creating a precarious inconsistency. Consequently one should always provide the `pos` and `len` attributes regardless.

2.2.3 Sequence Attribute Buckets

Attribute	Description
`predecessors`	List of valid sequence `id`'s

Table 13: <sequence> attribute bucket parameters

Just as with `<node>` elements, `<sequence>` elements may have attribute buckets, which are listed in their `<attributes>` child section. Valid attribute buckets for a `<sequence>` object include the `<alignment>` bucket, which is the same as that for a `<node>` element listed in Table 4. Additional attribute buckets are the `<sequence>` bucket (an unfortunate choice in nomenclature) and the `<rfcavity>` bucket. The `<sequence>` attribute bucket is quite simple with only one attribute, `predecessors`, shown in Table 13. This value of this attribute is a space delimited list of `<sequence>` `id`'s which may precede the current `<sequence>` object in the beamline. In this manner, XAL knows that the current sequence may be placed after these predecessors to form viable beam paths. Referring to Excerpt 4 we see that the `<sequence>` attribute bucket has its `predecessors` attribute set to "`null`". This situation indicates that the sequence has no other sequences preceding it, that is, it is the beginning of the beamline. The last possible attribute bucket, `<rfcavity>`, is significantly more complex.

Attribute	Description
`amp`	Design amplitude (kV)
`phase`	Design phase (degrees)
`freq`	Resonant frequency (MHz)
`ampFactor`	Ratio of True/PV amplitude
`phaseOffset`	Difference True–PV phase
`TTFCoefs`	Even field TTF expansion
`TTFPrimeCoefs`	Even field TTF' expansion
`STFCoefs`	Odd field TTF expansion
`STFPrimeCoefs`	Odd field TTF' expansion
`TTF_endCoefs`	End cell expansion
`TTFPrime_EndCoefs`	End cell expansion
`STF_endCoefs`	End cell expansion
`STFPrime_endCoefs`	End cell expansion
`structureMode`	0 or π mode (0, 1)
`qLoaded`	Quality factor under load
`structureTTF`	TTF used in LLRF

Table 14: <rfcavity> attribute bucket parameters

A listing of all possible parameters of an `<rfcavity>` attribute bucket is given in Table 14. Clearly this attribute bucket applies only to `<sequence>` objects representing some type of accelerating RF structure. This structure is assumed to be composed of RF gaps, for which many of the `<rfcavity>` parameters apply. The first three parameters are rather straightforward, `amp`, `phase`, and `freq`. These are all design parameters for the cavity whose values indicate its drive amplitude (unfortunately in kV rather than Volts), its drive phase (in degrees), and its resonant frequency (unfortunately in MHz rather than Hz), respectively. The next two parameters, `ampFactor` and `phaseOffset`, are conversion parameters from actual values to process variable (PV) values. Specifically, the parameter `ampFactor` is the ratio of the true RF amplitude seen at the cavity input (i.e., V_0) versus the values of the amplitude process variables. For the `RfCavity` object (`<sequence> type` value "RF") these process variables are those identified with the channel handles `cavAmpAvg` and `cavAmpSet` (see Table 23 of Appendix B). The `phaseOffset` attribute is the difference between the phase at the entrance of the cavity (i.e., ϕ_0) and the klystron phase process variables. These process variables are identified by the XAL channel handles `cavPhaseAvg` and `cavPhaseSet` (see Table 23). Thus, `ampFactor` and `phaseOffset` are correction factors which may be utilized in case the amplitude and phase process variables differ from that seen at the cavity.

The next eight parameters of the `<rfcavity>` attribute bucket all contain modeling parameters characterizing the transit time functions $T(\beta)$ and $S(\beta)$, and their derivatives $T'(\beta)$ and $S'(\beta)$. Note that XAL takes the transit time function to be functions of normalized particle velocity β, rather than the wave number k as defined in Eqs. (4). This situation is simply more convenient, to convert from $T(k)$ and $S(k)$ to $T(\beta)$ and $S(\beta)$ we simply substitute Eq. (2) for k. Within the XAL model, each of the functions $T(\beta)$, $S(\beta)$, $T'(\beta)$ and $S'(\beta)$ are expanded as quadratic functions of β. Thus, the attributes `TTFCoefs`, `STFCoefs`, `TTFPrimeCoefs`, `STFPrimeCoefs` contain the coefficients of the expansions for $T(\beta)$, $S(\beta)$, $T'(\beta)$ and $S'(\beta)$, respectively. For example, given the entry TTFCoefs="*a b c*" where *a*, *b*, *c* are some real-number formats, then we take

$$T(\beta) = a + b\beta + c\beta^2 , \quad (9)$$

likewise with all the other transit time functions. The attributes `TTF_endCoefs`, `TTFPrime_EndCoefs`, `STF_endCoefs`, and `STFPrime_endCoefs` appear in case the transit time functions for the end cells in the RF cavity are different from those of all the initial cavity cells. If the cavity has identical transit time functions for each gap then these attributes values should be the same as those of their respective primary cell attributes.

The final three parameters of the `<rfcavity>` attribute bucket are `structureMode`, `qLoaded`, and `structureTTF`. The `structureMode` attribute indicates whether the cavity is being operated in normal mode or π mode (where the longitudinal field E_z changes sign at each cell boundary). A value of "0" indicates normal mode and a value of "1" indicated π mode. The value of `qLoaded` is the fully loaded quality factor of the cavity, that is, the quality factor including all external contributions including the beam. The final attribute, `structureTTF`, is the transit time factor used by the low-level RF system.

2.2.4 Sequence Connectivity: The <channelsuite> Element Revisited

Analogous to `<node>` elements, `<sequence>` elements may also have a `<channelsuite>` section. As with the `<node>` element, the `<channelsuite>` element contains a listing of all the signals associated with that `<sequence>` as a composite accelerator component. The syntax and format of a `<channelsuite>` element is identical to that of a `<node>`. Specifically, a `<channelsuite>` contains a list of `<channel>` elements, each of which describes a process variable connected to the parent `<sequence>`. Again, the attributes of a `<channel>` are listed in Table 10 and their descriptions, along with a description of `<channelsuite>`, are presented in Section 2.1.3. Typically, the process variables described by the `<channel>` entries of a `<sequence>` are those for an RF cavity, since sequences most often describe RF cavities. Thus, the channel handles for the `RfCavity` object in Table 23, Appendix B, are the one most often seen. This is the situation seen in Excerpt 4.

2.3 Combination Sequences: The <comboseq> Element

The initial section of the XAL optics file usually contains a set of `<comboseq>` blocks, each containing a list of `<sequence>` elements. This feature is primarily a convenience to XAL users where `<sequence>` objects may be grouped together to form larger composite lattice objects making sense in the context of the current accelerator complex. For example, in Excerpt 2 we see a `<comboseq>` entry describing the aggregation of seven `<sequence>` objects consisting of a Medium Energy Beam Transport (MEBT) plus six Drift-Tube Linac (DTL) tanks. The composite object is then called "MEBT-DTL". There it is assumed that the "MEBT-DTL" composite is frequently analyzed in this configuration.

As mentioned above, a `<comboseq>` element contains a consecutive listing of `<sequence>` elements. The only attribute of a `<comboseq>` element is `id`, as shown in Table 15. The value of `id` is the unique identifier string for the described combination sequence. The child elements of a combination sequence, that is `<sequence>` elements, also have only attribute, also called `id`. The value of `id` in this case refers to the unique sequence identifier of the sequence within the combination sequence.

Attribute	Description
`id`	identifier of combination sequence

Table 15: <comboseq> element attributes

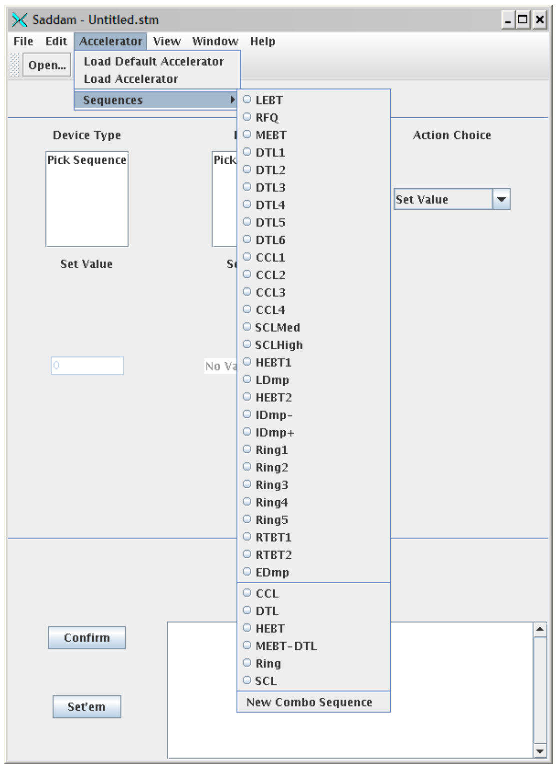

Figure 9: example application menu with combination sequences

2.3.1 Working with Sequences and Combinations Sequences

Figure 9 shows an example XAL application built from the XAL Application Framework. The Application Framework provides the `Accelerator` menu from which any of the sequences or combination sequences described in the optics file can be selected. (The `Accelerator` menu also enables you to load the main XAL file describing the accelerator complex.) As seen in Figure 9, the `Accelerator/Sequences` selection brings up a child menu of all the sequences and combination sequences described in the optics file. The user need only select the accelerator section that he or she wants to analyze. Also shown in the figure is the option of creating new combination sequences, which can then be analyzed, as well as those stored in the optics file for later use.

```
<powersupplies>
  <ps type="main" id="MEBT_Mag:PS_DC">
    <channelsuite name="pssuite">
      <channel handle="I" signal="MEBT_Mag:PS_DC:I"/>
      <channel handle="I_Set" signal="MEBT_Mag:PS_DC:I_Set"/>
      <channel handle="fieldSet" signal="MEBT_Mag:PS_DC:B_Set"/>
      <channel handle="psFieldRB" signal="MEBT_Mag:PS_DC:B"/>
      <channel handle="cycleState" signal="MEBT_Mag:PS_DC:cycleState"/>
      <channel handle="cycleEnable" signal="MEBT_Mag:PS_DC:cycEnable"/>
    </channelsuite>
  </ps>
  <ps type="trim" id="CCL_Mag:ShntC_QTV309">
    <channelsuite name="pssuite">
      <channel handle="trimSet" signal="CCL_Mag:ShntC_QTV309:B_Set"/>
      <channel handle="trimRB" signal="CCL_Mag:ShntC_QTV309:B"/>
      <channel handle="trimI_Set" signal="CCL_Mag:ShntC_QTV309:I_Set"/>
      <channel handle="trimI" signal="CCL_Mag:ShntC_QTV309:I"/>
      <channel handle="cycleState" signal="CCL_Mag:ShntC_QTV309:cycleState"/>
      </channelsuite>
    </ps>
  …
  </powersupplies>
```

Excerpt 5: example <powersupplies> entry in XAL optics file

2.4 Power Supplies: The <powersupplies> element

The optics file contains a single instance of the `<powersupplies>` element, although this single element is typically a large listing. This element usually occurs at the end of the optics file, after all the `<sequence>` elements are defined. Located under the `<powersupplies>` element is a listing of all the power supplies used in the accelerator description. The direct children of `<powersupplies>` are `<ps>` elements, which describe the actual power supplies. Each `<ps>` element contains a `<channelsuite>` entry listing all the channels connected to that supply. An example of the this syntax for listing the power supplies of an accelerator is shown in Excerpt 5.

Although the `<powersupplies>` entry is not defined in the `xdxf.dtd` file, it is important that this information be included in the optics file, and with the syntax shown in Excerpt 5. This mechanism is how XAL binds beamline devices in the `<sequence>` entries to their power supply process variables. As mentioned in Section 2.1.2 this is also the method for defining bulk power supplies for multiple devices, such as that for quadrupole magnets. The power supply is defined once in the `<powersupplies>` section, and the magnets each reference it using the `<ps>` element within the magnet `<node>` definition.

Attribute	Description
`id`	UID string of power supply
`type`	supply type ("main" or "trim")

Table 16: <ps> element attributes under <powersupplies>

The `<powersupplies>` element itself has no XML attributes. It simply marks the listing of power supply entries. Different from its context under a `<node>` element (see Section 2.1.2), the `<ps>` element here has two attributes, `id` and `type`. The value of `id` is the unique identifier string of the power supply while the value of `type` is currently either "`main`" or "`trim`". If `type` equals "`main`" then the software object representing it in XAL is of Java class type `MainMagnetSupply`. If `type` equals "`trim`" then the object representing the supply is of class type

`MagnetTrimSupply`. The remaining part of the `<ps>` entry describes all the process variables connected to the power supply. As shown in Excerpt 5 this action is accomplished with a single `<channelsuite>` element. The `<channelsuite>` element here is the same as that for a `<node>` element described in Section 2.1.3. The channel suite contains a listing of each `<channel>` object connected to the power supply, each channel being described by its XAL handle and the signal name of the process variable. The channel handles used here are typically those of the last three entries of Table 23, Appendix B, specifically, the classes `MagnetPowerSupply`, `MainMagnetSupply`, and `MagnetTrimSuppply`.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE timing SYSTEM "xdxf.dtd">
<timing>
    <channelsuite name="timingsuite">
        <!-- beam trigger PV: 0=Trigger, 1=Counting -->
        <channel handle="trigger" signal="ICS_Tim:Gate_BeamOn:SSTrigger"/>
        <!-- beam trigger mode PV: 0=Continuous, 1=Single-shot -->
        <channel handle="mode" signal="ICS_Tim:Gate_BeamOn:SSMode"/>
        <!-- specify how many beam pulse(s) -->
        <channel handle="countDown" signal="ICS_Tim:Gate_BeamOn:SSCountDown"/>
        <!-- readback while it's counting down -->
        <channel handle="count" signal="ICS_Tim:Gate_BeamOn:SSCount"/>
        <!-- readback of overall rep rate -->
        <channel handle="repRate" signal="ICS_Tim:Gate_BeamOn:RR"/>
        <!-- beam on event -->
        <channel handle="beamOnEvent" signal="ICS_Tim:Util:event36"/>
        <!-- beam on event counter -->
        <channel handle="beamOnEventCount" signal="ICS_Tim:Util:event36Count"/>
        <!-- diagnostic demand event -->
        <channel handle="diagnosticDemandEvent" signal="ICS_Tim:Util:event45"/>
        <!-- slow (1 Hz) diagnostic event -->
        <channel handle="slowDiagnosticEvent" signal="ICS_Tim:Util:event46"/>
        <!-- fast (6 Hz) diagnostic event -->
        <channel handle="fastDiagnosticEvent" signal="ICS_Tim:Util:event47"/>
        <!-- Machine mode -->
        <channel handle="machineMode" signal="ICS_Tim:MPS_Mode:MachMode"/>
    </channelsuite>
</timing>
```

Excerpt 6: XAL Timing File

3 Timing Process Variables: The <timing_source> File

XAL supports the inclusion of several high-level timing signals that the application developer can monitor. A separate XML file, called the *timing file*, defines all the timing signals available to an XAL developer. As seen in Excerpt 1, this file is tagged `<timing_source>` in the main XAL configuration file `main.xal`. The `url` attribute of the `<timing_source>` element points to the timing file. Although it is likely that these timing signals seldom will be used, they are available and we outline the configuration method. This configuration

Handle	Description
trigger	Trigger flag – 0 triggered, 1 counting
mode	Trigger mode – 0 continuous, 1 single
countDown	Number of beam pulses
count	Pulse number during countdown
repRate	Overall repetition rate
beamOnEvent	??
beamOnEventCount	Beam on event counter
diagnosticDemandEvent	??
diagnosticDemandEventCount	Diagnostic demand event counter
slowDiagnosticEvent	Slow (1 Hz) diagnostic event
slowDiagnosticEventCount	Slow diagnostic event counter
fastDiagnosticEvent	Fast (6 Hz) diagnostic event
fastDiagnosticEventCount	Fast diagnostic event counter
ringFrequency	Readback of ring frequency (MHz)
machineMode	Machine mode ?

Table 17: timing channel handles

mechanism is much simpler than that for the optics configuration; it basically consists of a single `<channelsuite>` entry.

An example timing file is shown in Excerpt 6. The first two lines are the standard XML statements designating the version, character encoding, name of the document, and the DTD file associated with the document (which, again, is `xdxf.dtd`). On the next line starts the root element, `<timing>`. There are no attributes to this element. It simply contains one child element, a `<channelsuite>` element, listing all the timing channels of the accelerator. The `<channelsuite>` element is used here just as it is in Section 2.1.3. However, here we are directly populating an instance of the Java class `TimingCenter` with the listed `Channel` objects. As mentioned in Section 1.4, the `TimingCenter` object is attached to the main `Accelerator` object. Thus, to acquire a timing signal, the developer must first reference the `TimingCenter` through the `Accelerator` object, then work with the appropriate channel handle.

The supported channel handles are defined in the `TimingCenter` class. Table 17 lists all the channel handles currently supported by the `TimingCenter` class along with a description of each. However, since we work directly with `Channel` objects in this case, it is possible to define your own channel handle within the timing file and request it directly with the method `TimingCenter.getChannel(String channelHandle)`.

4 Auxiliary Data: Using the <tablegroup_source> Element

Most of the high-level machine hardware in an accelerator system can be described in the XAL optics file. However, not all information required by XAL is contained in this file. A prime example is the beam data that XAL needs to describe and *simulate* the beam. These data are not included in the optics file because they do not describe the machine; the optics file is formatted strictly as a machine description mechanism. Any data not supported by the XAL optics file may be included using a separate mechanism called a *table group*.

Attribute	Description
`name`	ID within accelerator EditContext
`url`	URL of XML data file

Table 18: <tablegroup_source> element attributes

A table group allows arbitrary data to be made available to application users and developers within the XAL framework. It is essentially a miniature database, the data itself being contained in tables within an XML file. The structure of this data file is described in Section 4.1. The location of the data file is identified in the main XAL configuration file (`main.xal`) using a `<tablegroup_source>` element entry. Such an entry is demonstrated in Excerpt 1 for the case of the modeling parameters. In the excerpt we see the two attributes of the `<tablegroup_source>` entry, `name` and `url`, which are listed in Table 18 with a description. The `name` attribute contains the unique string identifier of the table group. This identifier string is used to reference the data tables forming the table group. XAL has a special container called an *edit context* which contains all the table groups listed in the `main.xal` file (discussed below). The value of the `url` attribute is the URL of the table group's data file. The data within this file is loaded and converted to data tables when the XAL configuration file is read. Referring to Excerpt 1, the URL of the table group named "modelparams" is `model.param`, which is assumed to be in the same directory as `main.xal`.

The edit context object referred to above is of Java type `EditContext`. Within XAL, it is a singleton class, meaning there is only one `EditContext` instance within the entire XAL framework. The edit context singleton is accessed from the main `Accelerator` object with a call to the method `Accelerator#editContext()`. From there, individual table groups may be retrieved with the accessor `EditContext#getTablesForGroup(String name)`. Programmatically, a table group is actually a Java `Collection` of `DataTable` objects. Thus, the primary objects of a table group are `DataTable` instances. Analogous to tables in a database, each `DataTable` is composed of a set of related data records. However, unlike a database, application developers may register with the edit context to receive event notification whenever data in the tables, or the tables themselves, are changed. The details of this mechanism are, however, beyond our current scope.

4.1 Basic Form of a <tablegroup> Element

For any table group specified by a `<tablegroup_source>` element, the data file is an XML file with the document type definition file `tablegroup.dtd`. The contents of `tablegroup.dtd` are listed in

Appendix C. Each table group file consists of one `<tablegroup>` element with one or more `<table>` children. There are no restrictions on the number of `<table>` elements within a table group file.

Excerpt 7 lists a simple table group data file. The first two lines are boiler plate XML as seen before, the second line indicating that the DTD file is `tablegroup.dtd`. The third line is the singleton `<tablegroup>` element entry. Within this particularly simple table group, there is only one `<table>` entry (recall that there can be many). The purpose of this particular table is to list the properties of several different particles. Each `<table>` element has two attributes: `name`, which is required, and `recordClass`, which is optional. These attributes are listed in Table 19 along with brief descriptions. The `name` attribute contains the table's unique identifier string. This identifier facilitates access to the particular data table within the table group. The `recordClass` attribute specifies the Java class type of the records within the data table. In Excerpt 7 we see that the value there is `GenericRecord`, which is, in fact, its default value. The Java class `GenericRecord` is suitable for most all table records within a table and it is unlikely that one would need to implement a specialized class here. Thus, one seldom sees the `recordClass` attribute actually specified in the `<table>` element entry.

Attribute	Description
`name`	Table ID within accelerator EditContext
`recordClass`	Java class of data record

Table 19: <table> element attributes

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE particles SYSTEM "tablegroup.dtd">
<tablegroup>
    <table name="species" recordClass="gov.sns.tools.data.GenericRecord">
        <schema>
            <attribute type="java.lang.String" name="name" isPrimaryKey="true"/>
            <attribute type="java.lang.Double" name="charge" isPrimaryKey="false"/>
            <attribute type="java.lang.Double" name="mass" isPrimaryKey="false"/>
        </schema>
        <record charge="-1.0" name="ELECTRON" mass="5110000.0"/>
        <record charge="1.0" name="PROTON" mass="9.38272E8"/>
        <record charge="-1.0" name="HMINUS" mass="9.393014E8"/>
    </table>
</tablegroup>
```

Excerpt 7: example table group data file

As seen in the table group DTD file of Appendix C and in Excerpt 7, there are two types of child elements for a `<table>` entry, `<schema>` and `<record>`. According to the DTD specification, there is exactly one `<schema>` element, which then may be following by any number of `<record>` elements. The `<schema>` element specifies the "schema" of the table, specifically, the structure of the data records contained in the current table. The `<record>` elements contain the actual data of the data table. There are no child elements of a `<record>` element, only attributes. The attributes of the `<record>` element are described in the table's `<schema>` entry, and each represents a field of the data record.

A table's `<schema>` element has no XML attributes, only child elements which are all of type `<attribute>`. This set of `<attribute>` elements describes completely the structure of the data records within the table. Specifically, each `<attribute>` entry identifies a field in the data record. The `<attribute>` element has no children, only attributes. The four (XML) attributes of an `<attribute>` element are listed in Table 20. They are `name`, `type`, `isPrimaryKey`, and `defaultValue.` All attributes are required except the `defaultValue` attribute, which may be omitted. The `name` attribute contains the name of the field in the data record. The `type` attribute specifies the Java class type for the field, that is, how its value is stored programmatically within the record. This specification is necessary because attributes of XML elements are all stored internally as strings and it is necessary to convert the string values to their proper representation within XAL. The attribute `isPrimaryKey` is a Boolean flag (value "true" or "false") specifying whether or not the field is a primary key of the record.

Attribute	Description
`name`	Name of the record field
`type`	Java class type of record field
`isPrimaryKey`	Record field's primary key flag
`defaultValue`	Default value of the record field

Table 20: <attribute> element attributes

Programmatically, table records are retrieved by the values of their primary keys. Thus, to be able to select a particular record in a table, one must know the value for a primary key within the record. Each record must then contain at least one primary key. Thus, at least one `<attribute>` element of the parent `<schema>` must have its `isPrimaryKey` attribute set to "true". This requirement follows because the records of each table are indexed according to the primary key values (so that they can be retrieved as such). There may be multiple primary keys within a record; we require that there is at least one. The final attribute, `defaultValue`, is the default value of the field. The record field will take on this value if not explicitly specified in the table records. Although it is impossible to enforce this condition with a DTD, the only required attribute of a `<record>` entry is the primary key; this value must be present or XAL will throw an exception. All other attributes are optional. Consequently, if a `<record>` attribute (field) is omitted, then its value is given by the value of `defaultValue` described in the `<schema>` definition.

Referring back to Excerpt 7, we see that the table with name "species" has a schema specifying three fields for each record of the table, `name`, `charge`, and `mass`. The `name` field is of Java type `String`, while the `charge` and `mass` are both of Java type `Double`. The `name` field is the primary key of the table and, thus, the table records must be retrieved with the value of this field. After the `<schema>` entry in the table, we find the actual records of the table. In this case of Excerpt 7 there are three records, one named "ELECTRON" representing an electron, one named "PROTON" representing a proton, and the last named "HMINUS" representing an H^- ion. The mass of each particle is in electron-Volts (i.e., the rest energy of the particle) while the charge is normalized to the unit charge $e \approx 1.602{\times}10^{-19}$. Since the data stored in table groups is user data, he or she may use any units or table formatting that seems appropriate, so long as it conforms to the schema of a table group.

4.2 The Model Parameters <tablegroup> Entry

There is, at present, one table group that is particular to XAL, that is, it must take a particular format. This special table group is that containing the modeling parameters for a beam under simulation by the XAL online model [2]. An example of this table group is shown in Appendix D, it is that for the SNS machine at Oak Ridge, Tennessee. Note that there are five separate tables in this table group, they have the names "species", "beam", "adaptivetracker", "twiss", and "location". Although the table group of Appendix D is an example, all model parameter table groups must follow the same schema exactly, only the data within the table records may differ. That is, Appendix D is essentially a template for all model parameter table groups.

4.2.1 The Species Table

Excerpt 8 lists the schema of the XAL model-parameters table group. Basically, it is Appendix D without the `<record>` element entries. As mentioned above, there are five tables. The first of these, "`species`", is the same as that discussed in the preceding section. The table describes all the particle species composing the beam within the accelerator. Each species is given a name, that is, the value of the `name` attribute, for which it can be referenced in the following tables. The `name` field is the primary key of the table and, thus, the table records must be retrieved with the value of this field. The value of the `mass` attribute is actually the rest energy E_r of the particle where $E_r = m/c^2$ (m being particle mass). Thus, this value should be in electron-Volts. The `charge` attribute is the particle charge normalized to the unit charge $e \approx 1.602{\times}10^{-19}$. At present, the "`species`" table records are referenced only within one other table, the "`location`" table.

4.2.2 The Beam Table

The next table, called "`beam`", describes the collective properties of the particle beam being modeled. Each record within the table has three fields, `name`, `I`, and `Q`. The `name` attribute is the primary key and, consequently, the mechanism by which the table records are reference in the other tables. The value of the `name` field is user defined; it should be a unique identifier string which makes sense in the simulation context. The `I` and `Q` fields are the beam current (in Amperes) and beam charge (in Coulombs), respectively. In an RF cavity these fields are related by the formula

$$(10) \qquad I = fQ\,,$$

```
<tablegroup>
  <table name="species">
    <schema>
      <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="mass" type="java.lang.Double"/>
      <attribute isPrimaryKey="false" name="charge" type="java.lang.Double"/>
    </schema>
  </table>

  <table name="beam">
    <schema>
      <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="I" type="java.lang.Double"/>
      <attribute isPrimaryKey="false" name="Q" type="java.lang.Double"/>
    </schema>
  </table>

  <table name="adaptivetracker">
    <schema>
      <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="errortol" type="java.lang.Double" defaultValue="1.0E-3"/>
      <attribute isPrimaryKey="false" name="initstep" type="java.lang.Double" defaultValue="0.01"/>
      <attribute isPrimaryKey="false" name="maxstep" type="java.lang.Double" defaultValue="0.0"/>
      <attribute isPrimaryKey="false" name="norm" type="java.lang.Integer" defaultValue="0"/>
      <attribute isPrimaryKey="false" name="order" type="java.lang.Integer" defaultValue="2"/>
      <attribute isPrimaryKey="false" name="slack" type="java.lang.Double" defaultValue="0.05"/>
      <attribute isPrimaryKey="false" name="maxiter" type="java.lang.Integer" defaultValue="100"/>
    </schema>
  </table>

  <table name="twiss">
    <schema>
      <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
      <attribute isPrimaryKey="true" name="coordinate" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="alpha" type="java.lang.Double"/>
      <attribute isPrimaryKey="false" name="beta" type="java.lang.Double"/>
      <attribute isPrimaryKey="false" name="emittance" type="java.lang.Double"/>
    </schema>
  </table>

  <table name="location">
    <schema>
      <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="species" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="W" type="java.lang.Double"/>
      <attribute isPrimaryKey="false" name="elem" type="java.lang.String" defaultValue=""/>
      <attribute isPrimaryKey="false" name="s" type="java.lang.Double" defaultValue="0"/>
      <attribute isPrimaryKey="false" name="t" type="java.lang.Double" defaultValue="0"/>
    </schema>
  </table>
</tablegroup
```

Excerpt 8: model parameters table group schema

where *f* is the frequency (in Hertz) of the RF power. Thus, for each separate beam, one must prescribe a name (identifier) then the bunch charge and beam current. Currently, there is only one beam record used, it specifies the beam throughout the entire machine.

4.2.3 The Adaptive Tracker Table

The third table, named “`adaptivetracker`”, contains numerical tuning parameters for the space charge algorithm used in the RMS envelope simulation. The XAL online model uses an adaptive integration procedure for the simulation. Consequently, there are several numerical parameters that can be used to fine tune this integration process. Records in this table contain the data fields `name`, `errortol`, `initstep`, `maxstep`, `slack`, `maxiter`, `norm`, and `order`. Below we list each `parameter` with a description of its function. For each parameter, we also offer default values which appear to provide reasonable performance in most cases.

- name: This is the unique identifier of the data record, and the primary key. One should set this attribute value to the <sequence> name of the beamline where the data record applies. That is, the tuning parameter values in the current record will be used for simulation in the given beamline described by the sequence id.
- errortol: The integration algorithm is designed to maintain a specific accuracy in the computed solution. Moreover, it is designed to keep the integration step size *h* as large as possible such that $\|\boldsymbol{\sigma}_{\text{sim}} - \boldsymbol{\sigma}_{\text{soln}}\| \leq$ errortol at each step, where $\boldsymbol{\sigma}_{\text{sim}}$ is the simulated moment matrix, $\boldsymbol{\sigma}_{\text{soln}}$ is the "exact" moment matrix, and $\|\cdot\|$ is a suitable matrix norm (see norm below). This technique provides the fastest solution time while maintaining the given error tolerance. A reasonable default value of errortol is 10^{-5}.
- initstep: The initial step size (in meters) used to start the adaptive integration algorithm. The integration algorithm continually adjusts the actually step size *h* to maintain the error tolerance specified by the parameter errortol. Choosing an initial step too small will marginally slow the solution time, while choosing a value too large will not significantly affect the solution time. A reasonable default value of initstep is 0.01 (1 cm).
- maxstep: Maximum allowable step size (in meters). For whatever reason, it is possible for the user to specify a maximum step size for the adaptive stepping algorithm. This value will prevent the algorithm from taking any step sizes *h* greater than maxstep. To turn off this feature set maxstep = "0". Under normal conditions this feature is not used, that is, the default value of maxstep is 0 (no maximum step size).
- slack: Slack tolerance parameter. For each integration step in the simulation, a new value of the step size *h'* is computed for the next step. If *h'* is smaller than the previous step *h*, the current integration step must be rolled back and the solution recomputed with new step size *h'* in order to maintain the solution tolerance errortol. However, if *h'* is only marginally smaller than *h* (perhaps due to noise or rounding errors), this backtracking can be a significant waste of CPU time. Thus, we only change the integration step size if $|h'-h|/h >$ slack. (The reason for the absolute value is that we also must re-compute the element sub-transfer matrix exp($h\mathbf{A}$) if *h* is changed in either direction.) Thus, slack provides some backlash before the adaptive stepping is triggered. Consequently, slack can save significant CPU time, however, choosing a value to large will compromise the accuracy of the solution and the solution tolerance errortol. A reasonable default value of slack is 5% (slack="0.05").
- maxiter: Maximum number if integration steps allowed through a beamline element. Because of the adaptive stepping procedure, it is possible that the algorithm may stall, choosing smaller and smaller step sizes *h*. This is a pathological situation that may occur, for example, if the errortol parameter is set too small. To prevent the simulation from entering this situation, the maxiter parameter specifies the maximum number if integration steps the algorithm will allow through any beamline element before stopping and throwing an exception. A reasonable value for maxiter is 100.
- norm: The matrix norm used in the adaptive stepping algorithm. There are three possible values for norm, which are the three most common Lesbeque norms: "0" indicates the l_∞ norm where the matrix norm is taken as the largest matrix element, "1" indicates the l_1 norm where the norm is the summation of the absolute value of all the matrix elements, and "2" indicates the l_2 norm where the norm is the square-root of the sum of matrix element squares. Any of the three norms work equivalently, the most popular seems to be the l_∞ norm (i.e., norm="0").
- order: The order of integration. The integration algorithm will support either a first-order integration scheme (i.e., Euler integration) or a second-order method. The integration order is specified with an order value of 1 or 2, respectively. We do not gain much by integrating to higher orders, due to the way space charge effects are handled. It appears best to use the second-order integration scheme as a default, that is, order="2".

4.2.4 The Twiss Table

The next table in the model-parameters table group is called "twiss". This table contains the Twiss, or Courant-Snyder, parameters of the beam at various locations along the beamline. The fields of each record are `name`, `coordinate`, `alpha`, `beta`, and `emittance`. Clearly, the `alpha`, `beta`, and `emittance` fields are the Courant-Snyder parameters for the beam ellipse in a single phase plane. The units for `beta` are meters/radian while `alpha` is unitless. The `emittance` field expects the RMS emittance for the beam in units of meter-radians (even for longitudinal emittances). The `name` field is similar to that of the previous table. Its value is the identifier string for the record and, once again, we set the value of this field to the name of the `<sequence>` describing the machine sector where this set of Courant-Snyder applies. However, unlike the previous table, this value need not be unique. In fact, there should be three records having the same value for the `name` field (see below). The `coordinate` field specifies the phase plane to which the Courant-Snyder parameters belong. For the model parameters table group to work properly, one must set the value of this field to either "`x`", "`y`", or "`z`", specifying the *x*, *y*, and *z* phase planes, respectively.

It is important to note here that the "`twiss`" table has two primary key fields. Specifically, `name` and `coordinate` are both primary keys. Thus, each record is indexed according to the value of both the `name` and `coordinate` fields. To expand on this condition, the `name` attribute specifies the `<sequence>` where the Courant-Snyder parameters apply, specifically, the entrance to that `<sequence>`. The `coordinate` field then specifies the phase plane for the particular Courant-Snyder parameters. Thus, for each `<sequence>` where one wishes to begin a simulation, there must be three records whose `name` field has the value of the `<sequence>` identifier, each of these records specifying the Courant-Snyder parameters for a separate phase plane (*x*, *y*, or *z*). In this manner each record can be uniquely identified by the values of the two primary keys.

4.2.5 The Location Table

The last table is the "`location`" table. This table correlates all of the preceding tables. It tells XAL which beam to use at each `<sequence>` location within the beamline. Currently, the table is somewhat incomplete, in that it truly only specifies which particle species to use (i.e., records from the "`species`" table), along with several other parameters. In the future it may be necessary to include records from the "`beam`" table as well.

There are six fields in each record of the "`location`" table, `name`, `species`, `W`, `elem`, `s`, and `t`. Once again, the `name` field specifies the `<sequence>` to which the record applies, that is, the value of the `name` field is the `<sequence>` identifier string. Since there is only one primary key, this value must be unique within the table. The `species` field indicates which particle species, from the "`species`" table, to use with the `<sequence>`. Thus, the value of the `species` field is the unique identifier string of the desired species record in the "`species`" table. The `W` field specifies the beam kinetic energy at the entrance to the `<sequence>` in electron-Volts. The `elem` field, if included, indicates that the simulation should begin at that particular element within the sequence, rather than the sequence entrance. Thus, the value of the `elem` field should be the unique identifier string of the `<node>` entry where the simulation is to begin. The `s` field specifies the starting value of the path length parameter at the sequence entrance. It has a default value of zero, which typically need not be changed for most all simulations. The final field, `t`, specifies the starting time of the simulation. Once again this field has a default value of zero which is fine for typical simulation requirements.

5 Particle Beam Representation: Probe Files

As indicated in the introduction, there is one additional data file supported by XAL, the *probe file*. Although its practical use has been lessened by the model-parameters table group discussed in Section 4.2, the probe file continues to be supported. This file is an XML file used only by the XAL online model, it is typically small. It describes the particle beam, or rather aspects thereof, to be simulated. The XAL online model supports several different types of beam simulation, for example, single-particle, RMS envelope, and transfer map generation. Each of these different types of simulation requires differing data describing the particular aspects of the beam we are modeling. Thus, there is no rigid format for the probe file and, consequently, no DTD specification. There is a general format where the particulars depend upon the type of simulation desired by the user. We cover this general format below.

Because the probe file is currently being deprecated by the model-parameters file, our presentation is more laconic then for the previous files. However, it is important to cover the basic probe file format because the model-parameters table group, in its current form, is formatted primarily to support single-particle simulation and RMS envelope simulation. Thus, at least at present, it may even be necessary to use a probe file for a different simulation type.

To understand the basic structure of a probe file it is helpful to appreciate the basic architecture of the XAL online model (for a more detailed exposition see [1] and [2]). The online model is built upon the Element/Algorithm/Probe design pattern conceived by Malitsky and Talman [18]. In this framework, the particle beam simulator is composed of three separate software components, 1) the *Element* representing the machine hardware, 2) the *Algorithm* encapsulating the numerical simulation technique, and 3) the *Probe* which represents some aspect or aspects of the beam under study. In this manner, the same machine representation can support multiple simulation strategies. This fact alone is a tremendous savings in development effort, since such a large part of the simulation is devoted toward proper representation of the machine hardware. The Element, or machine description, is taken from the optics file which has already occupied most of this presentation. The Algorithm and Probe are identified by the probe file. Specifically, the probe file tells the XAL online model which type of beam simulation to perform and which algorithm to use during the simulation.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<probe time="Feb 9, 2004 2:42:00 PM" type="gov.sns.xal.model.probe.EnvelopeProbe">
    <comment text="XAL model probe representing rms envelope state at HEBT entrance"/>
    <algorithm type="gov.sns.xal.model.alg.EnvTrackerAdapt" ver="2">
        <tracker debug="false" update="1"/>
        <adapt errortol="1.0E-5" initstep="0.031278684521382855"
       maxstep="0.0" norm="0" order="2" slack="0.05" maxiter="50"/>
    </algorithm>
    <state id="" type="gov.sns.xal.model.probe.traj.EnvelopeProbeState">
        <location W="2500000.0" elem="" s="0.0"/>
        <species Er="9.393014E8" q="-1."/>
        <beam I="0.020" Q="4.96894E-11"/>
        <envelope
            alphaX="-1.962" betaX="0.183" emitX="2.73e-6"
            alphaY="1.768" betaY="0.161" emitY="2.73e-6"
            alphaZ="0.0196" betaZ="0.5844" emitZ="3.8638e-6"/>
    </state>
</probe>
```

Excerpt 9: example envelope probe file

5.1 The <probe> Element

After the initial perfunctory XML entries, a probe file begins with the singleton `<probe>` element. This is the root of the probe file. This situation is shown in the example probe file of Excerpt 9. There are four possible attributes for a `<probe>` element, `type`, `id`, `time`, and `author`; only the `type` attribute is required. These attributes are listed in Table 21. The `type` attribute is the Java class type (full package name) of the probe object. This attribute is required so that XAL knows to instantiate the correct probe type. Referring to Excerpt 9 we see that the value there is "`gov.sns.xal.model.probe.EnvelopeProbe`", which is the class type representing the RMS envelopes of a particle beam. The `id` attribute is the identifier string of the probe instance within XAL. This identifier does not need to be unique; it is simply a convenience for the user in order to refer to a specific probe object. The `time` attribute is a time stamp for the probe file, to maintain versioning if desired. The `author` attribute facilitates a reference to the file's author for documentation purposes.

Attribute	Description
`type`	Java class type of probe object
`id`	Identifier string of probe object
`time`	Time stamp of probe file
`author`	Author of probe file

Table 21: <probe> element attributes

There are three child elements of the root `<probe>` element, all singletons. First, there is an optional `<comment>` element which allows users to include any additional information concerning the contents of the file. The next entry is the `<algorithm>` element, which describes the simulation algorithm used to propagate the probe down the beamline. The architecture of XAL allows different simulation algorithms to be used with the same beam probe (not simultaneously). The last entry is the `<state>`

element which describes the internal state variables of the beam probe. Every beam probe maintains a set of state variables describing the aspect of the beam it represents, for example, the moments of the beam up to second order in the case of the RMS envelope probe. The probe requires the initial value of these state variables to begin the simulation. We cover each child element of `<probe>` below.

5.2 File Comments: The <comment> Element

We can annotate the probe file with additional user data using the `<comment>` element. These comments will then also be available programmatically and at run time within the XAL environment. As seen in Excerpt 9 the text of the comment is included as the value of the `text` attribute. In that case the probe file is declared to be the initial state of an RMS envelope probe at the entrance to a HEBT section. There are two more supported attributes for a `<comment>` entry which are not shown in the excerpt, `author` and `date`. The `author` attribute allows one to identify the author of the comment; the `date` attribute is a date stamp for the comment.

5.3 Specifying the Simulation Algorithm: The <algorithm> Element

Every beam probe needs an algorithm; this is an object which knows how to propagate the probe down the beamline. Algorithm classes are built for specific probe classes, but probe classes are independent of algorithm classes. Thus, it is possible for a single probe class to be associated with many different algorithm classes, but each algorithm understands only one probe. (Currently, most probes have only one type of algorithm associated with them.) The `<algorithm>` element specifies the algorithm class to be used with the probe instance, and any numerical parameters needed by the algorithm.

The `<algorithm>` element has two attributes, `type` and `ver`. The `type` attribute is required and specifies the algorithm class to use for the simulation. Unfortunately, its value is the Java class type of the algorithm object (the fully qualified name, package included). This situation is different from the previous version of XAL where the `type` value was the value of this attribute is actually the Java string returned by the `getType()` method of the algorithm class. (All algorithm classes are derived from the base class `gov.sns.xal.model.alg.Tracker`, so existence of this method is guaranteed.) Now, the string returned by this method is the class name of the algorithm with the package prefix. For example, in Excerpt 9 the value of `type` is `gov.sns.xal.model.alg.EnvTrackerAdapt`. The `ver` attribute is optional and contains a version stamp for the algorithm. This attribute is available to maintain backward compatibility in case any major changes are made to existing algorithm classes.

The `<algorithm>` element has at least one child element, `<tracker>`. Other elements pertaining to the particular algorithm class will likely be present. For example, the `<algorithm>` entry in Excerpt 9 contains the child `<adapt>` which specifies numerical parameters for the adaptive space charge algorithm. We discuss the `<tracker>` element, then the `<adapt>` element of the RMS envelope probe since it is the most common additional element.

5.3.1 The <tracker> Element

In the parenthetical remark above we mentioned that all algorithm classes in XAL are derived from a common base class, `gov.sns.xal.model.alg.Tracker`. The `<tracker>` element specifies parameters of this base class and, thus, parameters common to all algorithm classes. There are currently only two attributes of this element, `debug` and `update`, which are listed in Table 22. The `debug` attribute is a Boolean debugging flag. When the value is set to “`1`” the algorithm object will send debugging information to the console. Under normal operation this value should be set to “`0`” indicating that no debugging information is generated. The `update` attribute is more complex, it specifies the manner in which the probe’s trajectory information is saved throughout the beamline.

Attribute	Description	
`debug`	Debugging flag (“0” or “1”)	
`update`	Trajectory updating scheme	
	0	Never
	1	Always
	2	Element exit
	4	Element entrance
	6	Entrance and exit

Table 22: <tracker> attributes

To be useful, the probe object must save its past states as it propagates down the beamline. We call these past states of the probe the *probe trajectory*, and it is treated as a single object within the XAL framework. There are several instances where the probe must compute many more states than is necessary to store. For example, to maintain a given accuracy the probe may need to make many steps through a beamline element but only the probe states at the exit of each element are needed for a particular accelerator application. Rather than saving all the intermediate states generated by the probe, which can lead to a significant waste of computer resources, we can specify what states we wish to keep after termination of the simulation. This capability is accomplished using the `update` attribute of the `<tracker>` element. Table 22 lists the supported trajectory update schemes of all the algorithm classes in XAL. The value of the `update` attribute determines the amount of trajectory information (i.e., the number of probe states) stored in the trajectory object upon completion of the simulation.

Referring to Table 22 we see that there are five possible values for the `update` attribute; all are integer indexes. A value of “`0`” indicates that the algorithm should never update the trajectory object. This value would be specified only if a custom probe algorithm was used, one that maintains its own update scheme, otherwise no simulation information would be generated. A value of “1” requests that the algorithm store every state generated in the simulation. This situation could potentially create an enormous trajectory object. However, it may be of value, for example, when using the adaptive space charge algorithm for the RMS envelope probe. There one may wish to determine where in the beamline the algorithm is spending the most computation effort and, thus, where the beam is experiencing large dynamic motion. Setting the value of `update` to “`2`” causes the algorithm to save state information only at the exit of each beamline element. This setting is probably the most common. Setting the value of `update` to “`4`” causes the algorithm to save state information only at the entrance of each beamline element. Since, in the context of most simulations, the exit of one beamline element is the entrance to the following, the only difference between value “`2`” and value “`4`” is that the trajectory generated by “`4`” will not contain the final state of the simulation (the initial states of the simulation is always saved in the trajectory). Finally a value of “6” indicates that the probe state information is to be saved at both the entrance and the exit of each beamline element.

5.3.2 The <adapt> Element

The `<adapt>` element is valid only in the context of an RMS envelope probe when using the adaptive space charge algorithm. Specifically, when the `type` attribute of the `<probe>` element is set to “`gov.sns.xal.model.probe.EnvelopeProbe`” and when the `type` attribute of the `<algorithm>` child element is set to “`EnvTrackerAdapt`”. (In the case of the RMS envelope probe there exists other available algorithms. The adaptive space charge algorithm is the most general, thus, the most practical and most widely used.) Since the RMS envelope probe is widely used for simulation, we briefly cover the `<adapt>` element. The attributes of this element are the fields of the “`adaptivetracker`” table in the model-parameter table group covered in Subsection 4.2.3, less the `name` field. Specifically then, these attributes are `errortol`, `initstep`, `maxstep`, `slack`, `maxiter`, `norm`, and `order`. The values of these fields are the same as those listed in Subsection 4.2.3, consequently, the descriptions there apply.

5.4 Specifying the Initial Probe State: The <state> Element

Beam probe objects maintain their state using a separate *probe state* objects. The `<state>` element describes a probe state at given instance, in the case of a probe file, at the initial position. Probe states maintain all the information necessary to describe a beam probe’s internal condition at any point along the beamline. Of course this information differs depending upon the type of probe being simulated; therefore, the structure of a `<state>` entry is not uniform. However, all probe state objects are derived from a common base class `gov.sns.xal.model.probe.traj.ProbeState`. Accordingly, there are common data to each probe state and, thus, common child elements of each `<state>` element. We cover these common elements and the `<envelope>` element particular to the RMS envelope probe. First we describe the attributes of the `<state>` element.

There are two attributes of the `<state>` element, `type` and `id`. The `type` attribute contains the Java class type of the probe state object. We see in the case of Excerpt 9, this class is `gov.sns.xal.model.probe.traj.EnvelopeProbeState`, indicating the probe-state Java class for the RMS envelope probe (Java type `gov.sns.xal.model.probe.EnvelopeProbe`). The `id` attribute is optional, it provides an identifier for the initial probe state which may be accessed programmatically, or during run time. In Excerpt 9 we see this value is left as `null` (i.e., `id`="").

5.4.1 Setting the Initial Energy, Time, and Position: The <location> Element

Each `<state>` element contains a `<location>` child element. The attributes of this element contain data in the base class `gov.sns.xal.model.probe.traj.ProbeState` common to all probe state objects. There are four possible attributes here, `W`, `s`, `t`, and `elem`. The `W` attribute is required; it contains the initial kinetic energy of the beam in electron-Volts. The other three attributes are all optional, having default values if not specified. The `s` attribute is the initial path length parameter. Typically this value is zero, which is its default value. There is no real significance to this value; it does not change the starting location of the simulation. It just offers the user the capability of initiating the path length parameter to a nonzero value. The `t` attribute is the initial time of the simulation. Once again, the actual value is not significant as far as the simulation is concerned; it is simply an arbitrary starting time. The default value for `t` is zero (i.e., `t="0.0"`). The last attribute, `elem`, does have an effect on the simulation. The value of this attribute is the unique `<node>` identifier where the simulation is to begin, that is., the `id` attribute of the `<node>` element. Setting the `elem` attribute to a valid node identifier causes the simulation to skip to that `<node>` within the sequence where it belongs. The default value of `elem` is `null` (`elem=""`), in which case the simulation begins at the entrance of the `<sequence>` element where it is started.

5.4.2 Specifying the Particle Charge and Mass: The <species> Element

Also part of the base `ProbeState` is the charge and rest energy of the beam particle. These values are specified using the `<species>` element. There are two attributes of this element, `q` and `Er`. The attribute `q` is the charge of the beam particle in units of the unit charge $e \approx 1.602\times10^{-19}$. The `Er` attribute is the rest energy of the beam particle in units of electron volts. Since the rest energy E_r of a particle is proportional to is mass m by a factor c^2 ($E_r = mc^2$), this is equivalent to specifying that beam particle's mass. For example, in the case of a proton, the rest energy would be specified by setting `Er="9.38e8"`.

5.4.3 Specifying the Beam Current and Bunch Charge: The <beam> Element

For simulations that involve the notion of a beam, for example an RMS envelope simulation, the corresponding probe states are derived from a child class of `ProbeState` called `gov.sns.xal.model.probe.traj.BeamState`. The later class adds parameters to the probe state which have context in these situations, namely the notions of beam current and bunch charge. (For a counter example, consider the case of single-particle simulation where there is no "beam" per se.) To represent these parameters the `<beam>` element is included under the <state> parent. This element has two attributes `I` and `Q`, representing the beam current and bunch charge, respectiviely. The value of the `I` attribute is in units of Amperes, whereas the `Q` attribute has units of Coulombs. Recall that for bunched beams in an RF system with frequency f, these parameters are related by Eq. (10).

5.4.4 Specifying the Initial RMS Beam Ellipse: The <envelope> Element

In the special case of an RMS envelope simulation, the `<envelope>` element should be present as a child of the `<state>` element. This situation is depicted in Excerpt 9. The `<envelope>` element specifies the Courant-Snyder, or Twiss, parameters of the initial RMS beam ellipse. The attributes of `<envelope>` are the α and β Courant-Snyder parameters, along with the RMS emittances for each phase plane of the beam. Thus, to specify all these parameters we need a total of six attributes, three for each phase plane. Referring to Excerpt 9, we see that there are attributes `alphaX`, `alphaY`, and `alphaZ` which are the initial α parameters for the beam ellipses in the x, y, z planes respectively. The values of these attributes are unit-less. The `betaX`, `betaY`, and `betaZ` attributes specify the initial Courant-Snyder β parameters for the x, y, z phase planes respectively. The units there are in radians/meter. Finally, the attributes `emitX`, `emitY`, and `emitZ` are the initial RMS emittances of the beam in the x, y, and z phase planes, respectively. The units for all emittance parameters, including that of the longitudinal plane z, are radian-meters.

6 Conclusion

We have covered much of what is necessary to configure an XAL installation to a particular accelerator complex. There are four primary XML files to be populated. The `main.xal` file is the first place XAL looks to configure itself. This file contains the types and locations of all the remaining configuration files. The most important, and typically the largest, is the optics file, tagged `<optics_source>` in `main.xal`. It describes all the machine hardware in the beamline relevant to XAL operation. The next file is the timing file, tagged `<timing_source>` in `main.xal`. This file is typically a small file listing the channels of general timing signals within the accelerator complex. XAL can be used without this file, although many of the existing applications that ship with XAL use these signals. The last file is the model-parameters file. This file is a special instance of the more general table-group mechanism used by XAL to import user data. All such table groups are tagged with the `<tablegroup_source>` entry within the `main.xal` file. We have discussed the general outline of a table group and how it is used to include data particular to an accelerator site, and how it may be accessed within XAL. The model-parameters table group must have the `name` attribute of its corresponding `<tablegroup_source>` entry set to "modelparams" for XAL to recognize it as such. The model-parameters table group must be included in order to use the simulation capabilities of XAL, that is, to use the XAL online model.

Since XAL is under continual development (and essentially a collaborative effort), this document is necessarily dated. As the software system is improved and updated, there are bound to be changes in the specifics described here. Fortunately, XAL is based upon a solid architectural foundation so that any fundamental changes in the XAL implementation will likely to be minimal. Thus, although some details may be altered in the future, the basic structure described here is unlikely to incur any major changes. It is also hoped that this will be a living document, one that will evolve as XAL does. By keeping this document in the public domain, developers can record their improvements and additions so that these features are accessible to all.

References

[1] C.K. Allen, C.A. McChesney, C.P. Chu, J.D. Galambos, W.-D. Klotz, T.A. Pelaia, A. Shislo, "A Novel Online Simulator for Applications Requiring a Model Reference", ICALEPCS 2003 Conference Proceedings, Kyongju, Korea, October 13-17, 2003.

[2] C.K. Allen, C.A. McChesney, N.D. Pattengale, C.P. Chu, J.D. Galambos, W.-D. Klotz, T.A. Pelaia, A. Shislo, "A Modular On-Line Simulator for Model Reference Control of Charged Particle Beams", PAC 2003 Conference Proceedings, Portland, OR, May 12-16, 2003.

[3] J. H. Billen and L. M. Young, "POISSON/SUPERFISH on PC Compatibles," Proceedings of the 1993 Particle Accelerator Conference, Vol. 2 of 5, 790-792 (1993).

[4] G. Booch, J. Rumbaugh and Ivar Jacobson, *The Unified Modeling Language User Guide* (Addison-Wesley, Reading, Massachusetts, 1999).

[5] D.C. Carey, *The Optics of Charged Particle Beams* (Harwood Academic Publishers, London, 1987).

[6] C.M. Chu, J.D. Galambos, W.-D. Klotz, T.A. Pelaia, A. Shislo, C.K. Allen, C.A. McChesney, N.D. Pattengale, "Applications Programming Structure and Physics Applications", PAC 2003 Conference Proceedings, Portland, OR, May 12-16.

[7] C.M. Chu, J. Galambos, J. Wei, C.K. Allen and P. McGehee, "SNS Applications Programming Plan", ICALEPCS 2001 Conf. Proceed., San Jose, CA, Nov 27-30, 2001.

[8] http://control.cosylab.com

[9] http://java.sun.com/j2se/1.5.0/docs/api

[10] http://www.jython.org/Project/index.html

[11] http://www.mathworks.com/

[12] http://www.w3.org/TR/1998/NOTE-datetime-19980827

[13] J. D. Galambos, C.P. Chu, S.M. Cousineau, T.A. Pelaia, A.P. Shishlo, C.K. Allen, "XAL Application Programming Structure", PAC05 Conference Proceedings, Oak Ridge, Tennessee , May 16-20, 2005, pp. 79-83.

[14] J. Galambos, C.M. Chu, T.A. Pelaia, A. Shishlo, C.K. Allen and N. Pattengale, "SNS Applications Programming Environment", EPAC 2002 Conference Proceedings, Paris, June, 2002.

[15] E.R. Harold and W.S. Means, *XML in a Nutshell* (O'Reilly, Sebastopol, CA, 2001).

[16] P. Lapostolle and M. Weiss, "Formulae and Procedures Useful for the Design of Linear Accelerators", CERN-PS-2000-001.

[17] S.Y. Lee, *Accelerator Physics* (World Scientific, Singapore, 1999).

[18] N. Malitsky and R. Talman, "The Framework of Unified Accelerator Libraries", ICAP 1998.

[19] T.L. Owens, M.B. Popovic, E.S. McCrory, C.W. Schmidt, and L.J. Allen, "Phase Scan Signature Matching for Linac Tuning", *Part. Accel.* Vol. 48, pp. 169-179 (1994).

[20] W.K.H. Panofsky and M. Phillips, *Classical Electricity and Magnetism* (Addison-Wesley, Reading, MA, 1962).

[21] M. Reiser, *Theory and Design of Charged Particle Beams* (Wiley, New York, 1994).

[22] D.H. Sattinger and O.L. Weaver, *Lie Groups and Algebras with Applications to Physics, Geometry, and Mechanics* (Springer-Verlang, New York, 1986).

[23] H. Wiedemann, *Particle Accelerator Physics I* (Springer-Verlag, Berlin, 1993).

[24] T.P. Wangler, *RF Linear Accelerators* (Wiley, New York, 1998), Chapt. 2, 6.

Appendix A: Listing of xdxf.dtd

```
<?xml version="1.0" encoding="US-ASCII" ?>


<!-- EXTENDED DESCRIPTION EXCHANGE FORMAT (XDXF)
  -       Document Type Definition
  -
  -  Version   : 1.2.0
  -  Author    : Christopher K. Allen, Paul C. Chu
  -  Modified  : June, 2002
  -
  -->


<!-- ################################################################## -->
<!-- XDXF Edit Context -->

<!ELEMENT sources ( optics_source, optics_extra*, timing_source, tablegroup_source* ) >

<!ELEMENT optics_source EMPTY >
<!ATTLIST optics_source name NMTOKEN #REQUIRED >
<!ATTLIST optics_source url NMTOKEN #REQUIRED >

<!ELEMENT timing_source EMPTY >
<!ATTLIST timing_source name NMTOKEN #REQUIRED >
<!ATTLIST timing_source url NMTOKEN #REQUIRED >

<!ELEMENT optics_extra EMPTY >
<!ATTLIST optics_extra name NMTOKEN #REQUIRED >
<!ATTLIST optics_extra url NMTOKEN #REQUIRED >

<!ELEMENT tablegroup_source EMPTY >
<!ATTLIST tablegroup_source name NMTOKEN #IMPLIED >
<!ATTLIST tablegroup_source url NMTOKEN #IMPLIED >

<!ELEMENT table ( schema, record+ ) >
<!ATTLIST table name NMTOKEN #REQUIRED >
<!ATTLIST table recordClass NMTOKEN #IMPLIED >

<!ELEMENT schema ( attribute+ ) >

<!ELEMENT tablegroup ( table+ ) >

<!ELEMENT attribute EMPTY >
<!ATTLIST attribute isPrimaryKey NMTOKEN #IMPLIED >
<!ATTLIST attribute name NMTOKEN #REQUIRED >
<!ATTLIST attribute type NMTOKEN #REQUIRED >

<!ELEMENT record EMPTY >
<!ATTLIST record alphaX NMTOKEN #IMPLIED >
<!ATTLIST record alphaY NMTOKEN #IMPLIED >
<!ATTLIST record alphaZ NMTOKEN #IMPLIED >
<!ATTLIST record beamCharge NMTOKEN #IMPLIED >
<!ATTLIST record beamCurrent NMTOKEN #IMPLIED >
<!ATTLIST record betaX NMTOKEN #IMPLIED >
<!ATTLIST record betaY NMTOKEN #IMPLIED >
<!ATTLIST record betaZ NMTOKEN #IMPLIED >
<!ATTLIST record eX NMTOKEN #IMPLIED >
<!ATTLIST record eY NMTOKEN #IMPLIED >
<!ATTLIST record eZ NMTOKEN #IMPLIED >
<!ATTLIST record envelopeID NMTOKEN #IMPLIED >
<!ATTLIST record particleCharge NMTOKEN #IMPLIED >
<!ATTLIST record particleID NMTOKEN #IMPLIED >
<!ATTLIST record particleKineticEnergy CDATA #IMPLIED >
<!ATTLIST record particleRestEnergy CDATA #IMPLIED >
<!ATTLIST record position NMTOKEN #IMPLIED >
<!ATTLIST record x NMTOKEN #IMPLIED >
<!ATTLIST record xp NMTOKEN #IMPLIED >
```

```
<!ATTLIST record y NMTOKEN #IMPLIED >
<!ATTLIST record yp NMTOKEN #IMPLIED >
<!ATTLIST record z NMTOKEN #IMPLIED >
<!ATTLIST record zp NMTOKEN #IMPLIED >


<!-- ################################################################## -->
<!-- XDXF Attribute Library -->

<!ELEMENT attributes ( align | magnet | rfgap )* >

<!ELEMENT align EMPTY >
<!ATTLIST align pitch NMTOKEN #IMPLIED >
<!ATTLIST align roll NMTOKEN #IMPLIED >
<!ATTLIST align x NMTOKEN #IMPLIED >
<!ATTLIST align y NMTOKEN #IMPLIED >
<!ATTLIST align yaw NMTOKEN #IMPLIED >
<!ATTLIST align z NMTOKEN #IMPLIED >

<!ELEMENT aperture EMPTY>
<!ATTLIST aperture shape  CDATA   #IMPLIED >
<!ATTLIST aperture x      NMTOKEN #IMPLIED >
<!ATTLIST aperture y      NMTOKEN #IMPLIED >

<!ELEMENT displacement EMPTY>
<!ATTLIST displacement x     NMTOKEN #IMPLIED >
<!ATTLIST displacement y     NMTOKEN #IMPLIED >
<!ATTLIST displacement z     NMTOKEN #IMPLIED >

<!ELEMENT magnet EMPTY >
<!ATTLIST magnet dfltMagFld NMTOKEN #IMPLIED >
<!ATTLIST magnet len NMTOKEN #REQUIRED >
<!ATTLIST magnet polarity NMTOKEN #IMPLIED >

<!ELEMENT rfgap EMPTY >
<!ATTLIST rfgap TTF NMTOKEN #REQUIRED >
<!ATTLIST rfgap ampFactor NMTOKEN #IMPLIED >
<!ATTLIST rfgap amp NMTOKEN #IMPLIED >
<!ATTLIST rfgap freq NMTOKEN #IMPLIED >
<!ATTLIST rfgap length NMTOKEN #REQUIRED >
<!ATTLIST rfgap phaseFactor NMTOKEN #IMPLIED >
<!ATTLIST rfgap phase NMTOKEN #IMPLIED >

<!ELEMENT rotation EMPTY>
<!ATTLIST rotation phi   NMTOKEN #IMPLIED >
<!ATTLIST rotation theta NMTOKEN #IMPLIED >
<!ATTLIST rotation psi   NMTOKEN #IMPLIED >

<!ELEMENT twiss EMPTY >
<!ATTLIST twiss ax CDATA #IMPLIED >
<!ATTLIST twiss ay CDATA #IMPLIED >
<!ATTLIST twiss az CDATA #IMPLIED >
<!ATTLIST twiss bx CDATA #IMPLIED >
<!ATTLIST twiss by CDATA #IMPLIED >
<!ATTLIST twiss bz CDATA #IMPLIED >
<!ATTLIST twiss etpx CDATA #IMPLIED >
<!ATTLIST twiss etpy CDATA #IMPLIED >
<!ATTLIST twiss etx CDATA #IMPLIED >
<!ATTLIST twiss ety CDATA #IMPLIED >
<!ATTLIST twiss mux CDATA #IMPLIED >
<!ATTLIST twiss muy CDATA #IMPLIED >
<!ATTLIST twiss x CDATA #IMPLIED >
<!ATTLIST twiss y CDATA #IMPLIED >


<!-- ################################################################## -->
<!-- Device Signals -->

<!ELEMENT channel EMPTY >
<!ATTLIST channel handle NMTOKEN #REQUIRED >
<!ATTLIST channel settable ( false | true ) #IMPLIED >
```

```
<!ATTLIST channel signal NMTOKEN #REQUIRED >

<!ELEMENT channelsuite ( channel* ) >
<!ATTLIST channelsuite name NMTOKEN #IMPLIED >


<!-- ################################################################# -->
<!-- Timing Signals -->

<!ELEMENT timing ( channelsuite ) >


<!-- ################################################################# -->
<!-- XDXF Data Structure -->

<!ELEMENT xdxf ( sequence+ ) >
<!ATTLIST xdxf date CDATA #REQUIRED >
<!ATTLIST xdxf system NMTOKEN #REQUIRED >
<!ATTLIST xdxf ver NMTOKEN #REQUIRED >

<!ELEMENT comment (#PCDATA)>
<!ATTLIST comment author CDATA  #IMPLIED >
<!ATTLIST comment date CDATA  #IMPLIED >

<!ELEMENT sequence ( channelsuite | node+ | sequence )* >
<!ATTLIST sequence id NMTOKEN #REQUIRED >
<!ATTLIST sequence len NMTOKEN #IMPLIED >
<!ATTLIST sequence pos NMTOKEN #IMPLIED >
<!ATTLIST sequence type NMTOKEN #IMPLIED >

<!ELEMENT node ( attributes, channelsuite? ) >
<!ATTLIST node id ID #REQUIRED >
<!ATTLIST node len NMTOKEN #IMPLIED >
<!ATTLIST node pos NMTOKEN #REQUIRED >
<!ATTLIST node type ( BCM | Bnch | BPM | DCH | DCV | QH | QV | PMQH | PMQV | RG | WS )
#REQUIRED >
```

Appendix B: XAL Channel Handles

Although the situation may be somewhat volatile, we list the channel handles for many of the hardware object in the `smf` package. Note that the safest procedure for determining these values is to consult the source code. However, it is also worthwhile to consolidate the channel handle values in order to facilitate the XAL configuration process and provide a central documentation of such.

Class	**Class Field**	**Channel Handle**	**Description**
`Electromagnet` `VDipoleCorr` `HDipoleCorr` `Bend` `Quadrupole` `TrimmedQuadrupole` `Sextupole`	FIELD_RB_HANDLE	fieldRB	Field strength readback
`RfCavity` `CCL` `DTLTank` `ReBuncher` `SCLCavity`	CAV_AMP_SET_HANDLE	cavAmpSet	Set cav. amplitude channel
	CAV_PHASE_SET_HANDLE	cavPhaseSet	Set cav. phase channel
	CAV_AMP_AVG_HANDLE	cavAmpAvg	Cav. amplitude readback
	CAV_PHASE_AVG_HANDLE	cavPhaseAvg	Cav. phase readback
	DELTA_TRF_START_HANDLE	deltaTRFStart	Delta T RF start
	DELTA_TRF_END_HANDLE	deltaTRFEnd	Delta T RF end
	T_DELAY_HANDLE	tDelay	Time delay
`Vacuum` `CvgGauge` `IonGauge`	PRESS_HANDLE	P	Pressure readback (torr)
`BLM` `NeutronDetector`	LOSS_AVG_HANDLE	lossAvg	Averaged current loss
	T_AVG_LEN_HANDLE	tAvgLen	Averaged beam length
`BPM`	X_AVG_HANDLE	xAvg	Averaged hor. position
	Y_AVG_HANDLE	yAvg	Averaged vert. position
	AMP_AVG_HANDLE	amplitudeAvg”	Ave’ed current amplitude
	PHASE_AVG_HANDLE	phaseAvg	Averaged phase position
	X_TBT_HANDLE	xTBT	Turn-by-turn hor. position
	Y_TBT_HANDLE	yTBT	Turn-by-turn vert. position
	AMP_TBT_HANDLE	ampTBT	Turn-by-turn amplitude
	PHASE_TBT_HANDLE	phaseTBT	Turn-by-turn phase pos.
	T_AVG_LEN_HANDLE	tAvgLen	Turn-by-turn ave’ed length
`RingBPM`	STAGE1_LEN_HANDLE	Stage1Len	Stage 1 length
	STAGE1_GAIN_HANDLE	Stage1Gain	Stage 1 gain (4 settings)
	STAGE1_METHOD_HANDLE	Stage1Method	Stage 1 mode (base, or 402.5)
	STAGE2_LEN_HANDLE	Stage2Len	Stage 2 length
	STAGE2_GAIN_HANDLE	Stage2Gain	Stage 2 gain (4 settings)
	STAGE2_METHOD_HANDLE	Stage2Metod	Stage 2 mode (base, or 402.5)

	STAGE3_LEN_HANDLE	Stage3Len	Stage 3 length
	STAGE3_GAIN_HANDLE	Stage3Gain	Stage 3 gain (4 settings)
	STAGE3_METHOD_HANDLE	Stage3Method	Stage 3 mode (base, or 402.5)
	STAGE4_LEN_HANDLE	Stage4Len	Stage 4 length
	STAGE4_GAIN_HANDLE	Stage4Gain	Stage 4 gain (4 settings)
	STAGE4_METHOD_HANDLE	Stage4Method	Stage 4 mode (base, or 402.5)
`CurrentMonitor`	Q_INTEGRAL_HANDLE	Particles	Macro-pulse charge
	T_AVG_LEN_HANDLE	DisplayLength	Averaged pulse length
	I_TBT_HANDLE	currentTBT	Turn-by-turn current
	T_DELAY_HANDLE	tDelay	Time delay
	I_AVG_HANDLE	currentAvg	Average beam current
	I_MAX_HANDLE	currentMax	Maximum beam current
`ProfileMonitor`	POS_HANDLE	Position	Real-time position of wire
	RT_GRAPH_HANDLE	RTGraph	???
	ABORT_SCAN_HANDLE	abortScan	Stop wire scan
	BEGIN_SCAN_HANDLE	beginScan	Begin wire scan
	CHANGE_PARAMS_HANDLE	ChangeParams	???
	ACCEPT_PARAMS_HANDLE	AcceptParams	???
	STAT_ARRAD_HANDLE	statusArray	Wire scanner status array ?
	VDATA_ARRAD_HANDLE	vDataArray	Vertical data array?
	DDATA_ARRAD_HANDLE	dDataArray	Diagonal data array?
	HDATA_ARRAD_HANDLE	hDataArray	Horizontal data array?
	POS_ARRAD_HANDLE	positionArray	Wire positions array?
	STEPS_HANDLE	nSteps	Number of scanner steps
	STEP1_POS_HANDLE	Step1Pos	Set start position of wire (mm)
	POS_SPACING_HANDLE	PosSpacing	Distance between steps?
	NO_MEAS_HANDLE	NoMeas	No. of pulses for each wire pos.
	SCAN_LEN_HANDLE	scanLength	Length of scan (mm)
	BIAS_HANDLE	Bias	Wire bias voltage (Volts)
	V_AREA_F_HANDLE	vAreaF	Vertical area fit?
	V_AMP_F_HANDLE	vAmpF	Vertical amplitude fit?
	V_MEAN_F_HANDLE	vMeanF	Vertical mean fit?
	V_SIGMA_F_HANDLE	vSigmaF	Vertical sigma fit?
	V_OFFST_F_HANDLE	vOffstF	Vertical offset fit?
	V_SLOPE_F_HANDLE	vSlopeF	Vertical slope fit?
	V_AREA_M_HANDLE	vAreaM	Vertical area RMS
	V_AMP_M_HANDLE	vAmpM	Vertical amplitude RMS
	V_MEAN_M_HANDLE	vMeanM	Vertical mean RMS
	V_SIGMA_M_HANDLE	vSigmaM	Vertical sigma RMS
	V_OFFST_M_HANDLE	vOffstM	Vertical offset RMS

	V_SLOPE_M_HANDLE	vSlopeM	Vertical slope RMS
	D_AREA_F_HANDLE	dAreaF	Diagonal area fit?
	D_AMP_F_HANDLE	dAmpF	Diagonal amplitude fit?
	D_MEAN_F_HANDLE	dMeanF	Diagonal mean fit?
	D_SIGMA_F_HANDLE	dSigmaF	Diagonal sigma fit?
	D_OFFST_F_HANDLE	dOffstF	Diagonal offset fit?
	D_SLOPE_F_HANDLE	dSlopeF	Diagonal slope fit?
	D_AREA_M_HANDLE	dAreaM	Diagonal area RMS
	D_AMP_M_HANDLE	dAmpM	Diagonal amplitude RMS
	D_MEAN_M_HANDLE	dMeanM	Diagonal mean RMS
	D_SIGMA_M_HANDLE	dSigmaM	Diagonal sigma RMS
	D_OFFST_M_HANDLE	dOffstM	Diagonal offset RMS
	D_SLOPE_M_HANDLE	dSlopeM	Diagonal slope RMS
	H_AREA_F_HANDLE	hAreaF	Horizontal area fit?
	H_AMP_F_HANDLE	hAmpF	Horizontal amplitude fit?
	H_MEAN_F_HANDLE	hMeanF	Horizontal mean fit?
	H_SIGMA_F_HANDLE	hSigmaF	Horizontal sigma fit?
	H_OFFST_F_HANDLE	hOffstF	Horizontal offset fit?
	H_SLOPE_F_HANDLE	hSlopeF	Horizontal slope fit?
	H_AREA_M_HANDLE	hAreaM	Horizontal area RMS
	H_AMP_M_HANDLE	hAmpM	Horizontal amplitude RMS
	H_MEAN_M_HANDLE	hMeanM	Horizontal mean RMS
	H_SIGMA_M_HANDLE	hSigmaM	Horizontal sigma RMS
	H_OFFST_M_HANDLE	hOffstM	Horizontal offset RMS
	H_SLOPE_M_HANDLE	hSlopeM	Horizontal slope RMS
	V_FIT_HANDLE	vFit	Vertical fit array
	D_FIT_HANDLE	dFit	Diagonal fit array
	H_FIT_HANDLE	hFit	Horizontal fit array
	V_POS_HANDLE	vPos	Vertical positions array (mm)
	D_POS_HANDLE	dPos	Diagonal positions array (mm)
	H_POS_HANDLE	hPos	Horizontal pos.'s array (mm)
	V_RAW_HANDLE	vRaw	Raw vert. intensity array (AU)
	D_RAW_HANDLE	dRaw	Raw diag. intensity array (AU)
	H_RAW_HANDLE	vRaw	Raw vert. intensity array (AU)
	V_REAL_DATA_HANDLE	vRealData	Vertical real data stream
	D_REAL_DATA_HANDLE	dRealData	Diagonal real data stream
	H_REAL_DATA_HANDLE	hRealData	Horizontal real data stream
`MagnetPowerSupply`	CYCLE_STATE_HANDLE	cycleState	Invalid 0, cycling 1, valid 2
	CURRENT_SET_HANDLE	I_Set	Set power supply current
	CURRENT_RB_HANDLE	I	PS current readback

MainMagnetSupply	CYCLE_ENABLE_HANDLE	cycleEnable	Flag: cycle mag.. when set
	FIELD_SET_HANDLE	fieldSet	Set mag. field strength
	FIELD_RB_HANDLE	psFieldRB	Field strength readback
	FIELD_BOOK_HANDLE	B_Book	MPS field setpoint
MagnetTrimSupply	FIELD_SET_HANDLE	trimSet	Set mag. field strength
	FIELD_RB_HANDLE	trimRB	Field strength readback
	TRIM_CURRENT_SET_HANDLE	trimI_Set	Set trim PS current
	TRIM_CURRENT_RB_HANDLE	trimI	Trim PS current readback

Table 23: XAL channel handle listing

Appendix C: Listing of tablegroup.dtd

```
<?xml version="1.0" encoding="US-ASCII" ?>


<!-- TABLE GROUP DOCUMENT TYPE DEFINITION
  -
  -  Version   : 1.0.0
  -  Author    : Christopher K. Allen
  -  Modified  : May, 2006
  -
  -->


<!-- ################################################################## -->
<!-- TABLEGROUP DEFINITION -->

    <!ELEMENT tablegroup (table*) >
    <!ATTLIST tablegroup name NMTOKEN #REQUIRED>

    <!ELEMENT table (schema, record*) >
       <!ATTLIST table name NMTOKEN #REQUIRED >
       <!ATTLIST table recordClass CDATA #IMPLIED >

       <!ELEMENT schema (attribute*) >

       <!ELEMENT attribute EMPTY >
       <!ATTLIST attribute isPrimaryKey (true|false) #IMPLIED>
       <!ATTLIST attribute name NMTOKEN #REQUIRED >
       <!ATTLIST attribute type CDATA #REQUIRED >
       <!ATTLIST attribute defaultValue CDATA #IMPLIED>

       <!ELEMENT record EMPTY >
       <!ATTLIST record name NMTOKEN #REQUIRED >
```

Appendix D: Example Modeling Parameters Table Group

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE modelparams SYSTEM "tablegroup.dtd">
<tablegroup>
   <table name="species">
      <schema>
         <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
         <attribute isPrimaryKey="false" name="mass" type="java.lang.Double"/>
         <attribute isPrimaryKey="false" name="charge" type="java.lang.Double"/>
      </schema>
      <record name="HMINUS" mass="9.393014E8" charge="-1"/>
      <record name="PROTON" mass="9.382720E8" charge="1"/>
   </table>

   <table name="beam">
      <schema>
         <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
         <attribute isPrimaryKey="false" name="I" type="java.lang.Double"/>
         <attribute isPrimaryKey="false" name="Q" type="java.lang.Double"/>
      </schema>
      <record name="default" I="0.020" Q="4.96894E-11"/>
   </table>

   <table name="adaptivetracker">
      <schema>
         <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
         <attribute isPrimaryKey="false" name="errortol" type="java.lang.Double" defaultValue="1.0E-3"/>
         <attribute isPrimaryKey="false" name="initstep" type="java.lang.Double" defaultValue="0.01"/>
         <attribute isPrimaryKey="false" name="maxstep" type="java.lang.Double" defaultValue="0.0"/>
         <attribute isPrimaryKey="false" name="norm" type="java.lang.Integer" defaultValue="0"/>
         <attribute isPrimaryKey="false" name="order" type="java.lang.Integer" defaultValue="2"/>
         <attribute isPrimaryKey="false" name="slack" type="java.lang.Double" defaultValue="0.05"/>
         <attribute isPrimaryKey="false" name="maxiter" type="java.lang.Integer" defaultValue="100"/>
     </schema>
     <record name="default"/>
     <record name="MEBT"/>
     <record name="DTL1"/>
     <record name="DTL2"/>
     <record name="DTL3"/>
     <record name="DTL4"/>
     <record name="DTL5"/>
     <record name="DTL6"/>
     <record name="CCL1"/>
     <record name="CCL2"/>
     <record name="CCL3"/>
     <record name="CCL4"/>
     <record name="SCLMed"/>
     <record name="SCLHigh"/>
     <record name="HEBT1" initstep="0.1" maxiter="100"/>
     <record name="IDmp-"/>
     <record name="RTBT1"/>
      <record name="Ring1"/>
   </table>

   <table name="twiss">
      <schema>
         <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
         <attribute isPrimaryKey="true" name="coordinate" type="java.lang.String"/>
         <attribute isPrimaryKey="false" name="alpha" type="java.lang.Double"/>
         <attribute isPrimaryKey="false" name="beta" type="java.lang.Double"/>
         <attribute isPrimaryKey="false" name="emittance" type="java.lang.Double"/>
      </schema>
      <record name="MEBT" coordinate="x" alpha="-1.620000" beta=" 0.155000" emittance="+3.02000e-006"/>
      <record name="MEBT" coordinate="y" alpha=" 3.230000" beta=" 0.381000" emittance="+3.46000e-006"/>
      <record name="MEBT" coordinate="z" alpha=" 0.019600" beta=" 0.584400" emittance="+3.86380e-006"/>
      <record name="DTL1" coordinate="x" alpha="-0.374691" beta=" 0.586174" emittance="+3.01915e-006"/>
      <record name="DTL1" coordinate="y" alpha="-0.231111" beta=" 0.137198" emittance="+3.45946e-006"/>
      <record name="DTL1" coordinate="z" alpha=" 0.299183" beta=" 0.471763" emittance="+3.86355e-006"/>
      <record name="DTL2" coordinate="x" alpha=" 2.701320" beta=" 0.700338" emittance="+1.73777e-006"/>
```

```
    <record name="DTL2" coordinate="y" alpha="-2.603076" beta=" 0.794770" emittance="+1.99123e-006"/>
    <record name="DTL2" coordinate="z" alpha="-0.397774" beta=" 0.403277" emittance="+2.20041e-006"/>
    <record name="DTL3" coordinate="x" alpha=" 1.297644" beta=" 0.549192" emittance="+9.92285e-007"/>
    <record name="DTL3" coordinate="y" alpha="-3.214553" beta=" 1.824498" emittance="+1.13702e-006"/>
    <record name="DTL3" coordinate="z" alpha=" 0.196423" beta=" 1.817600" emittance="+1.21681e-006"/>
    <record name="DTL4" coordinate="x" alpha=" 2.014815" beta=" 1.248958" emittance="+7.49028e-007"/>
    <record name="DTL4" coordinate="y" alpha="-2.002180" beta=" 1.295824" emittance="+8.58283e-007"/>
    <record name="DTL4" coordinate="z" alpha="-0.383242" beta=" 1.495460" emittance="+8.87229e-007"/>
    <record name="DTL5" coordinate="x" alpha="-0.080589" beta=" 2.648011" emittance="+6.25412e-007"/>
    <record name="DTL5" coordinate="y" alpha="-0.165355" beta=" 0.735660" emittance="+7.16623e-007"/>
    <record name="DTL5" coordinate="z" alpha="-0.565187" beta=" 2.410661" emittance="+7.16145e-007"/>
    <record name="DTL6" coordinate="x" alpha=" 2.126661" beta=" 1.792066" emittance="+5.49904e-007"/>
    <record name="DTL6" coordinate="y" alpha="-1.977689" beta=" 1.719692" emittance="+6.30142e-007"/>
    <record name="DTL6" coordinate="z" alpha="-0.068538" beta=" 2.218784" emittance="+6.10027e-007"/>
    <record name="CCL1" coordinate="x" alpha="-3.279096" beta=" 3.879661" emittance="+5.00893e-007"/>
    <record name="CCL1" coordinate="y" alpha=" 0.810056" beta=" 0.766892" emittance="+5.73962e-007"/>
    <record name="CCL1" coordinate="z" alpha=" 0.653812" beta=" 3.145347" emittance="+5.40359e-007"/>
    <record name="CCL2" coordinate="x" alpha=" 2.473646" beta=" 3.594577" emittance="+4.48495e-007"/>
    <record name="CCL2" coordinate="y" alpha="-0.946671" beta=" 1.394213" emittance="+5.13934e-007"/>
    <record name="CCL2" coordinate="z" alpha="-0.773927" beta=" 4.520875" emittance="+4.65244e-007"/>
    <record name="CCL3" coordinate="x" alpha=" 2.283667" beta=" 4.582043" emittance="+4.02964e-007"/>
    <record name="CCL3" coordinate="y" alpha="-1.293711" beta=" 2.236370" emittance="+4.61762e-007"/>
    <record name="CCL3" coordinate="z" alpha=" 0.538514" beta=" 3.153244" emittance="+3.99505e-007"/>
    <record name="CCL4" coordinate="x" alpha=" 2.787507" beta=" 7.072388" emittance="+3.65633e-007"/>
    <record name="CCL4" coordinate="y" alpha="-1.313166" beta=" 3.617282" emittance="+4.18994e-007"/>
    <record name="CCL4" coordinate="z" alpha="-0.634955" beta=" 4.202553" emittance="+3.45471e-007"/>
    <record name="SCLMed" coordinate="x" alpha="-1.573126" beta=" 8.131378" emittance="+3.34159e-007"/>
    <record name="SCLMed" coordinate="y" alpha=" 0.684698" beta=" 8.634511" emittance="+3.82927e-007"/>
    <record name="SCLMed" coordinate="z" alpha=" 0.213766" beta=" 6.071008" emittance="+2.99985e-007"/>
    <record name="SCLHigh" coordinate="x" alpha=" 0.379621" beta=" 4.666692" emittance="+2.19297e-007"/>
    <record name="SCLHigh" coordinate="y" alpha=" 2.364523" beta="13.876136" emittance="+2.51303e-007"/>
    <record name="SCLHigh" coordinate="z" alpha="-1.622597" beta=" 9.553335" emittance="+1.40606e-007"/>
    <record name="HEBT1" coordinate="x" alpha=" 1.014436" beta=" 3.569722" emittance="+1.21644e-007"/>
    <record name="HEBT1" coordinate="y" alpha="-1.748688" beta=" 9.444302" emittance="+1.39475e-007"/>
    <record name="HEBT1" coordinate="z" alpha="-103.218446" beta="7096.290320" emittance="+3.67111e-008"/>
    <record name="HEBT2" coordinate="x" alpha="-0.641247" beta=" 4.717389" emittance="+1.21644e-007"/>
    <record name="HEBT2" coordinate="y" alpha=" 1.777757" beta="19.469339" emittance="+1.39475e-007"/>
    <record name="HEBT2" coordinate="z" alpha="-170.8246" beta="18586.384" emittance="+3.67111e-008"/>
    <record name="IDmp-" coordinate="x" alpha=" 0.073049" beta="12.568964" emittance="+1.21644e-007"/>
    <record name="IDmp-" coordinate="y" alpha="-0.271860" beta="11.454688" emittance="+1.39475e-007"/>
    <record name="IDmp-" coordinate="z" alpha="-314.2278" beta="58746.924" emittance="+3.67111e-008"/>
    <record name="IDmp+" coordinate="x" alpha="-0.057543" beta="12.588942" emittance="+1.21644e-007"/>
    <record name="IDmp+" coordinate="y" alpha="-0.560435" beta="13.467029" emittance="+1.39475e-007"/>
    <record name="IDmp+" coordinate="z" alpha="-316.8352" beta="59691.567" emittance="+3.67111e-008"/>
    <record name="RTBT1" coordinate="x" alpha="-1.3168" beta="5.8471" emittance="1.60e-4"/>
    <record name="RTBT1" coordinate="y" alpha=" 0.6831" beta="9.2607" emittance="1.60e-4"/>
    <record name="RTBT1" coordinate="z" alpha="-0.0036" beta="9589.2334" emittance="11.4e-3"/>
    <record name="Ring1" coordinate="x" alpha="0.05" beta="11.8" emittance="1.50e-7"/>
    <record name="Ring1" coordinate="y" alpha=" 0.05" beta="13.4" emittance="1.50e-7"/>
    <record name="Ring1" coordinate="z" alpha="1." beta="18586.384" emittance="+5.e-8"/>
  </table>

  <table name="location">
    <schema>
      <attribute isPrimaryKey="true" name="name" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="species" type="java.lang.String"/>
      <attribute isPrimaryKey="false" name="W" type="java.lang.Double"/>
      <attribute isPrimaryKey="false" name="elem" type="java.lang.String" defaultValue=""/>
      <attribute isPrimaryKey="false" name="s" type="java.lang.Double" defaultValue="0"/>
      <attribute isPrimaryKey="false" name="t" type="java.lang.Double" defaultValue="0"/>
    </schema>
    <record name="MEBT" species="HMINUS" W="2.5E6"/>
    <record name="DTL1" species="HMINUS" W="2.5E6"/>
    <record name="DTL2" species="HMINUS" W="7.525E6"/>
    <record name="DTL3" species="HMINUS" W="2.2885E7"/>
    <record name="DTL4" species="HMINUS" W="3.9765E7"/>
    <record name="DTL5" species="HMINUS" W="5.6536E7"/>
    <record name="DTL6" species="HMINUS" W="7.2521E7"/>
    <record name="CCL1" species="HMINUS" W="86.828E6"/>
    <record name="CCL2" species="HMINUS" W="107.161E6"/>
```

```
        <record name="CCL3" species="HMINUS" W="131.142E6"/>
        <record name="CCL4" species="HMINUS" W="157.214E6"/>
        <record name="SCLMed" species="HMINUS" W="185.6266E6"/>
        <record name="SCLHigh" species="HMINUS" W="391.4331E6"/>
        <record name="HEBT1" species="HMINUS" W="1001.122E6"/>
        <record name="HEBT2" species="HMINUS" W="1001.122E6"/>
        <record name="IDmp-" species="HMINUS" W="1001.122E6"/>
        <record name="Ring1" species="PROTON" W="1000.025E6"/>
        <record name="Ring2" species="PROTON" W="1000.025E6"/>
        <record name="Ring3" species="PROTON" W="1000.025E6"/>
        <record name="Ring4" species="PROTON" W="1000.025E6"/>
        <record name="Ring5" species="PROTON" W="1000.025E6"/>
        <record name="IDmp+" species="PROTON" W="1000.025E6"/>
        <record name="RTBT1" species="PROTON" W="1000.025E6"/>
    </table>
</tablegroup>
```