An XML-Based File Format for Archival Storage of Analytical Instrument Data

Table of Contents

Introduction **1**
Existing Standardized Data Formats for Analytical Instrumentation **3**
JCAMP 3
ANDI (AIA) 4
Galactic SPC 5
Review 6
XML: A New Data Interchange Model **7**
Representing Analytical Instrument Data Using XML **9**
Dictionary XML of Terms 10
Document 10
Element 10
Attribute 10
Generalized Analytical Markup Language Hierarchy (GAML) 11
The <GAML> Element 11
The <experiment> Element 11
The <trace> Element 12
The <Xdata> Element 12
The <altXdata> Element 13
The <Ydata> Element 14
The <coordinates> Element 14
The <values> Element 15
The <parameter> Element and Meta-Data 15
The <integrity> Element 17
The <link> Element and "linkid" and "linkref" Attributes 17
Generalized Peak Table Hierarchy 19
The <peaktable> Element 19
The <peak> Element 20
The <peakXvalue> and <peakYvalue> Elements 20
The <baseline> Element 21
XML Examples Of Different Types of Analyses 22
Chromatogram Example 22
LC-PDA Spectra Example 22
GC/LC-MS Spectra Example (Centroided Scans) 22
MS^n Spectra Example (Centroided Scans) 23
TGA Analysis Example 24
Simple Optical Spectroscopy Example (FTIR, UV-Vis, NIR, Raman, etc.) 24
FTIR Spectroscopy Example (FTIR, UV-Vis, NIR, Raman, etc.) 24
1D NMR Spectroscopy Example 24
nD NMR Spectroscopy Example 25
Imaging Spectroscopy Example 26
Encapsulation of Data in Other XML Formats 27
Appendices **28**
Appendix A – Enumerated Attribute List of Trace Techniques 28
Appendix B – Enumerated Attribute List of Axis Units 29
Appendix C – Enumerated Attribute List of Array Data Formats 31
Appendix D – Enumerated Attribute List of Data Value Ordering 32
Appendix E – GAML Element and Attribute Definitions 33
Appendix F – GAML Schema Definition (XSD) 47

Introduction

There is a growing demand for digital storage and archiving systems for analytical instrument data. Although there are data archiving products currently on the market, they are incomplete solutions for long term archiving of data from analytical instruments. In general, these systems offer a centralized method of indexing, storing and retrieving the original binary data files generated by the many proprietary instrument control software packages used throughout the laboratory. When a particular piece of data is retrieved from such a system, it is viewed using that same proprietary software that generated it.

There are two principal problems with this approach:

1. Laboratories typically have many more people that need to access the data than they have computers running the proprietary software required to view it. In some cases, those people may be in a different lab, building or country than the proprietary data station software. It is impractical to deliver copies of proprietary software applications to every person who might need to access the data.
2. Instruments and data systems often have shorter lifetimes than the required retention periods for the actual data they generate. It is very likely that when a critical piece of data is needed some time in the distant future (to demonstrate compliance to a regulator or for a legal defense of a company's intellectual property), the data station software, hardware or even operating system is obsolete or cannot be obtained anymore.

Users must be able to access, view and potentially even reprocess the data in the archive throughout the entire record retention lifetime and beyond. Thus, in order to make data accessible for an undetermined length of time, it must be "normalized" in such a way that it can be easily understood outside the realm of any individual software system.

In the process of developing a new software product for enterprise-wide management of instrument data, Thermo LabSystems has proposed a new XML-based data model. This paper will focus on how XML solves a number of issues related to access and storage of analytical instrument data and will describe the features of the proposed data model in detail.

The users with the greatest need for data archiving systems are those working in companies that operate in markets that are regulated by government institutions. Currently, one of the most visible regulations is USFDA 21 CFR Part 11 and how it relates to electronic record keeping. What is interesting to note is that this regulation specifically acknowledges the difficulties in maintaining accessibility to electronically archived data (see "Comments on the Proposed Rule", Section III, Part 69) by stating:

> "The agency agrees that providing exact copies of electronic records in the strictest meaning of the word ‘‘true’’ may not always be feasible. The agency nonetheless believes it is vital that copies of electronic records provided to FDA be accurate and complete. Accordingly, in §11.10(b), ‘‘true’’ has been replaced with ‘‘accurate and complete.’’ The agency expects that this revision should obviate the potential problems noted in the comments. The revision should also reduce the costs of providing copies by making clear that firms need not maintain obsolete equipment in order to make copies that are ‘‘true’’ with respect to format and computer system."

Using 21 CFR Part 11 as a guide, the goal of this document is to define a common data format that meets the needs for long term data archiving which is an "accurate and complete" representation of the data collected by analytical instruments. Given the outline provided by this regulation, there are a number of criteria that must be met to make a data format viable for long term electronic archiving of analytical instrument data:

1. It must be "readable" for an undetermined length of time into the future. The lifetime of proprietary instrument software and related systems (computer hardware and operating systems) is usually much shorter than that of the data related to a project or imposed by regulatory requirements. It is very likely that the only usable version of the data when it is needed in the future will be that which is stored in the common format.

2. It must be able to outlast the computer hardware and operating system(s) used to store it. It must be something that can be easily migrated as electronic storage systems, operating systems and software are upgraded.
3. It should be (or be based on) an existing data format standard that is in the public domain. Even data formats that are "open" can be considered proprietary data formats if software and documentation cannot be obtained for a particular computer platform. The data format should be cross-platform compatible and no single source or group should control access to the documentation. Anyone should be able to easily obtain or develop tools for making use of the data stored in the common format.

Existing Standardized Data Formats for Analytical Instrumentation

Ever since the beginning of the era of computers in spectroscopy and chromatography, instrumentation users have looked for a way to easily exchange data between the different systems. The root cause of these efforts has always been the proprietary design of software provided with the instruments, which prevents the data files from being viewed, processed or printed outside the system. As such, there are already a number of data format standards in use within the analytical laboratory instrumentation marketplace.

Before expending the effort to design a completely new data format for analytical data, Thermo Galactic and Thermo LabSystems conducted a review of existing data format standards to determine if any would meet the requirements for long term archiving. However, it became apparent that none of these formats would meet all three of the requirements. In retrospect, the driving force behind the previous efforts to create these formats was not to create a 'future-proof' way of storing data (as required for archival storage), but instead to exchange data with users of other software systems. On the other hand, there are some useful ideas and a wealth of experience that can be gleaned from examining the precedent efforts in the context of designing a new archival storage format. The following sections discuss the formats reviewed in more detail.

JCAMP

One of the first attempts at developing a common data interchange format was the JCAMP effort. This was first proposed by a group of spectroscopists and led by Bob McDonald[1] back in the mid-1980's as a way to seamlessly transfer FTIR spectra among the various spectrometer manufacturers' computer data stations of that era. In subsequent years, it was adopted and moved forward by a working group within the IUPAC[2,3], and standards for other types of data have been proposed (NMR, Mass Spectrometry). In addition, many universities, research organizations and instrument vendors have produced data files in their own "flavors" of the JCAMP format.

The idea behind this format was to make it ASCII such that it was easily readable/writable by anybody with a text editor and explicitly described all the information in the data. Important data is identified by a system of standard "tags" which are described in a dictionary within the standards documentation. This makes the files fundamentally useful in nearly all software systems since ASCII is common to almost all computer platforms and operating systems and is fundamentally "human-readable."

The main problem with JCAMP is the fact that the specifications are both complicated and incomplete. As the documentation grew to allow for other types of instrument data, peak table data, and chemical structures, among other things, programmers had trouble interpreting how the tags should be used to write out their type of data. This led to the situation where JCAMP files created by a given software system could not be read by any other system because of inconsistent software implementations. This was exacerbated by the fact that there was (and still is) no way of validating the adherence to the JCAMP standards for files created by different software packages. There is no testing software available, and there is no overseeing body organizing round-robin testing among the vendors and policing their efforts.

This type of problem is fixable, while the issue of data completeness is not. One of the best ideas the developers of the JCAMP format put forth was to purposely keep the dictionary of tags as small as possible. Programmers were free to store as much other information in the file as they needed in private "comment" tags. Since the data in the file is stored as ASCII, any reasonable human reading these tags (or computer programmed to look for them) would be able to decipher the information.

However, in choosing to also use ASCII to store the numerical values that make up the "curves" in the data, they created a format which is unsuitable for archival storage. Unless the values are written to the JCAMP file using the same precision as the binary data in the original proprietary file, they will not be exactly the same values when the JCAMP file is read by another piece of software. For example, it is common for instrument vendors to use 32-bit floating point numbers to store data values in their proprietary data file formats. In order to accurately translate a number stored in this representation into ASCII, the software must preserve all 8 significant figures and the scientific exponent (i.e.,

123.45678e+09). Any other ASCII representation (e.g., 123.45e+9) loses data precision and will result in a *different* 32-bit floating point number when read by another piece of software. This does not meet the archiving requirement for an "accurate and complete" representation of the data.

References

1. R. S. McDonald and P. A. Wilks Jr., "JCAMP-DX: A Standard Form for Exchange of Infrared Spectra in Computer Readable Form", *Appl. Spec.*, Vol. 42, 1988, p. 151.
2. A. N. Davies and P. Lampen, "JCAMP-DX for NMR", *Appl. Spec.*, Vol. 47, 1993, p. 1093.
3. P. Lampen, H. Hillig, A. N. Davies and M Linscheid, "JCAMP-DX for Mass Spectrometry", *Appl. Spec.*, Vol 48. 1994, p.1545.

ANDI (AIA)

Another effort aimed at creating a standardized data interchange file format is called ANDI (**AN**alytical **D**ata **I**nterchange) originally developed, supported and promoted by the Analytical Instrumentation Association (AIA), a marketing organization that tracks trends and researches growth and new technologies in the industry. AIA organized a series of separate committees comprised of instrument vendors and users to design formats for specific analytical disciplines. Steered by their members, they decided to initially work on the techniques that were most relevant: Chromatography and Mass Spectrometry (although efforts were also started for FTIR/Raman, NMR and PDA data types). The feeling was that these groups could better design a file format for the data if they did not have to worry about making it general enough to cope with all instrumental data, but only had to make one sufficient for a single technique.

Unlike the JCAMP effort, they had specific goals of not only allowing data interchange, but also of providing a mechanism that could maintain GLP & GMP integrity of the data. Any file format that was completely stored in ASCII (like JCAMP) was deemed not secure enough to meet this goal; some type of binary format that could not be easily manipulated was required. This was also a requirement to maintain the precision of the data values in the original measurement, another failing of ASCII-based formats like JCAMP.

The AIA committees also had a mandate to make sure the data files would be portable among many different operating systems and software platforms, a very difficult task with binary data. In the end, they ended up choosing Unidata's netCDF format. This is supported by free, public domain software and source code that can be compiled for cross-platform (operating system) compatibility.

The committees focused on defining the dictionary of items that would be stored in the file and defining as many tags as possible to give a completely GLP-compliant way of storing the data. Because of this, the dictionaries for these formats are extraordinarily large. To simplify implementation, the ANDI formats define a series of five levels of compliance and the dictionary tags are divided into groups. In order for a file to be compliant with a certain level of completeness, it must include all the tags for that level and prior levels as well.

This effort was backed up with a program of round-robin testing among the instrument vendors, all coordinated by the AIA. Vendors traded files and published their results in reading each other's data in order to guarantee the best level of interoperability for their customers. Initial efforts looked promising as the instrument vendors cooperated on the Chromatography, MS and FT-IR specifications. The first two were actually publicly released and AIA/ANDI export utilities can be found in many vendors' software packages for these types of instrumentation. More recently, AIA turned over the management of the ANDI efforts to the E01.25 committee of the American Society of Testing and Materials (ASTM) and both the Chromatography and MS data formats have been approved as formal ASTM standards[1-4].

The concept of using the netCDF binary format to preserve GLP and numerical accuracy was a key development. Since netCDF was already in the public domain, and supported by software and source code available from an independent agency (Unidata), there were presumably no issues that the data would be lost in future revisions of operating systems and software. In addition, having panels of experts

in each instrumental discipline to define the dictionaries insured that the formats would represent the information in the data files in the most complete manner possible.

But despite these apparent benefits, ANDI has significant drawbacks as a format for archival storage of analytical data. While it addresses the “accurate” requirement fairly well, it falls short in the area of being an open standard and future-proof. The main reason is due to the choice of netCDF as a data storage format. Although the netCDF software is public domain technology, it is not widely used. It is likely that as computer hardware and operating systems evolve, the netCDF code base will not be upgraded for compatibility in a timely manner. Ultimately, data stored using the netCDF software may be just as inaccessible to future computer systems as the proprietary binary instrument files themselves.

Another weakness lies in the complex dictionaries of required tags for data completeness. While the fixed sets of tags defined in the ANDI formats are rather extensive, they do not allow the format to be extended easily to new instrumental analysis techniques as they develop. The breadth of the dictionaries also creates a situation where full compliance for certain vendors may be impossible. If a certain vendor's instrument or file format does not provide a parameter that is considered “required” in the ANDI standards, then it cannot be stored when the file is translated. This could prevent other software packages from being able to read the file, as it is not considered "complete" according to the standard.

A final problem is that the published standards only cover Chromatography and Mass Spectrometry data sets at this point. There are a vast number of other types of instrumentation in use in today's laboratories (i.e., NMR, UV-Vis, PDA, etc.) and none of the data generated by these pieces of equipment can be represented in the ANDI formats.

References

1. ASTM E1947-98, "Standard Specification for Analytical Data Interchange Protocol for Chromatographic Data".
2. ASTM E1948-98, "Standard Guide for Analytical Data Interchange Protocol for Chromatographic Data"
3. ASTM E2077-00, "Standard Specification for Analytical Data Interchange Protocol for Mass Spectrometric Data"
4. ASTM E2078-00, "Standard Guide for Analytical Data Interchange Protocol for Mass Spectrometric Data"

Galactic SPC

As an independent, third party software company for over 15 years, Thermo Galactic (formerly Galactic Industries Corp.) has produced products that allow analytical chemists to manipulate their instrument data on an PC-compatible computer. This was accomplished by working with customers and instrument vendors to develop a large number of translation modules that allow the software to directly read the proprietary instrument file formats. As part of this effort, Galactic developed a file format of its own known as “Galactic SPC” that is used in all of its products. Since its inception and invention, this format has been published in Galactic’s documentation and via other public domain sources (i.e., the Internet).

Initially focused on storing only FTIR spectroscopy data, Galactic’s customer base (and thus its library of file translators) quickly grew to include users with other types of instruments including UV-VIS, Fluorescence, Raman, NIR, Chromatography, NMR, MS and many hyphenated techniques such as GC-IR, GC-MS, and LC-DAD. To this day, Galactic continues to develop conversion routines for as many different file formats as possible, including the previously discussed “standard” formats. Since users can bring raw binary data files from their instruments directly over to Galactic software, translate and then process them, there has been little need for the instrument vendors to supply the user with an export routine to another format. Thus the need for standard data exchange formats has been minimized for a large part of the analytical instrument community (at least among those who are already using Galactic software)

Another part of this is Galactic's success as an OEM supplier of instrument software. Many instrument vendors now use Galactic software products as their direct data acquisition interface to their instrumentation. These packages naturally store the data in Galactic's native SPC file format. Thus there is an ever growing collection of data in the world stored in Galactic SPC files.

Due to this, a large number of other software packages support export to SPC files; in some cases in lieu of supporting export to one of the "standard" file formats (i.e., J-CAMP, ANDI or even simple 2 column ASCII). In fact, the SPC file format is becoming a de facto standard for exchanging spectroscopic data. This is almost certainly true in the FT-IR and Raman spectroscopy marketplaces, and becoming so for UV-Vis and NIR spectroscopy.

The Galactic SPC format was designed to meet the needs of a user who wants to view, process and print the data outside the instrument vendor's software. It is flexible enough to store many different kinds of data, while at the same time uses a binary representation for the data values to maintain the all-important numerical accuracy of the curves. In addition, it has a flexible mechanism for unformatted ASCII parameter information (called "Log Text") to allow for carrying along important information from the source instrument file format. In addition, Galactic even generated internal dictionaries of Log Text items for storing commonly used parameters from many of the different instrumental analysis techniques. One of the nicer things about the format is that there are a large number of software products and tools available from both Galactic and other vendors that can be used to validate SPC files without requiring extensive round robin testing.

It was not, however, designed as an archival storage format and because of this, suffers from some of the same deficiencies as the JCAMP and ANDI formats. For one, while it may be accurate in terms of storing the numerical representation of the curves, the current dictionary of Log Text items is not "complete" in terms of the total volume of information available in most proprietary instrument data files. In addition, although it is supported by many software packages on the market, it is not a universal standard. It is still a binary file format that is specific to certain types of software. Finally, it was designed using a "flat file" format, making it very difficult, if not impossible to expand it to address new instrumental analysis techniques as they are developed (i.e., it cannot encapsulate all the data from today's multi-hyphenated techniques such as LC-MS-PDA).

Review

It is interesting to note that all of these formats still survive today in various software products. They serve specific needs, but do not fully address the requirements of archival storage. However, as mentioned earlier, there are good ideas in all these attempts. As an example, it is encouraging to note that while the Galactic SPC format is not the ultimate archival storage format, for over 10 years it has remained unchanged while instruments and software technology have changed quite drastically. Rather than try to design the perfect format, Galactic set out to design something that met the specific requirements of its customer base; to use their data on any PC and share it with their colleagues. In this respect, the Galactic SPC format has been very successful.

Given today's climate of rapid advances in software technology and the push towards standards in networking and data interchange brought on by the growth of the Internet, it seems that this is the arena where we will find a public domain technology that meets the needs for archiving analytical instrument data. By taking the best ideas from the various precedent formats, melding them with a new data storage mechanism and making the resulting format completely open, it should be possible to create something that will actually be a universal and future-proof format. It does not have to have vast dictionaries to address every corner case and deal with every possible parameter that can ever be generated by an instrument. It merely has to be flexible and extensible enough to allow the information to be packaged along with the numerical curves, in such a way that it can be used long into the future as an "accurate and complete" record of the original data.

XML: A New Data Interchange Model

All of the precedent standardized instrument data formats have distinct advantages in the context of the problem they were intended to address. For the most part, that was to allow interchange of data between various software packages. However, none of the groups (either public or private) that put forward these formats had the weight of large groups of users, instrument developers, or government regulations pushing them to do this. At the time when most of these standards were developed (between 8 and 15 years ago), the need to store and share data was not that great, mainly because the volume of available data was just not that big and the world of laboratory instrumentation was not that connected. Most data tended to stay where it came from – on the instrument workstations that collected it.

However, today the term "instrumental analysis" has quite a different meaning. Instead of collecting samples and analyzing them in serial fashion, the push is to move towards higher throughput technologies. On top of that, there has been incredible growth in new developments in the areas of genomics, proteomics and pharmaceuticals. Recently, new government regulations regarding authentication, controlled access, and long term storage of digital information have caused an incredible burst in demand for software systems capable of archiving and sharing all this data. However, the current state-of-the-art in standardized data representations is comprised of the data interchange formats developed long ago.

Meanwhile, both the capabilities and connectivity of the Internet have grown at such a remarkable pace that mainstream business applications quickly migrated from mainframe systems to use web browser front-ends. However, as these applications have become more web-centric, they have also needed to become more "connected." Vast quantities of data are being generated in real-time and stored in databases in various incompatible formats. To make use of these stores of data, web-enabled applications must have a way to seamlessly exchange information.

To address this need, the World Wide Web Consortium (W3C, http://www.w3.org), the independent standards body that governs the definitions of the formats used on the Internet, put together a working group and developed the Extensible Markup Language (XML); a universal format for exchanging structured documents and data on the Web. It is the existence of XML that allows a company with manufacturing facilities all over the world to harness the power of the Internet to keep track of their entire corporate inventory even though the data is stored on many different computers. XML is used by companies to exchange "live" data among computers on the various world stock exchanges and execute purchase and sell orders between brokers and buyers.

The key to XML is that it is not actually a file format, but instead is a standardized and application-independent way of representing data. In XML, all data and information is stored in plain ASCII text (because that's the only way it can be transmitted across the Internet). However, XML is very different from the free-form ASCII files that can be "dumped" from instrument software. XML has an inherent system of defining hierarchical tags and attributes that describe and represent the relationships between pieces of data. Unlike non-structured ASCII files, the data structures within an XML file are not left entirely to the programmer's whim. The XML document model gives the ability to link to other documents (many times on the publicly-accessible web sites) that describe exactly what each piece of data means and how it is related. These documents, called Data Type Definitions or DTDs (now being replaced with a newer technology called XML Schema), exactly describe how the XML language is used to store a particular type of information. With the appropriate DTD or Schema, a software application can parse data from any XML data structure and determine that it is well formed (a major problem with unformatted ASCII files) *even though it has no idea how the data is relevant*.

What makes XML useful for exchanging a certain type of data is when a group of software developers agrees to use a single DTD or Schema. Taking it one step further, if there are many other developers and companies that can benefit from a standardized representation of that type of data, it can be put into the public domain. Today, there are a growing number of public domain XML DTDs and Schemas for a wide range of data types including Mathematics, Banking, Inventory Control , Accounting, Airline Reservations and many others (visit http://www.xml.org for an up-to-date list).

One very relevant example comes from a group at Imperial College in the United Kingdom. They have developed what they call "CML," an XML-based format for chemical information. They have defined a Schema to be used for storage and exchange of chemical structures and crystal lattices (for more information, visit http://www.xml-cml.org). The documentation and examples they have posted on their web site give excellent insight into the power and portability of data when it is represented in XML.

As a technology, XML is tailor made as basis of a file format for long term archiving of analytical instrument data for a number of reasons:

1. It is based on a public domain standard controlled by a completely independent body, the W3C.
2. It is currently used as a data interchange mechanism by many mainstream business applications and has been proposed to be a universal data interchange standard for all types of data via networked applications (i.e., Intranets and the Internet). This means it is highly likely that it will be supported on future computer platforms.
3. The Schema or DTD that defines the structures for a particular type of data can be widely distributed (i.e., a web site, database or file server) and software applications can use it to automatically validate the "correctness" of the formatted data whenever an XML file is opened.
4. It uses ASCII as the storage mechanism (like JCAMP) which is fundamentally "human readable" and should be for a long time into the future. Presumably the standard ASCII character set will continue to be supported on all future computer platforms.
5. It uses a system of hierarchical data structures and attached attributes, which allows representing implied and complex relationships between information stored in the file (like ANDI/netCDF).
6. It uses a system based on a dictionary of well-defined tags to ascribe both relevance and identity to the data to which they are attached (like JCAMP & ANDI/netCDF).
7. It can encapsulate data as binary information to maintain accuracy and precision (like ANDI and Galactic SPC). However, it accomplishes this using standardized encoding mechanisms to allow storing the information into a simple ASCII string.

For these reasons, among others, Thermo Galactic and Thermo LabSystems feel that XML is the best currently available technology to use as the basis for a new standardized file format for archival storage of analytical instrument data.

Representing Analytical Instrument Data Using XML

The following is an overview of a proposed XML-based data format designed specifically for archiving data from analytical instrumentation. The concept is to use the hierarchical nature of XML to imply the relationships between the data stored in the file, while at the same time attempting to be efficient in terms of storage space. In other words, there is no sense in storing large blocks of data that will be repeated throughout the file (i.e., the wavelength values for all the spectra from an LC-PDA run) if it can be implied by a hierarchical structure.

The overall design of the format is based on the fact that nearly any data collected from an instrument can be represented as arrays of numbers. This is fundamentally the "atomic unit" of information in a data file from an analytical instrument. One could argue that a single X,Y data point pair is the atomic unit; however, knowing the values of one pair of coordinates in a chromatogram or spectrum without the context of all the other data points that were collected is not useful. In addition, this information can easily be generated from the arrays of stored data values.

How the various arrays are “tied together” can be expressed by the nested hierarchy offered by the XML data model. For data from high order detectors (i.e., multidimensional data sets like GC/LC-MS, MS^n, imaging spectroscopy, multi dimensional NMR), each array can be tagged with any number of coordinates indicating their position or mapping in other dimensions. This gives the most flexibility in representing data at the cost of some efficiency in accessing “columns” of data where only a few values or a single value in every array is needed to generate an alternate slice of the data (i.e., a chromatogram at a specific wavelength from a set of PDA spectra).

The design that follows specifically avoids trying to define hierarchies of data that can be derived from the actual raw data collected from the detector. For example, there is no explicit way of storing a Total Ion Chromatogram (TIC) within the hierarchy of a set of MS spectra collected from an LC-MS run. This information can be derived from the coordinates of each MS scan (time values in this case) and the individual sums of the values in the data array for each scan. However, the software developer implementing an "export to XML" function could choose to add an additional data structure to the file containing the TIC chromatogram. In cases where additional information was derived from the TIC that must be stored (such as a peak table with quantitation results), this might be an absolute requirement.

Dictionary of XML Terms

In order to fully understand the organization of any XML-based data format, it is necessary to understand the following basic terms and how they apply to the data structure. Note that this dictionary is in no way complete; it is purely intended to provide a frame of reference for the sections that follow. For those interested in learning the complete language and terminology of XML, please refer to the XML resources section of the W3C web site (http://www.w3.org/XML/).

Document

The standard term for a body of data that is formatted and described using XML is generally referred to as a "document." Although the ASCII characters that make up an XML data stream can be stored to non-volatile storage in many hardware and operating systems, the concept of a "file" does not really apply to XML data. XML representations of data can also be generated from database queries, software applications, or even hardware and streamed between computers without ever existing as a "file" in any one particular operating system. However, for practical purposes in the case of storing analytical instrument data to a long term archive (files in non-volatile storage), the term "file" and "document" are basically equivalent and are used as interchangeable terms in the following descriptions.

Element

An "element" is a unique tag used to represent a specific piece of information in XML. An element can consist of data or be a collection of other elements. Elements are the fundamental building blocks used to identify data and define relational hierarchies of information in an XML document.

Attribute

An "attribute" is an item of information attached to an element that is relevant to the data contained within the element itself. In some cases, information stored in an attribute could also be represented as an additional element within the parent element's hierarchy. However, attributes of a given type can be applied to one or more elements within an XML hierarchy, while elements generally follow a specific hierarchy (although this is not a requirement).

Generalized Analytical Markup Language Hierarchy (GAML)

The following is a generalized overview of the proposed XML data structure for archiving analytical instrument data. This is not an exact definition of how it will appear in an XML file, but rather a representation to familiarize the reader with the hierarchical nature of XML and how it can be used advantageously for this type of data.

```
<GAML>                                  required root identifier
   <integrity>                          document integrity check
   <parameter>                          meta-data; can appear multiple times
   <experiment>                         data from single instrument “run”
      <parameter>                       meta-data; can appear multiple times
      <collectdate>                     measurement date/time
      <trace>                           data from a single detector
         <parameter>                    meta-data; can appear multiple times
         <coordinates>                  coordinates for multidimensional data types
            <parameter>                 meta-data; can appear multiple times
            <values>                    array of base64 encoded values
         <Xdata>                        X axis (abscissa) data
            <link>                      link reference to other elements
            <parameter>                 meta-data; can appear multiple times
            <values>                    array of base64 encoded values
            <altXdata>                  alternate X axis (abscissa) data
               <parameter>              meta-data; can appear multiple times
               <values>                 array of base64 encoded values
            <Ydata>                     Y axis (ordinate) data
               <parameter>              meta-data; can appear multiple times
               <values>                 array of base64 encoded values
               <peaktable>              peak table data
```

The <GAML> Element

All XML-based data formats require a "root element.” This provides parsing applications with a definitive place to start reading the encapsulated data. In the case of data formatted according to this proposal, the root element is <GAML>, which as an acronym for "Generalized Analytical Markup Language.”

The root elements of XML documents in a specific format have a number of standard attributes. In many cases, there will be an "xmlns" attribute indicating the XML namespace which tells parsers what Schema was used to write the document. This indicates where the Schema can be found and can be a local file or an Internet URL among other things. The <GAML> root element also has a required "version" attribute which is used to compare the version of a referenced Schema to the version used to write the document. This assures applications developers who use validating XML parsers that the document they are reading is properly formatted against the Schema for that version.

The <GAML> element can have an optional "name" attribute which describes the document for informational purposes. The only elements that can appear within the <GAML> root element are <parameter> and <experiment>.

The <experiment> Element

The concept of what makes up an "experiment" varies among instrument types, vendors and software packages. However, for the purpose of this data format, it represents the parent item that allows storing all the data sets and parameters needed to keep a complete record of a given analysis that must be kept together for reasons of data completeness.

For example, an experiment may consist of an injection on liquid chromatograph with multiple detectors connected (i.e., UV, RI, EC, etc.). In this case, all of the chromatograms, the associated instrument settings and peak quantitation reports must be kept together in order to insure that the data set is "complete.” However, in the case of a simple UV-Vis scan, it might be just a single spectrum.

Note that any single XML document in this format can contain multiple <experiment> elements. This allows encapsulation of entire "runs" of data sets that represent a complete analysis set. For example,

this could be used for a complete sequence of chromatograms that includes data measured from calibration standards, samples, check samples and blanks. Another example is a set of chromatographic or spectroscopic measurements made on each well on a 96 well plate.

The <experiment> element can also have an optional "name" attribute which describes the experiment for informational purposes.

The only valid elements in the <experiment> level of the hierarchy are <collectdate>, <parameter> and <trace>. The latter two can appear as many times as necessary to develop a complete record of the experiment; however, the <collectdate> can only appear once. For more discussion on the <parameter> element, refer to the "Meta-Data" section later in this chapter.

The <trace> Element

The <trace> element can only appear under the <experiment> element in the hierarchy and is used to delineate data that was taken from a single detector. For example, a complete set of mass spectral scans from an LC-MS run can be stored in a single <trace> section. However, if the experiment also entailed using a PDA detector for the same injection, then an additional <trace> section would appear in the document containing all the PDA spectra.

The <trace> elements are not intended to be used to encapsulate data from separate instrumental analyses done on the same sample. For example, an NMR spectrum and an FTIR spectrum of the same sample measured on separate instruments would not be stored as separate <trace> elements under the same <experiment>. Instead, each would be stored as a separate XML document or under a separate <experiment> element in the same document.

As there can be many types of data collected from a single experiment, there can be multiple <trace> elements within a single XML document. The <trace> element must identify the type of detector used to collect the data within the element. This will allow software that reads the file to determine if the data in a particular <trace> element is relevant without having to parse all of its data to imply the type. For this, each <trace> element will have an attribute called "technique." In order to ensure that software interpreting the data can easily determine the type of data without having to parse free-form strings, there will only be a specific set of allowed values for the "technique" attribute. The proposed list of potential "technique" attributes for the <trace> element is shown in Appendix A. Note that this list is likely to be incomplete at this point.

As discussed previously, all the <trace> elements encapsulated within the same <experiment> element can only come from a single analysis. When more than one <trace> element appears under a single <experiment>, it implies that the data contained in each <trace> are related. For example, when there is a <trace> element with a "technique" attribute set to "CHROM" (indicating an chromatogram) and another with a "technique" attribute set to "MS" (indicating mass spectral data), there is an implied relationship between them. In this case, it might be that two detectors were used (i.e., UV & MS), or that the chromatogram is the Total Ion Chromatogram from the MS spectra.

At the <trace> level in the hierarchy, there can be none, one, or multiple <Xdata>elements and one or more <coordinates> or <parameter> elements.

The <Xdata> Element

The <Xdata> element is used to encapsulate all information in a single <trace> element that is common to a single set of abscissa values for the measured data. There can be many <Ydata> elements encapsulated within this level of the data hierarchy, but there always must be at least one. This allows developers to make use of the XML hierarchy to gain some economy in data storage requirement. Although the simplest representation for data would be to pair the X and Y (abscissa and ordinate) data arrays together regardless of the data type, this would create unnecessarily large XML documents in some cases.

For example, in a set of PDA spectra measured from a chromatographic analysis, all use the same X values because each diode measures the same wavelength in the series of spectra. Storing the X values

for each spectral scan would effectively double the size of the document with no gain in descriptive knowledge of the data.

For instrument types that generate a series of arrays where the X data is variable between "scans" of the sample (i.e., a series of centroided spectra from and LC-MS analysis), a single <trace> element might contain a large number of <Xdata> elements, each with a single encapsulated <Ydata> element.

Each <Xdata> element is required to have a "units" attribute that describes the units of the abscissa values it encapsulates. The value of the "units" attribute can only come from a specific list of allowed settings. This is required so that applications parsing the data in the <Xdata> section of the document can exactly know the units of the data to allow for inter-unit transformation of the data arrays. It has been shown in precedent formats like JCAMP and ANDI that axis unit labels cannot be simple strings as developers tend to implement them without any thought to consistency and compatibility with other applications. While "Minutes," "Min.," "Time (min)" may all mean the same thing to a human viewing the text, it is nearly impossible for a programmer developing code to transform the array values into another unit (i.e., "Seconds") if he has to determine the current unit basis by examining the string for all possible textual representations of the term "Minutes." The list of unit attributes can also be used for any other data element in the format that needs unit descriptors. The proposed list of unit type attributes is listed in Appendix B and is likely to be incomplete at this point.

The <Xdata> element can have a number of optional attributes. The "name" attribute can be used to store additional textual information that must be kept with the X axis data values. This might be a description or some identifier relevant to the source data system that generated the data.

The optional "label" attribute can be used to notify a parsing program what text should be used to label the axis when displaying the data. Normally, this will be inferred from the setting of the "units" attribute, but the "label" attribute can be used if a different string is used in the source data system's display or reporting facilities.

The optional attribute "valueorder" can be used to indicate the ordering of the X axis data point values in the subsequent <values> element. This is optional due to the fact that parsing programs can read the array values and reorder them if necessary. One thing to note, however, is that the proposed data format assumes that there is a one-to-one correspondence between the data values in the <values> elements in the <Xdata> and <Ydata> parent elements. In other words, the first data value in the <values> element attached to the <Xdata> element corresponds to the first data value in the <values> element attached to the <Ydata> element. If the values attached to <Xdata> must be re-ordered for any purpose, the exact same re-ordering process must be applied to the values attached to <Ydata> to maintain the appropriate X,Y data pairings. The same logic applies to the values in any <altXdata> sub-elements.

The "linkid" attribute is used to link the axis data to another element in the same document and is described in detail in the <link> element section later in this document.

The <Xdata> element will always contain only one <values> element which will contain the actual abscissa data values (see below). It will also always contain at least one <Ydata> element; however, for some data instrument analyses, there can be many <Ydata> elements that all share the same <Xdata> (as mentioned previously).

The <Xdata> element can optionally contain one or more <altXdata>, <link> and <parameter> elements.

The <altXdata> Element

The <altXdata> element is used to store additional arrays of abscissa values for those experiments that make measurements as a function of multiple independent variables. For example, a thermal analyzer can measure the weight loss of a sample as it is heated. However, the change in temperature is usually not a simple ramp, but uses a temperature programmer that can adjust the temperature up, down or keep it constant at different points in the analysis. Therefore, each weight value (the ordinate information) is actually measured as a function of a specific time since the beginning of the analysis and the temperature of the sample at that point. Depending on what the programmer considers the "primary" X axis data, the <Xdata> element would contain one of the value sets (either time or temperature) and the <altXdata> element would be used to store the other.

There is no limit to the number of <altXdata> elements that can appear within a given <Xdata> element. This is entirely dependent on the number of alternative sets of abscissa values that must be carried along with the ordinate data values.

All of the attributes ("units," "name," "label" and "valueorder") and their associated restrictions that apply for <Xdata> also apply to <altXdata> elements. However, unlike <Xdata>, the <altXdata> element can only contain two elements itself: <values> and <parameter>.

As with the <Xdata> element, the "linkid" attribute is used to link the axis data to another element in the same document and is described in detail in the <link> element section later in this document.

The <Ydata> Element

The <Ydata> element encapsulates all information relating to the values read from the detector that are paired with the data values in the <Xdata> (and optional <altXdata>) elements. As mentioned in the discussion of the <Xdata> element, there can be one or more <Ydata> elements within any given <Xdata> element. All <Ydata> elements that appear within the same <Xdata> element use the same set of abscissa values.

As with the <Xdata> and <altXdata> elements, the <Ydata> element has a mandatory "units" attribute and supports the optional "name" and "label" attributes. See the discussion of these attributes in the <Xdata> element section for more information.

The <Ydata> element will always contain only one <values> element which will contain the actual measured ordinate data values (see below). The <Ydata> element can also contain a number of optional <peaktable> elements which encapsulate peak-related information calculated from the Y data values. The <peaktable> hierarchy is described in a later section in this document.

The <coordinates> Element

The <coordinates> element is used to store arrays of values that identify the location of the <Ydata> arrays in an additional coordinate system. For example, if a <trace> element encapsulates a set of spectra from a PDA detector used in a chromatographic analysis, there will be a single <Xdata> element with many <Ydata> sub-elements; one for each PDA spectrum collected during the separation. In this case, the <coordinate> element is used to store the array of the time values (from the start of the separation), one value for each <Ydata> element that follows within the same <trace> element.

Note that this implies that the number of data values in the array must match the number of <Ydata> elements found within the same <trace> element as the <coordinate> element. In addition, as with pairing the data values in the <Xdata> and <Ydata> elements, there is an implied relationship between the order of the values in the <coordinate> array and the order of the <Ydata> elements in the same <trace> element. The first value in a <coordinate> array applies to the first <Ydata> element, the second value in the same <coordinate> array applies to the second <Ydata> element, and so on.

There can be multiple <coordinate> elements in a single <trace> element. This is useful for experiment types that create higher order data sets. For example, spectral imaging data sets might store a large collection of spectra as wavelength values in the <Xdata> elements and reflectance values in the <Ydata> elements. There would be two <coordinate> elements within the same <trace>, one for the pixel position along the horizontal part of the image and one for the position on the vertical.

As with all elements in the format used to store arrays of data values, the <coordinate> element can have a number of attributes attached. All of the common attributes ("units," "name," and "label") and their associated restrictions that apply for <Xdata> and <Ydata> also apply to <coordinate> elements.

In addition, there can be an optional attribute "valueorder" to indicate the ordering of the coordinate data point values in the subsequent <values> element. This is optional due to the fact that parsing programs can read the array values and reorder them if necessary. As noted above, the proposed data format assumes that there is a one-to-one correspondence between the data values in the <values> element and the subsequent <Ydata> elements. In other words, the first data value in the <values> element attached to a <coordinate> element corresponds to the first <Ydata> element within the <trace> element.

If the values attached to <coordinate> must be re-ordered for any purpose, the parsing software must make sure to match the values to the appropriate <Ydata> element.

The <coordinate> element can also have a "linkid" attribute as well as one or more <link> sub-elements used to link the data to another element in the same document. This is described in detail in the <link> element section later in this document.

The <values> Element

The <values> element stores the actual arrays of data values. However, contrary to what has been done in prior ASCII-based file formats (such as JCAMP), the data values are not represented as strings of "readable" numbers. Instead, the data values are stored as ASCII strings of compressed binary data values using MIME Base64 encoding. The Base64 encoding mechanism allows binary streams of data to be stored as ASCII strings within XML files yet maintain the 'readability' of the original binary information. Base64 is actually a Internet standard and is used by nearly all Internet-enabled email software to transmit binary attachments as part of a standard email message.

The choice of Base64 encoding for the actual data values gets around one of the major flaws of ASCII-based data formats: loss of numerical precision. This was discussed earlier in the context of the JCAMP data exchange file format, but is worth reiterating. For example, to properly represent a complete 32-bit floating point value at full precision requires at least 14 ASCII characters (14 bytes), i.e., "-1.234567E-01". Storing the value in an ASCII file using fewer characters will cause the software that reads the file to generate a different number than the original 32-bit floating point value (it will effectively be a rounded version of the original number). On the other hand, a Base64 encoded representation of that same 32-bit floating point value can completely regenerate the *exact same value* as the original number when the ASCII data stream is decoded.

In order for a programmer to be able to implement a reader for the binary stream, the data must have attributes that describe the binary representation of the information once it has been decoded from the Base64 string. Therefore, the data arrays will always have a "format" attribute attached to them describing the binary representation of the decoded data. In order to maintain the best possible numerical accuracy while at the same time minimizing the complexity of the data format, only two binary format representations are allowed: "FLOAT32" (for IEEE 32-bit floating point representation) and "FLOAT64" (for IEEE 64-bit floating point representation).

In addition, as a majority of analytical instrumentation software are based on PC hardware architectures and operating system software, only high-ending byte ordering (a.k.a. Intel order) is allowed. Software developed for computer systems that use other byte ordering for binary values (i.e., UNIX, Mac) will have to transpose the bytes to properly read or store the data values. Therefore, to indicate this requirement to programmers who may be required to work with this data format, an additional "byteorder" attribute must be included in the <values> element which can only have the value "INTEL."

The <parameter> Element and Meta-Data

In addition to storing and describing the actual data values, the format must also have the ability to store alphanumeric information that describes how the data was collected and in some cases how it was processed. The precedent file formats (ANDI, JCAMP and SPC) used various technical approaches to solve this issue, but the basic concept was the same. That is, a group of people worked together to define a "dictionary" of terms that refer to specific instrument parameters relevant for a given type of data and published it in the public domain.

For example, in the ANDI Mass Spectrometry standard, a tag "*detector-type*" was created to store the name of the type of detector used on the spectrometer that collected the data. In the case of the ANDI Chromatography standard, this was called "*detector-name."* The JCAMP standard defines a tag for this parameter as well, called "*DETECTOR*" and it is defined as "*DET*" in the Galactic SPC file format.

One of the main reasons that earlier attempts at defining standard file formats generated such complex dictionaries was that the goal was to completely transfer data between various software systems tied to particular pieces of instrumentation. However, in many cases, it was very difficult for the instrument

vendors and software developers to map their data structures onto the items in the dictionary. Admittedly, a minimum set of parameters is required in any data system to allow it to properly interpret the "standard" files and translate them into its own internal representation for display and processing. However, to this day it is very unlikely that any instrument or software vendor's implementation of an "exported" data file in one of these standard formats can be read by all other vendors' software packages that supposedly support the same standard.

Given these examples, the problem of storing the meta-data in an XML file becomes immediately apparent; is the system of creating standardized parameter dictionaries even viable? The bottom line is that it is actually impossible to develop and maintain a complete dictionary of descriptive tags for all possible parameters from the vast variety of current instrumentation and vendors (and instruments yet to be invented) and still maintain an "accurate and complete" representation of the original data.

Therefore, for an analytical archiving format, it makes much more sense to keep the dictionary as small as possible and create a way that nearly any parameter can be stored in the file regardless of its relevance to any other data system. To that end, rather than trying to create a complex dictionary to handle meta-data from all the various instrument vendors, types, and software implementations, it is proposed to use a single element named <parameter> for all meta-data items.

The <parameter> element uses a series of attributes to describe the origin of the parameter and assign meaning to it for a human reading the file. In this manner, software systems reading a GAML file therefore don't have to try to interpret meaning to the data, but merely display the contents and the identifying attributes as information that is related to the data. It is then up to the person using the software to view the file (or reading the file itself) to interpret that parameter's meaning in the context of the data and the system used to collect it.

The "name" attribute is always required and is used to describe the parameter as it relates to the original data system software. Often this will be the name of the variable in the programmer's documentation for the software or the relevant name of the parameter in the file itself (i.e., in the case of parameters already stored in ASCII files). Similar to <Xdata> and the other value-oriented elements, the "label" attribute can optionally be used to assign an additional text string that describes the parameter. This can be used in software applications to display the parameter with meaning more useful to a human reader.

The optional "group" element is used to identify lists of parameters that exist in the same element as belonging to a logical group. This is especially useful in complex instrument control systems (i.e., Chromatography or NMR) where there are a large number of parameters related to making a measurement. This generic system of representing meta-data allows nearly any alphanumeric parameter present in a proprietary binary file format to be carried along when the data is translated to XML. In addition, the parameter and its value can be kept in context of the original data system that created the file. For example, the following is the XML representation of a set of parameters stored with a hypothetical chromatogram:

```
<parameter name="injvol" label="Injection Volume">6.00 ul</parameter>
<parameter name="dfac" label="Dilution Factor">2.5000</parameter>
<parameter name="AsampID" label="Autosampler Location">B124</parameter>
<parameter name="user" label="Analyst Name">Judy</parameter>
```

and here is a set of similar parameters from a different hypothetical chromatogram:

```
<parameter name="volume" label="Injection Volume">2.0000</parameter>
<parameter name="diluted" label="Dilution Factor">3 to 1</parameter>
<parameter name="cell_number" label="Autosampler Location">53</parameter>
<parameter name="run_by" label="Analyst Name">Bill Smith</parameter>
```

Note that there is no implied formatting and that the values themselves (the part in-between the start and end element markers) are effectively free-form information. A software package reading the information from either of these XML files could present it to the user as a list of parameters and their values. While the software itself cannot interpret the list, the information presented to the human in the context of viewing the data curve makes it relevant as this is how it might appear in the original software that created the data. This free-form mechanism for storing meta-data parameters is probably the only way to

achieve the goal of creating an "accurate and complete" representation of the original data that does not require continuous updating of the data format dictionary.

It is important to note that the <parameter> element can appear anywhere in the hierarchy, which is not true for all elements in the data format. This allows storing relevant parameters at any of the different levels of information represented by the XML hierarchy. In other words, parameters that are relevant to the overall experiment can be stored in <parameter> elements at the <experiment> level, while parameters that are specific to a single set of ordinate data values can be stored at the <Ydata> level.

The <parameter> element can also have an optional “group” attribute. This is intended to be used to identify a parameter as belonging to a particular collection of parameters that exist in the same level in GAML document hierarchy. The “group” attribute is especially useful for data originating from complex data systems such as chromatography and hyphenated sample analysis techniques such as LC-PDA-MS. These systems tend to have a large number of instrument control and processing parameters; however, they are usually broken down into ‘classes’ or ‘groups’ within the originating data system software. For example:

```
<parameter group="injection" name="injvol">6.00 ul</parameter>
<parameter group="injection" name="dfac">2.5000</parameter>
<parameter group="injection" name="AsampID" >B124</parameter>

<parameter group="instrument" name="flowrate" >1.5 ml/min</parameter>
<parameter group="instrument" name="columntemp" >27.5 C</parameter>

<parameter group="peak picking" name="pkthresharea" >27000</parameter>
<parameter group="peak picking" name="bunchfactor">11</parameter>
```

This allows parsers reading a GAML document to more easily retrieve sets of parameters that are logically connected to each other. It also provides an additional level of "readability" when GAML documents are viewed in their pure ASCII form.

The <integrity> Element

The <integrity> element is optionally used to store an encoded checksum of the contents of a GAML document to insure document integrity. This element can only appear once in the <GAML> element and applies across all content it contains. It does not apply to content outside the scope of the <GAML> element when it is embedded within a namespace in another XML document.

This element has one attribute, "algorithm," which currently is only allowed to have the value "SHA1". This indicates the algorithm used to encode the checksum value to prevent tampering with the data and the integrity check. This algorithm is described in the Federal Information Processing Standards Publication 180-1 (FIPS 180-1); see http://www.itl.nist.gov/fipspubs/fip180-1.htm for more details.

The content of the element is a string of 40 characters that are stored as ASCII string representations of the hexadecimal codes for the value of each individual byte in the encrypted checksum. For example, the ASCII value 122 is "7A" in hexBinary representation.

The intention of the <integrity> element is to allow applications that are using GAML files as a primary storage format the ability to detect changes to the data content that were made outside the control of the system that generated it

The <link> Element and "linkid" and "linkref" Attributes

The <link> element can be used within <coordinate>, <peaktable>, <Xdata> and <altXdata> elements to specify links between the data contained in that element and another element in the document. This is typically a set of data values associated with a shared set of axis values that exists in multiple <trace> elements within the document. Without the ability to link together data sets that exist in different element hierarchies within the document, parsing applications cannot determine the relationships between the data sets and therefore cannot display them in a way the user expects.

For example, assume an <experiment> element that contains data taken from an LC-MS analysis. In this case, there are two <trace> elements: one containing the set of mass spectral scans, the other containing

the RIC chromatogram. Ideally a software application that parses the data in this document would present the user with a view of both the chromatogram and the spectral scans. Since the RIC was generated from the mass spectral scans, the user would like to "click" on a point on the chromatogram and see the associated MS scan at that point. Alternatively, the user might be viewing a particular MS scan and want to see what point it appears in the RIC data. However, the time axis data that relates the two data sets is stored in unrelated places in two different <trace> elements within the document: as an <Xdata> element in the RIC and as a <coordinates> element in the MS scans.

To address this, a document like this would include "linkid" attributes in both the <coordinates> and <Xdata> elements in the two different <trace> elements. They would also include a <link> sub-element with a "linkref" attribute pointing to the name of the "linkid" in the other element. Below is a generalized example of what this might look like:

```
<GAML>                                                  root identifier
   <experiment>
      <trace technique="CHROM">                         extracted RIC chromatogram
         <Xdata units="MINUTES" linkid="RICTIME">       time values for chromatogram
            <link linkref="MSTIME">                     link to coordinates in spectra
            <values>
            <Ydata>
               <values>
               <peaktable>
      <trace technique="MS">                            mass spectral scans
         <coordinates linkid="MSTIME">                  time values for spectral scans
            <link linkref="RICTIME">                    link to time values in RIC
         <Xdata units="MASSZ">
            <values>
            <Ydata>
               <values>
         <Xdata units="MASSZ">
            <values>
            <Ydata>
               <values>
         .
         .
         .
```

The "linkid" attribute is what is known as an XML "ID" type attribute and "linkref" is an "IDREF" type. This system of linking is similar to the concept of bookmarks in HTML. It allows defining a link to another place in the document using a named link. One thing to note about this system is that the values of all "linkid" attributes must be unique throughout the scope of a single document.

The <link> element is used to point from one element's data to another's. The "linkref" attribute specifies the name of the unique "linkid" in the other data element to which it is linked. Note that there can be several <link> elements within a given data element. This allows relating a given element's data set to more than one other data set. This is especially useful for experiments that use multiple detectors all running simultaneously. In this case, a viewer application will want to be able to present all the linked data in a way that users can easily navigate the entire data set and know the relationships among all the <trace> elements in the experiment.

Generalized Peak Table Hierarchy

Any data format that is designed to store analytical instrument data must allow for storage of peak-related information as a fundamental data type. Although in most cases, this information is actually calculated from the measured data rather than a measurement itself. While this may seem to be the antithesis of defining a data format intended for the raw data from a given measurement, obtaining the peak-related information is often the primary reason for making the measurement in the first place.

The main reason it must be handled in the proposed data format is the fact that the algorithms used to calculate peaks in the instrument software are often proprietary and the results cannot be reproduced outside that software. Despite outward appearances, developing computer algorithms to correctly detect what a human sees as a "peak" in a set of data measurements is a very complex operation. Most instrument vendors keep these algorithms to themselves as industry secrets because they have evolved from many years of expensive engineering effort.

Therefore, since the goal of this data format is to create an "accurate and complete" representation of any measurement made on an instrument's data system, it is absolutely necessary to provide a mechanism for carrying the unique calculated peak information.

As with the generalized raw data representation given earlier, the following is a generalized overview of the proposed XML data structure for peak-related data. This is not an exact definition of how it will appear in an XML file, but rather a representation to familiarize the reader with the hierarchical nature of XML and how it can be used advantageously for this type of data.

```
<peaktable>
   <parameter>                      related to entire peak table
   <peak>                           has "number" and "group" attributes
      <peakXvalue>
      <peakYvalue>
      <parameter>                   "name" attribute used for Area, Name, etc.
      <baseline>                    peak baseline information
         <startXvalue>              required part of baseline structure
         <endXvalue>                required part of baseline structure
         <startYvalue>              required part of baseline structure
         <endYvalue>                required part of baseline structure
         <basecurve>                optional (for storing curved baselines)
            <baseXdata>             baseline curve X data
               <values>             array of base64 encoded values
            <baseYdata>             baseline curve Y data
               <values>             array of base64 encoded values
```

Important Note: The proposed peak table representation uses standard XML data elements to store the actual data values in ASCII. This is an apparent contradiction to the use of base64-encoded binary elements used to store the detector data values. The main reason is that it is believed that the precision required for the calculated peak values is less than for the measured data.

However, this aspect of the format is still open for comment. If after review, there are sufficient reasons to use a binary representation for calculated peak values, the format can be altered. In addition, the format can be altered in such a way as to allow storing the values either as standard XML elements or encoded binary. But, this must be done before the data format and Schema are released, as the <peaktable> element design in this proposal cannot be extended as written.

The <peaktable> Element

It is important to note from the earlier discussions on the raw data representation that the <peaktable> element can only appear within a <Ydata> element. This is due to the fact that the parameters describing a "peak" can only be calculated from a full set of X and Y data values. In addition, there is the possibility that there may be many peak tables within a <trace> element, one related to each <Ydata> element. For example, if a PDA detector was used on a chromatographic analysis, but rather than store full spectra for every time point, chromatograms are collected at only three wavelengths. Assuming that the final analysis method generated a peak report for each chromatogram, the data format must allow for storing

the peaks calculated from each of the three chromatograms even though they all exist in the same <trace> element.

Much like the <experiment> element in the top level of the data format hierarchy, the <peaktable> element is mainly a marker to encapsulate the peak-related information for the specific <Ydata> element in which it appears. A <peaktable> element can have an optional "name" attribute which describes the peak table for informational purposes and the only allowed sub-elements are <peak> and <parameter>.

The <peak> Element

The <peak> element encapsulates all data related to a single peak. This includes the peak location, descriptive information, and any calculated baseline information. As with the <peaktable> element, the <peak> element is mainly a marker to encapsulate the data for the peak and does not contain any data itself.

The <peak> element can have a number of optional attributes. The "number" attribute is a positive integer that indicates the number of the peak in the original software's referencing system. It can also have a "group" attribute which is a string that can define any type of grouping information that might be required to tie together different peaks in a single peak table. An optional "name" attribute can be included to identify the peak by a text string.

Every <peak> element must contain one <peakXvalue> and one <peakYvalue> element (discussed below). In addition, there can be an optional <baseline> element (also described in the following).

Note there are no elements for storing information like peak area, or height. This is due to the fact that these are not fundamental values that describe a peak, but are derived by using the data stored in the <peakXvalue>, <peakYvalue> and possibly the <baseline> elements and applying them to the data values stored in the <Xdata> and <Ydata> elements. Additionally, notice there are no elements for information such as the peak name (identity), quantity or other descriptive information.

Much like the issues with meta-data in the instrumental parameters, all the different instrument data systems have different terms for derived and calculated information they carry along with a peak. To attempt to define a dictionary for these widely varied parameter sets would lead to the same trap. Therefore, all parameters related to a peak other than those described here will be stored in the <peak> element using the very same <parameter> element and identified via their "name" attribute.

As all elements within a single <peaktable> element are encapsulated within the <Xdata> and <Ydata> elements, it is presumed that any values stored within them (i.e., <peakXvalue>, <peakYvalue>, all elements within a <baseline> element) are in the same units as the parent elements.

The <peakXvalue> and <peakYvalue> Elements

These elements are used to store the calculated location values for a single peak. The <peakXvalue> element describes the position of the peak along the abscissa, and the <peakYvalue> is for the ordinate location. Unlike the <values> elements that use encoded binary to store data values, these elements use ASCII strings. In investigations of different instrument data systems, it was found that the peak location values often did not require the same precision as the actual measured data values themselves. However, it is highly recommended that programmers implementing routines to read and write the values associated with the <peakXvalue> and <peakYvalue> elements use as many ASCII characters as are necessary to accurately represent them.

The <peakXvalue> is used to store the calculated X axis value for the position of the peak. In the case of a chromatogram data set, this is the retention time. In the case of optical spectroscopic measurements, this would be the wavelength or frequency of a peak. This could be the same as one of the values in the <values> element within the parent <Xdata> element encapsulating the <peaktable>. However, in most cases, it is not as the data system software will interpolate or extrapolate from the measured detector data values to calculate this value.

The <peakYvalue> is used to store the calculated Y value of the top of the peak. As with the <peakXvalue>element, this could be the same as one of the values in the <values> element within the <Ydata> element encapsulating the <peaktable>, but it is usually not due to interpolation or extrapolation.

It is important to note that the <peakYvalue> element is not used to store the calculated peak height. The concept of "peak height" is a measurement relative to a defined baseline. In the case where the baseline is perfectly flat and represents no offset in the measured signal, the measured "peak height" can be the same as <peakYvalue>, but they do not define the same measurement. As discussed in the description of the <peak> element, derived values like "peak height" and "peak area" are stored using the <parameter> element.

The <baseline> Element

The <baseline> item is an optional element used for storing baseline locations used when calculations were performed on the peak (i.e., for peak height or area). Like the <peakXvalue> and <peakYvalue> elements, the information stored in the <baseline> element generally cannot be reproduced outside the originating software that acquired and processed the data. Although there is no explicit limitation on what types of data can use the <baseline> element, it is expected that it will primarily be implemented in <trace> elements that contain chromatogram information.

A <baseline> element always contains four sub-elements: <startXvalue>, <endXvalue>, <startYvalue> and <endYvalue>. Like all elements in the proposed format, it can optionally contain a <parameter> element to describe meta-data information specific to the baseline of the peak.

The usage of the four required sub-elements should be reasonably apparent. The <startXvalue> and <endXvalue> elements contain the values along the abscissa that describe the beginning and end the section of data that is considered a "peak" for calculations. In the case of chromatographic data, these would be the retention time values at the beginning and end of the peak. The <startYvalue> and <endYvalue> describe the corresponding values on the ordinate. A line segment connecting the positions described by the <startXvalue>/<startYvalue> and <endXvalue>/<endYvalue> value pairs is the baseline for the peak.

The <baseline> element can also contain an optional <basecurve> element. This is used to store the data point values for curved baselines (which cannot be accurately represented by a straight line segment connecting the baseline begin and end values). The <basecurve> element consists of two sub-elements <baseYdata> and <baseXdata>. These sub-elements are similar to the <Xdata> and <Ydata> elements in a <trace> element in that they store the actual data points representing the baseline curve using a <values> array.

XML Examples Of Different Types of Analyses

The following are examples of the hierarchical structure of data from different types of analytical instrumentation as it would be represented in the proposed format. Note that these are not complete documents that can be put into software and interpreted into curves and meta-data. They are merely intended to show how the hierarchical nature of the XML language can be used to create data files that can store nearly any type of analytical instrument data.

Note that in all examples, the <peaktable> element is not expanded. It is presumed that it contains a full set of peak descriptors relevant for the type of data in the example. In addition, in some examples, the elements are shown with attributes for illustrative purposes only. Not all examples show all the possible optional or required attributes and their absence or presence should not be taken to infer that the example is a completely formed data set.

Chromatogram Example

Chromatograms are simple – only 2 arrays; one X and one Y with an attached peak table.

```
<GAML>
   <experiment>
      <trace technique="CHROM">
         <Xdata units="MINUTES" label="Time_(min)">
            <values>
            <Ydata units="MILLIVOLTS" label="mV">
               <values>
               <peaktable>
```

LC-PDA Spectra Example

In this example, the format is used to represent both the spectral data collected from the PDA detector and a reference chromatogram extracted from the spectral data and used for quantitation (indicated by data in the peak table). Note that since all spectra use the same set of wavelengths, all the Y arrays are sub-elements of a single X array.

```
<GAML>
   <experiment>
      <trace technique="CHROM">           extracted reference chromatogram
         <Xdata units="MINUTES">
            <link linkref="PDATIME">      link to coordinates array in spectra
            <values>
            <Ydata units="MABSORBANCE">
               <values>
               <peaktable>
      <trace technique="PDA">             PDA spectral data
         <coordinates linkid="PDATIME">   time values for all spectral scans
         <Xdata units="NANOMETERS">       all spectra (Y) use same wavelengths (X)
            <values>
            <Ydata units="MABSORBANCE">
               <values>
            <Ydata units="MABSORBANCE">
               <values>
            .
            .
            .
```

GC/LC-MS Spectra Example (Centroided Scans)

As with the previous LC-PDA example, there can be more than one <trace> element if there is a chromatogram extracted from the spectra with attached peak table information. However, in this case, each MS spectrum scan has its own X array since the number of XY pairs can be different for each scan. Thus there are multiple <Xdata> elements, each with only one <Ydata> sub-element.

```
<GAML>
   <experiment>
      <trace technique="CHROM">           extracted chromatogram (TIC, RIC, SIC, etc)
         <Xdata units="MINUTES">
            <link linkref="MSTIME">       link to coordinates array in spectra
            <values>
            <Ydata>
               <values>
               <peaktable>
      <trace technique="MS">              mass spectral data
         <coordinates linkid="MSTIME">    time values for all spectral scans
         <Xdata units="MASSZ">            each scan (Ydata) has its own Xdata
            <values>
            <Ydata>
               <values>
         <Xdata units="MASSZ">
            <values>
            <Ydata>
               <values>
         .
         .
         .
```

MS^n Spectra Example (Centroided Scans)

This is just an extension of the GC/LC-MS case. Here there are multiple coordinates for each scan. Note that the implementer must store the scans from the lowest level of fragmentation in the X and Y arrays and store the mass and time relationships in the coordinates.

```
<GAML>
   <experiment>
      <trace technique="MS">
         <coordinate units="SECONDS">     time values for spectrum scans
         <coordinate units="MASSZ">       parent mass(1) values for spectrum scans
         <coordinate units="MASSZ">       parent mass(2) values for spectrum scans
         <Xdata units="MASSZ">            each scan (Y) has own X array
            <values>
            <Ydata>
               <values>
         <Xdata units="MASSZ">
            <values>
            <Ydata>
               <values>
         .
         .
         .
```

TGA Analysis Example

Some types of data will have more than one set of X axis values (i.e., TGA can have time and temperature). In order to handle this, one set of X values serves as the “primary” while the additional arrays are stored as <altXdata> sub-elements. In most cases, there will only be one Y array, but the format does not impose that limitation.

```
<GAML>
   <experiment>
      <trace technique="THERMAL">
         <Xdata units="SECONDS">          primary X axis (i.e. time)
            <values>
            <altXdata units="DEGREES_C">  alternate X values (i.e. temp)
            <Ydata units="MILLIGRAMS">
               <values>
```

Simple Optical Spectroscopy Example (FTIR, UV-Vis, NIR, Raman, etc.)

This is actually the simplest case, just one X array and a corresponding Y array.

```
<GAML>
      <experiment>
            <trace technique="UVVIS">
                  <Xdata units="NANOMETERS">
                        <values>
                        <Ydata units="TRANSMISSION">
                              <values>
```

FTIR Spectroscopy Example (FTIR, UV-Vis, NIR, Raman, etc.)

In some cases, originating file formats store all the associated data from an analysis (i.e., for a single beam FTIR instrument, this might include the sample interferogram, the calculated spectrum, and the background reference spectrum). In this case, multiple <trace> elements can be used to store each piece of data and delineate them with the "name" attribute.

```
<GAML>
   <experiment>
      <trace technique="FTIR" name="Sample Interferogram">
         <Xdata units="ARBITRARY" label="Data Points">
            <values>
            <Ydata units="ARBITRARY" label="Energy">
               <values>
      <trace technique="FTIR" name="Sample Spectrum">
         <Xdata units="WAVENUMBER" label="Wavenumber_(cm-1)">
            <values>
            <Ydata>
               <array units="ABSORBANCE" label="Abs">
      <trace technique="FTIR" name="Background Reference Spectrum">
         <Xdata units="WAVENUMBER" label="Wavenumber_(cm-1)">
            <values>
            <Ydata units="ARBITRARY" label="Energy">
               <values>
```

1D NMR Spectroscopy Example

In modern FTNMR systems, there are actually two Y arrays collected simultaneously in the instrument (collectively known as a “FID” or Free Induction Decay). Having both arrays is what allows NMR processing software to Fourier transform and phase correct the signals into what the user sees as an NMR spectrum. This is usually only stored in one of the Y arrays (usually known as the “real” part of the signal); however, the second array (a.k.a. the “imaginary array") is absolutely required to properly reprocess the data. Therefore, the X array in the format will have two Y array sub-elements.

As with FTIR data, there can possibly be a set of original data that was collected off the detector (the FID signal) as well as a fully processed spectrum.

```
<GAML>
   <experiment>
      <trace technique="NMR" name="FID data">        raw FID signal data
         <Xdata units="SECONDS">                     Y arrays use same X values
            <values>
            <Ydata units="ARBITRARY" label="Real">  “real” values
               <values>
            <Ydata units="ARBITRARY" label="Imag">  “imaginary” values
               <values>
      <trace technique="NMR" name="Spectrum">        fully processed NMR spectrum
         <Xdata units="PPM">                         Y arrays use same X values
            <values>
            <Ydata units="ARBITRARY" label="Real">  “real” values
               <values>
            <Ydata units="ARBITRARY" label="Imag">  “imaginary” values
               <values>
```

nD NMR Spectroscopy Example

For multidimensional NMR, the hierarchy allows representation of data in nearly any dimensionality by including additional coordinates. Note that is will be very important to track the mapping of meta-data elements as to which coordinate system they belong to (i.e., nucleus frequency).

```
<GAML>
   <experiment>
      <trace technique="NMR">
         <coordinates>                  coordinate values for T1 dimension
         <coordinates>                  coordinate values for T2 dimension
         <Xdata>                        following Y arrays use same X values
            <values>
            <Ydata>                     “real” values
               <values>
            <Ydata>                     “imaginary” values
               <values>
         <Xdata>                        following Y arrays use same X values
            <values>
            <Ydata>                     “real” values
               <values>
            <Ydata>                     “imaginary” values
               <values>
         .
         .
         .
```

Imaging Spectroscopy Example

In this case, each Y array will have two or mode coordinates. These represent the alternative positioning of the data in the other dimensions. For example, in the case of an imaging FTIR system with a moving sample stage, the arrays will store actual spectra (X=wavelength, Y=response) and the coordinates will be the horizontal and vertical positions on the stage. In the case of CCD imaging spectrometers, each array will be a set of diodes on the array (X=horizontal position, Y=intensity) and the coordinates will be the wavelength and vertical position. Note that in this and other higher order data types, it will be up to the application programmer reading the data from the file to properly orient the data to the way that the user expects to see it, even though it may not be stored in the file that way.

```
<GAML>
   <experiment>
      <trace>                                For imaging spectroscopy
         <coordinates units="MICRONS">       pixel distance values in D1 dimension
         <coordinates units="MICRONS">       pixel distance values in D2 dimension
         <Xdata units="WAVENUMBER">
            <values>
            <Ydata units="REFLECTANCE">
               <values>
            <Ydata units="REFLECTANCE">
               <values>
```

Encapsulation of Data in Other XML Formats

XML can be used to define schemas for whatever standard you want to use. XML allows any number of schemas to be referenced within the framework of a document by using "name spaces." Basically this just points the parser to a different XML schema when the new name space is encountered in the document.

Therefore, if it is necessary to encapsulate additional information within the data that cannot be described by the elements and attributes in that schema, data can be included within the same file, but use a different schema. This can also be used for encapsulating analytical instrument data into an XML document that is interpreted and validated using a different schema.

This is especially useful if there is structured data that must be stored within the context of the analytical instrument data, but cannot be described by the constructs of this data format. For example, if an instrument had a complex set of calibration parameters which required a special schema or contained chemical structures that could be represented in another XML schema (i.e., CML).

```
<GAML>
<experiment>                                parent namespace uses this proposed schema
   <trace>                                  data from a single detector
      <CML:cml xmlns:CML="x-schema:cml_schema_02.xsd">
                                            new namespace using the "CML" schema
                                            data represented here is parsed using
                                            the different schema, but is stored in
                                            the same file.
      </CML:cml>                            end of "CML" namespace
      <Xdata>                               remainder of data is in parent namespace
         <values>
         <Ydata>
            <values>
            <coordinate>
            <peaktable>
```

In addition, the proposed format can be encapsulated into documents using other data types by identifying the information as coming from a different schema within the construct of the overall document.

```
<project>                                   hierarchies in parent document
   <notebook>
      <sample>
         <plate>
            <gaml:GAML xmlns:gaml="gaml.xsd">       beginning of data in this format
               <experiment>                 contents of this section are parsed using
                  .                         the schema described in this document
                  .
                  .
               </experiment>
            </gaml:GAML>                    end of data in this format
            <report>
               .
               .
               .
```

Appendices

Appendix A – Enumerated Attribute List of Trace Techniques

Defined String	Instrumental Analysis Technique
ATOMIC	Atomic Spectrum
CHROM	Chromatogram
FLUOR	Fluorescence Spectrum
IR	Infrared Spectrum (scanning or FT)
MS	Mass Spectrum (Continuum, Centroid, TOF, etc.)
NIR	NIR Spectrum
NMR	NMR Spectrum
PDA	Chromatography Photodiode Array Spectra
PARTICLE	Particle Size Distribution
POLAR	Spectral Polarimetry (i.e., Circular Dichroism)
RAMAN	Raman Spectrum (scanning, diode or FT)
THERMAL	Thermal Analysis (TGA, DSC, etc.)
UNKNOWN	Unknown technique
UVVIS	UV-VIS Spectrum
XRAY	X-ray Spectrum (XRD, XRF, etc.)

Appendix B – Enumerated Attribute List of Axis Units

Defined String	Unit Type
ABSORBANCE	Absorbance (Abs)
AMPERES	Amperes
ANGSTROMS	Angstroms (A°)
ATOMICMASSUNITS	Atomic Mass Units (amu)
CALORIES	Calories (cal)
CELSIUS	Temperature in °C (C)
CENTIMETERS	Centimeters (cm)
DAYS	Days
DECIBELS	Decibel
DEGREES	Degrees (along an angle i.e., 90°)
ELECTRONVOLTS	Electron Volts (eV)
EMISSION	Emission
FAHRENHEIT	Temperature in °F (F)
GIGAHERTZ	Gigahertz (GHz)
GRAMS	Grams (g)
HERTZ	Hertz (Hz)
HOURS	Hours
JOULES	Joules (J)
KELVIN	Temperature in °K (K)
KILOCALORIES	Kilocalories (Kcal)
KILOGRAMS	Kilograms (kg)
KILOHERTZ	Kilohertz (KHz)
KILOMETERS	Kilometer (km)
KILOWATTS	Kilowatts (KW)
KUBELKAMUNK	Kubelka-Munk
LITERS	Liters (l)
LOGREFLECTANCE	Log (1/Reflectance)
MASSCHARGERATIO	Mass-to-charge ratio (M/z)
MEGAHERTZ	Megahertz (MHz)
MEGAWATTS	Megawatts (MW)
METERS	Meters (m)
MICROGRAMS	Micrograms (ug)
MICRONS	Microns (um)
MICROSECONDS	Microseconds (uS)
MILLIABSORBANCE	Milliabsorbance (mAbs)
MILLIAMPS	Milliamperes

Appendix B – Enumerated Attribute List of Axis Units (cont.)

Defined String	Unit Type
MILLIGRAMS	Milligrams (mg)
MILLILITERS	Milliliters (ml)
MILLIMETERS	Millimeters (mm)
MILLIMOLAR	Millimolar
MILLISECONDS	Milliseconds (mS)
MILLIVOLTS	Millivolts
MILLIWATTS	Milliwatts (mW)
MINUTES	Minutes
MOLAR	Molar
MOLES	Moles
NANOGRAMS	Nanograms (ng)
NANOMETERS	Nanometers (nm)
NANOSECONDS	Nanoseconds (nS)
PPB	Parts per Billion (ppb)
PPM	Parts per Million (ppm)
PPT	Parts per Thousand (ppt)
RADIANS	Radians
RAMANSHIFT	Raman Shift (cm-1)
REFLECTANCE	Reflectance
SECONDS	Seconds
TRANSMISSIONPERCENT	% Transmission
TRANSMITTANCE	Transmittance
UNKNOWN	Unknown
VOLTS	Volts (V)
WATTS	Watts (W)
WAVENUMBER	Wavenumber (cm-1)
YEARS	Years

Appendix C – Enumerated Attribute List of Array Data Formats

Defined String	Data Value Type
FLOAT32	IEEE single precision floating point
FLOAT64	IEEE double precision floating point

Appendix D – Enumerated Attribute List of Data Value Ordering

Defined String	Data Value Type
EVEN	The difference between any two adjacent values in the data array is a constant value and the data values are in canonical order (either ascending or descending).
ORDERED	The data values are in canonical order (either ascending or descending), but the difference between adjacent values is variable.
UNSPECIFIED	The data value ordering is not specified; if needed the reading application will have to examine the <values> array to make a determination.

Appendix E – GAML Element and Attribute Definitions

The following is an alphabetical list of the defined Elements and Attributes in the Generalized Analytical Markup Language (GAML) data format and their restrictions and relationships.

Element Specifications

altXdata			
Content	eltOnly		
Type			
Member of	Xdata		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
link	0	unbounded	no
parameter	0	unbounded	no
values	1	1	yes
Attributes	**Type**	**Default**	**Required**
units	units		yes
label	label		no
linkid	linkid		no
valueorder	valueorder		no
Description			
Stores a secondary set of X values to be used with the <Ydata> values encapsulated in the same <Xdata> element. The "units" attribute is an enumerated list and unambiguously identifies the unit of measurement for the contained array of values. Required in order to allow inter-unit conversions (i.e., minutes to seconds, nanometers to micrometers). The "label" attribute can be used for storing alternate strings to label the axis.			

basecurve			
Content	eltOnly		
Type			
Member of	baseline		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
baseXdata	1	1	yes
baseYdata	1	1	yes
Attributes	**Type**	**Default**	**Required**
Description			
Encapsulates required elements for representation of a curved baseline for a peak. Note the sub-elements that are required can appear only once.			

baseline			
Content	eltOnly		
Type			
Member of	peak		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
startXvalue	1	1	yes
endXvalue	1	1	yes
startYvalue	1	1	yes
endYvalue	1	1	yes
Attributes	**Type**	**Default**	**Required**
Description			
Encapsulates required elements for representation of peak baselines. Note the sub-elements that are required can appear only once.			

baseXdata			
Content	eltOnly		
Type			
Member of	basecurve		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
values	1	1	yes
Attributes	**Type**	**Default**	**Required**
Description			
Stores X data values for a curved peak baseline.			

baseYdata			
Content	eltOnly		
Type			
Member of	basecurve		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
values	1	1	yes
Attributes	**Type**	**Default**	**Required**
Description			
Stores Y data values for a curved peak baseline.			

collectdate			
Content	textOnly		
Type	dateTime		
Member of	experiment		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**

Description

Date and time experiment run was made. Not necessarily equivalent to the date/time stamp on the originating operating system data file. Uses the standard date & time representation using the [ISO 8601] extended date format:

CCYY-MM-DDThh:mm:ss.

(Note the "T" character is a separator for the date and time parts.) This allows for specifying the date and time in local terms, or relative to GMT.

When the string is followed by the character "Z" this indicates the value is specified in Coordinated Universal Time (UTC) (i.e., `1999-12-31T23:00:00Z`).

When followed by a value with a + or – this indicates the local time as a difference from GMT (i.e., 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), would be written as `1999-05-31T13:20:00-05:00`).

coordinates			
Content	eltOnly		
Type			
Member of	trace		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
link	0	unbounded	no
parameter	0	unbounded	no
values	1	1	yes
Attributes	**Type**	**Default**	**Required**
units	units		yes
label	label		no
linkid	linkid		no
valueorder	valueorder		no

Description

Encapsulates a single set of coordinate values corresponding to the <Ydata>elements within the same <trace> element.

Note that this assumes that there is a one-to-one correspondence of the order of the values in the <values> elements attached to the <coordinates> element and the number of <Ydata> elements. In other words, the first data value in the <values> element attached to a <coordinates> element corresponds to the first <Ydata> element. If the values attached to <coordinates> must be re-ordered for any purpose, the correspondence of the appropriate <Ydata> elements must be maintained.

The "units" attribute is an enumerated list and unambiguously identifies the unit of measurement for the contained array of values. Required in order to allow inter-unit conversions (i.e., minutes to seconds, nanometers to micrometers).

The "label" attribute can be used for storing alternate strings to label the axis.

The "valueorder" attribute is optional and is an enumerated list used to indicate the order of the data values in the <values> element attached to <coordinates>. If "valueorder" is not present, the reader cannot assume that the data values are in any particular order and reading the entire array's values before attempting any operations on the data is recommended. Please see note above regarding matching of values in data arrays.

endXvalue			
Content	textOnly		
Type	double		
Member of	baseline		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**

Description

Ending X value for a peak baseline.

EndYvalue			
Content	TextOnly		
Type	Double		
Member of	Baseline		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
Description			
Ending Y value for a peak baseline.			

Experiment			
Content	EltOnly		
Type			
Member of	GAML		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
collectdate	0	1	no
parameter	0	unbounded	no
trace	1	unbounded	yes
Attributes	**Type**	**Default**	**Required**
name	name		no
Description			
Encapsulates all data taken in a single run; <trace> sub-elements are assumed to have come from the same measurement or analysis. The "name" attribute can be used to assign a string label to the experiment.			

GAML			
Content	eltOnly		
Type			
Member of	None (root element)		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
parameter	0	unbounded	no
experiment	1	unbounded	no
Attributes	**Type**	**Default**	**Required**
version	string	"1.00"	yes
name	name		no
Description			
Root element for all documents in this format. Allows encapsulation of multiple <experiment> elements for representing sample sequences (i.e., chromatography injection series, multi-well plates, etc.). The "name" attribute can be used to assign a string label to the document.			

link			
Content	eltOnly		
Type			
Member of	altXdata, coordinates, Xdata		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
linkref	linkref		yes
Description			
Defines a forward link to an element containing a <values> of values within the document that is linked to the data in the parent element. Must specify the name of a valid "linkid" attribute within the document. Used to link the data in the parent element encapsulating the <link> element to data in another element as part of the same <experiment>.			

parameter			
Content	textOnly		
Type	string		
Member of	altXdata, coordinates, experiment, peak, peaktable, trace, Xdata, Ydata		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
group			no
name	name		yes
label	label		no
Description			
For storage of generalized parameter (meta-data) information; can appear as a member of a large number of elements in the format. The attributes assign meaning to the data in the <parameter> element as important information relevant to the parent element in which it appears. The "name" attribute is used for describing the parameter in terms of the originating data system; often this is a variable name in programmer's documentation, or shorthand notation for the parameter. The "label" attribute is used for more human-friendly descriptions of the parameter; for example, how it might be described in the original software user interface. The "group" attribute is used to assign an additional level of linking between parameters that exist within the same element. This is especially useful for instrument data control systems that may need to store many parameters to completely describe how separate parts of the overall system were configured.			

peak			
Content	eltOnly		
Type			
Member of	peaktable		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
peakXvalue	1	1	yes
peakYvalue	1	1	yes
parameter	0	unbounded	no
baseline	0	1	no
Attributes	**Type**	**Default**	**Required**
number	positiveInteger		yes
group	string		no
name	name		no
Description			
Encapsulates the relevant elements for a single peak. The "number" attribute is required and is used to enumerate the peaks as they should appear in the originating format. The "group" attribute is optional and allows including additional textual information on the peak and how it relates to the remaining <peak> elements within the same <peaktable> element. The "name" attribute can be used to assign a string label to the peak.			

peaktable			
Content	eltOnly		
Type			
Member of	Ydata		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
parameter	0	unbounded	no
peak	1	unbounded	yes
Attributes	**Type**	**Default**	**Required**
name	name		no
Description			
Encapsulates a list of <peak> elements describing a list of peaks for the parent <Ydata> element. The "name" attribute can be used to assign a string label to the peak table.			

peakXvalue			
Content	textOnly		
Type	double		
Member of	peak		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
Description			
X value of peak position.			

peakYvalue			
Content	textOnly		
Type	double		
Member of	peak		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
Description			
Y value of peak position.			

startXvalue			
Content	textOnly		
Type	double		
Member of	baseline		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
Description			
Beginning X value for a peak baseline.			

startYvalue			
Content	textOnly		
Type	double		
Member of	baseline		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
Description			
Beginning Y value for a peak baseline.			

trace			
Content	eltOnly		
Type			
Member of	experiment		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
parameter	0	unbounded	no
coordinates	0	unbounded	no
Xdata	0	unbounded	no
Attributes	**Type**	**Default**	**Required**
name	name		no
technique	technique		yes
Description			
Encapsulates all data taken from a single detector. The required "technique" attribute is an enumerated list and unambiguously identifies the data content of the element. The "name" attribute can be used to attach a descriptive string to the <trace> element.			

values			
Content	textOnly		
Type	base64Binary		
Member of	Xdata, altXdata, Ydata, coordinates		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
Attributes	**Type**	**Default**	**Required**
format	format		yes
byteorder	byteorder		yes
numvalues	positiveInteger		no

Description

The data in this element is a linear array of MIME base64 encoded binary data values. The identity and usage of data values are implied by the parent element (Xdata, altXdata or Ydata). The values are always stored in either 32-bit (single) or 64-bit (double) floating point representation which is indicated by the value of the enumerated "format" attributes.

The "byteorder" attribute indicates the storage representation of the binary floating point values in the decoded array. Currently, only one value is allowed: "INTEL".

The optional "numvalues" attribute can be used to indicate the number of values in the array making it easier for parsers to allocate space for decoding the array of points.

Note: When using the current Microsoft parser and XML DOM (Document Object Model) component, there is an additional attribute "dt=dt:bin.base64" to indicate the encoding of the binary data into ASCII for inclusion in an XML document along with the appropriate reference to the namespace (xmlns:dt="urn:schemas-microsoft-com:datatypes"). When XML Schema language has been officially approved, this will be unnecessary as the Schema will specify the "base64Binary" data type instead.

Xdata			
Content	eltOnly		
Type			
Member of	trace		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
link	0	unbounded	no
parameter	0	unbounded	no
values	1	1	yes
altXdata	0	unbounded	no
Ydata	1	unbounded	yes
Attributes	**Type**	**Default**	**Required**
units	units		yes
label	label		no
linkid	linkid		no
valueorder	valueorder		no

Description

Encapsulates all data related to a single set of X values. Hierarchy indicates relationship of data values: all encapsulated <Ydata> uses the parent <Xdata>. However, a <trace> can consist of multiple <Xdata> elements each with its own <Ydata> element(s).

Note that this assumes that there is a one-to-one correspondence of the order of the values in the <values> elements attached to the <Xdata> and <Ydata> elements. In other words, the first data value in the <values> element attached to the <Xdata> element corresponds to the first data value in the <values> element attached to the <Ydata> element. If the values attached to <Xdata> must be re-ordered for any purpose, the exact same re-ordering process must be applied to the values attached to <Ydata> to maintain the appropriate X,Y data pairings. The same logic applies to the values in any <altXdata> sub-elements.

The "units" attribute is an enumerated list and unambiguously identifies the unit of measurement for the contained array of values. Required in order to allow inter-unit conversions (i.e., minutes to seconds, nanometers to micrometers).

The "label" attribute can be used for storing alternate strings to label the axis.

The "valueorder" attribute is optional and is an enumerated list used to indicate the order of the data values in the <values> element attached to <Xdata>. If "valueorder" is not present, the reader cannot assume that the data values are in any particular order and reading the entire array's values before attempting any operations on the data is recommended. Please see note above regarding matching of values in data arrays.

Ydata			
Content	eltOnly		
Type			
Member of	Xdata		
Member Elements	**MinOccurs**	**MaxOccurs**	**Required**
parameter	0	unbounded	no
values	1	1	yes
peaktable	0	unbounded	no
Attributes	**Type**	**Default**	**Required**
units	units		yes
label	label		no
Description			
Encapsulates all data related to a single set of X values. Hierarchy indicates relationship of data values: all encapsulated <Ydata> uses the parent <Xdata>. However, a <trace> can consist of multiple <Xdata> elements, each with its own <Ydata> element(s). The "units" attribute is an enumerated list and unambiguously identifies the unit of measurement for the contained array of values. Required in order to allow inter-unit conversions (i.e., Transmission to Absorbance). The "label" attribute can be used for storing alternate strings to label the axis.			

Attribute Specifications

byteorder	
Type	NMTOKENS
Required	Yes
Values	INTEL
Description	Describes the byte ordering for <values> elements.

format	
Type	NMTOKENS
Required	Yes
Values	See Appendix C
Description	Describes the decoded data format for <values> elements.

label	
Type	string
Required	No
Values	
Description	General attribute used on some elements with data values to allow supplying an alternate label string for the data values. Can be used for axis labels for data types not defined by an enumerated value listed in "units" attribute.

linkid	
Type	ID
Required	No
Values	
Description	Unique identifier for links from other elements in the data format. The value assigned to this attribute must be unique throughout the entire document. Used to link axis information together; generally only found in <altXdata>, <coordinate>, <peaktable> and <Xdata> elements.

linkref	
Type	IDREF
Required	No
Values	
Description	Link pointer to an element with an attached "linkid" attribute. Can only be found in <link> elements.

name	
Type	string
Required	No
Values	
Description	General attribute used by many elements to allow supplying a descriptive string for the element's contents.

technique	
Type	NMTOKENS
Required	Yes
Values	See Appendix A
Description	Enumerated list of analytical instrument types for uniquely identifying contents of <trace> elements.

units	
Type	NMTOKENS
Required	Yes
Values	See Appendix B
Description	Enumerated list of data unit types for uniquely identifying units of data values in subsequent <values> elements.

valueorder	
Type	NMTOKENS
Required	No
Values	See Appendix D
Description	Allows specifying data value ordering <values> elements within <coordinates> and <Xdata> elements to simplify logic for parsers.

Appendix F – GAML Schema Definition (XSD)

The following XML Schema Definition for the Generalized Analytical Mark Language data format (GAML) was developed according to the XML Schema definition language proposal as it was last published as a W3C Final Recommendation on May 2nd, 2001 (http://www.w3.org/TR/xmlschema-2/).

As of the writing of this document, no commercial XML parsers or products fully support validation of XML documents against a Schema developed to this standard. However, Microsoft has released a "technology preview" version (4.0) of their XML Parser COM object on their web site (http://msdn.microsoft.com/xml/general/newinaprilre.asp) which supports the XML Schema Proposed Recommendation as of March 30, 2001. The final version of XML Schema is not substantially different from this version and it is likely that the Microsoft object will be updated shortly to conform to the final release.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Schema for Generalized Analytical Markup Language (GAML).  MRU 6-29-01
James Duckworth, Thermo Galactic (James.Duckworth@thermogalactic.com)-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xsd:annotation>
      <xsd:documentation>Schema for GAML data model for archival storage of
analytical instrument measurement data.</xsd:documentation>
   </xsd:annotation>

   <xsd:element name="GAML">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="experiment" maxOccurs="unbounded"/>
            <xsd:element name="integrity" minOccurs="0">
               <xsd:complexType>
                  <xsd:simpleContent>
                     <xsd:extension base="xsd:hexBinary">
                        <xsd:attribute name="algorithm" use="required">
                           <xsd:simpleType>
                              <xsd:restriction base="xsd:NMTOKEN">
                                 <xsd:enumeration value="SHA1"/>
                              </xsd:restriction>
                           </xsd:simpleType>
                        </xsd:attribute>
                     </xsd:extension>
                  </xsd:simpleContent>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
         <xsd:attribute name="version" type="xsd:string" use="required"/>
         <xsd:attribute name="name" type="name" use="optional"/>
      </xsd:complexType>
   </xsd:element>
```

```
   <xsd:element name="experiment">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="collectdate"/>
            <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="trace" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="name" use="optional"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="trace">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="coordinates" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element ref="Xdata" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="name" type="name" use="optional"/>
         <xsd:attribute name="technique" type="technique" use="required"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="coordinates">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="link" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="values"/>
         </xsd:sequence>
         <xsd:attribute name="units" type="units" use="required"/>
         <xsd:attribute name="label" type="label" use="optional"/>
         <xsd:attribute name="linkid" type="linkid" use="optional"/>
         <xsd:attribute name="valueorder" type="valueorder" use="optional"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="Xdata">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="link" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="values"/>
            <xsd:element ref="altXdata" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="Ydata" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="units" type="units" use="required"/>
         <xsd:attribute name="label" type="label" use="optional"/>
         <xsd:attribute name="linkid" type="linkid" use="optional"/>
         <xsd:attribute name="valueorder" type="valueorder" use="optional"/>
      </xsd:complexType>
   </xsd:element>
```

```
<xsd:element name="altXdata">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="link" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="values"/>
      </xsd:sequence>
      <xsd:attribute name="units" type="units" use="required"/>
      <xsd:attribute name="label" type="label" use="optional"/>
      <xsd:attribute name="linkid" type="linkid" use="optional"/>
      <xsd:attribute name="valueorder" type="valueorder" use="optional"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Ydata">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="values"/>
         <xsd:element ref="peaktable" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="units" type="units" use="required"/>
      <xsd:attribute name="label" type="label" use="optional"/>
   </xsd:complexType>
</xsd:element>

<xsd:element name="values">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="xsd:base64Binary">
            <xsd:attribute name="format" use="required">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:NMTOKEN">
                     <xsd:enumeration value="FLOAT32"/>
                     <xsd:enumeration value="FLOAT64"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="byteorder" use="required">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:NMTOKEN">
                     <xsd:enumeration value="INTEL"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="numvalues" use="optional">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:positiveInteger"/>
               </xsd:simpleType>
            </xsd:attribute>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>
```

```
   <xsd:element name="peaktable">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="peak" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element ref="parameter" minOccurs="0"
maxOccurs="unbounded"/>
                     <xsd:element name="peakXvalue">
                        <xsd:complexType>
                           <xsd:simpleContent>
                              <xsd:restriction base="xsd:double"/>
                           </xsd:simpleContent>
                        </xsd:complexType>
                     </xsd:element>
                     <xsd:element name="peakYvalue">
                        <xsd:complexType>
                           <xsd:simpleContent>
                              <xsd:restriction base="xsd:double"/>
                           </xsd:simpleContent>
                        </xsd:complexType>
                     </xsd:element>
                     <xsd:element name="baseline" minOccurs="0">
                        <xsd:complexType>
                           <xsd:sequence>
                              <xsd:element name="startXvalue">
                                 <xsd:complexType>
                                    <xsd:simpleContent>
                                          <xsd:restriction base="xsd:double"/>
                                    </xsd:simpleContent>
                                 </xsd:complexType>
                              </xsd:element>
                              <xsd:element name="startYvalue">
                                 <xsd:complexType>
                                    <xsd:simpleContent>
                                          <xsd:restriction base="xsd:double"/>
                                    </xsd:simpleContent>
                                 </xsd:complexType>
                              </xsd:element>
                              <xsd:element name="endXvalue">
                                 <xsd:complexType>
                                    <xsd:simpleContent>
                                          <xsd:restriction base="xsd:double"/>
                                    </xsd:simpleContent>
                                 </xsd:complexType>
                              </xsd:element>
                              <xsd:element name="endYvalue">
                                 <xsd:complexType>
                                    <xsd:simpleContent>
                                          <xsd:restriction base="xsd:double"/>
                                    </xsd:simpleContent>
                                 </xsd:complexType>
                              </xsd:element>
                              <xsd:element name="basecurve" minOccurs="0">
                                 <xsd:complexType>
                                    <xsd:sequence>
                                          <xsd:element name="baseXdata">
                                                <xsd:complexType>
                                                      <xsd:sequence>
```

```
                                                                  <xsd:element
ref="values"/>
                                                            </xsd:sequence>
                                                      </xsd:complexType>
                                                </xsd:element>
                                                <xsd:element name="baseYdata">
                                                      <xsd:complexType>
                                                            <xsd:sequence>
                                                                  <xsd:element
ref="values"/>
                                                            </xsd:sequence>
                                                      </xsd:complexType>
                                                </xsd:element>
                                          </xsd:sequence>
                                    </xsd:complexType>
                               </xsd:element>
                               <xsd:element ref="parameter" minOccurs="0"
maxOccurs="unbounded"/>
                         </xsd:sequence>
                      </xsd:complexType>
                   </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="number" use="required">
                   <xsd:simpleType>
                      <xsd:restriction base="xsd:positiveInteger"/>
                   </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="group" type="group" use="optional"/>
                <xsd:attribute name="name" type="name" use="optional"/>
             </xsd:complexType>
          </xsd:element>
       </xsd:sequence>
       <xsd:attribute name="name" type="name" use="optional"/>
    </xsd:complexType>
 </xsd:element>

 <xsd:element name="parameter">
    <xsd:complexType>
       <xsd:simpleContent>
          <xsd:extension base="xsd:string">
             <xsd:attribute name="group" type="group" use="optional"/>
             <xsd:attribute name="name" type="name" use="required"/>
             <xsd:attribute name="label" type="label" use="optional"/>
          </xsd:extension>
       </xsd:simpleContent>
    </xsd:complexType>
 </xsd:element>

 <xsd:element name="link">
    <xsd:complexType>
       <xsd:attribute name="linkref" type="linkref" use="required"/>
    </xsd:complexType>
 </xsd:element>

 <xsd:element name="collectdate">
    <xsd:simpleType>
       <xsd:restriction base="xsd:dateTime"/>
    </xsd:simpleType>
 </xsd:element>
```

```
<xsd:simpleType name="group">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="label">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="linkid">
   <xsd:restriction base="xsd:ID"/>
</xsd:simpleType>

<xsd:simpleType name="linkref">
   <xsd:restriction base="xsd:IDREF"/>
</xsd:simpleType>

<xsd:simpleType name="name">
   <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="technique">
   <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="ATOMIC"/>
      <xsd:enumeration value="CHROM"/>
      <xsd:enumeration value="FLUOR"/>
      <xsd:enumeration value="IR"/>
      <xsd:enumeration value="MS"/>
      <xsd:enumeration value="NIR"/>
      <xsd:enumeration value="NMR"/>
      <xsd:enumeration value="PDA"/>
      <xsd:enumeration value="PARTICLE"/>
      <xsd:enumeration value="POLAR"/>
      <xsd:enumeration value="RAMAN"/>
      <xsd:enumeration value="THERMAL"/>
      <xsd:enumeration value="UNKNOWN"/>
      <xsd:enumeration value="UVVIS"/>
      <xsd:enumeration value="XRAY"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="units">
   <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="ABSORBANCE"/>
      <xsd:enumeration value="AMPERES"/>
      <xsd:enumeration value="ANGSTROMS"/>
      <xsd:enumeration value="ATOMICMASSUNITS"/>
      <xsd:enumeration value="CALORIES"/>
      <xsd:enumeration value="CELSIUS"/>
      <xsd:enumeration value="CENTIMETERS"/>
      <xsd:enumeration value="DAYS"/>
      <xsd:enumeration value="DECIBELS"/>
      <xsd:enumeration value="DEGREES"/>
      <xsd:enumeration value="ELECTRONVOLTS"/>
      <xsd:enumeration value="EMISSION"/>
      <xsd:enumeration value="FAHRENHEIT"/>
      <xsd:enumeration value="GHERTZ"/>
      <xsd:enumeration value="GRAMS"/>
      <xsd:enumeration value="HERTZ"/>
      <xsd:enumeration value="HOURS"/>
      <xsd:enumeration value="JOULES"/>
      <xsd:enumeration value="KELVIN"/>
      <xsd:enumeration value="KILOCALORIES"/>
      <xsd:enumeration value="KILOGRAMS"/>
      <xsd:enumeration value="KILOHERTZ"/>
      <xsd:enumeration value="KILOMETERS"/>
      <xsd:enumeration value="KILOWATTS"/>
      <xsd:enumeration value="KUBELKAMUNK"/>
      <xsd:enumeration value="LITERS"/>
      <xsd:enumeration value="LOGREFLECTANCE"/>
      <xsd:enumeration value="MASSCHARGERATIO"/>
      <xsd:enumeration value="MEGAHERTZ"/>
      <xsd:enumeration value="MEGAWATTS"/>
      <xsd:enumeration value="METERS"/>
      <xsd:enumeration value="MICROGRAMS"/>
      <xsd:enumeration value="MICRONS"/>
      <xsd:enumeration value="MICROSECONDS"/>
      <xsd:enumeration value="MILLIABSORBANCE"/>
      <xsd:enumeration value="MILLIAMPS"/>
      <xsd:enumeration value="MILLIGRAMS"/>
      <xsd:enumeration value="MILLILITERS"/>
      <xsd:enumeration value="MILLIMETERS"/>
      <xsd:enumeration value="MILLIMOLAR"/>
      <xsd:enumeration value="MILLISECONDS"/>
      <xsd:enumeration value="MILLIVOLTS"/>
      <xsd:enumeration value="MILLIWATTS"/>
      <xsd:enumeration value="MINUTES"/>
      <xsd:enumeration value="MOLAR"/>
      <xsd:enumeration value="MOLES"/>
      <xsd:enumeration value="NANOGRAMS"/>
      <xsd:enumeration value="NANOMETERS"/>
      <xsd:enumeration value="NANOSECONDS"/>
      <xsd:enumeration value="PPB"/>
      <xsd:enumeration value="PPM"/>
      <xsd:enumeration value="PPT"/>
      <xsd:enumeration value="RADIANS"/>
      <xsd:enumeration value="RAMANSHIFT"/>
      <xsd:enumeration value="REFLECTANCE"/>
      <xsd:enumeration value="SECONDS"/>
      <xsd:enumeration value="TRANSMISSIONPERCENT"/>
      <xsd:enumeration value="TRANSMITTANCE"/>
```

```
         <xsd:enumeration value="UNKNOWN"/>
         <xsd:enumeration value="VOLTS"/>
         <xsd:enumeration value="WATTS"/>
         <xsd:enumeration value="WAVENUMBER"/>
         <xsd:enumeration value="YEARS"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="valueorder">
      <xsd:restriction base="xsd:NMTOKEN">
         <xsd:enumeration value="EVEN"/>
         <xsd:enumeration value="ORDERED"/>
         <xsd:enumeration value="UNSPECIFIED"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```