LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type	**LIGO-T990023-01 - E**	3/3/1999

LDAS Lightweight Data
format specification

Roy Williams

Distribution of this document:

LIGO LDAS Group

This is an internal working document
of the LIGO Project.

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (818) 395-2129
Fax (818) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

LIGO_LW: the LDAS Lightweight Format

Roy Williams

California Institute of Technology
LIGO Data Analysis Group
6/23/99

I. Introduction

1. XSIL is an XML-based language for describing heterogeneous collections of scientific data and metadata. The LIGO_LW language is based on the XSIL foundation, and uses the same software. More information on XML can be found at:
   a) The World Wide Web Consortium standardization effort for XML: **http://www.w3c.org/XML**
   b) Seybold publishing XML site: **http://www.xml.com/**
   c) Initial paper on XSIL: **http://www.cacr.caltech.edu/ligo/XSIL/xsil.pdf**
2. A LIGO_LW file is text-based, designed to be flexible and small rather than efficient and large. Thus we expect a parser to read the entire file only once, creating structure in memory (or virtual memory); we do not expect the software to deal with gigabyte-sized LIGO_LW files.
3. While data can be contained locally in the LIGO_LW file, there is a rich structure for defining remote links to binary data objects, allowing very high performance I/O transactions, perhaps parallel.
4. Uses for LIGO_LW files:
   a) As a flexible and general transport format between disparate applications in a distributed archiving and computing system; a text-based object serialization that can be handled by common tools, or
   b) As a documentation mechanism for collections of data resulting from experiments or simulations; with all the parameters, structure, filenames and other information needed to keep a complete scientific record.
   c) As an "ultra-light" data format: a user can, if he wants, simply read the markup, then delete all except the actual data, or all except for the filenames where the data may be found.
5. Software support for LIGO_LW will be:
   a) An API to allow object-model access to the data contained and referenced by the LIGO_LW file.
   b) Tools built with the API for manipulating and transforming LIGO_LW files.
   c) Off-the-shelf XML software for creating and editing.
   d) Style sheets to provide browsing, printing and presentation of LIGO_LW files.

II. XML Basics

A. Syntax

1. An XML file is a hierarchical structure of elements that may contain other elements. An *element* generally consists of a *start tag*, a *body*, and an *end tag*, for example: **<Fruit>Banana</Fruit>**, where start and end tags are distinguished by the presence of a slash.
   a) An element may be ***empty***, meaning that there is only a single tag, with no body, for

example **<EmptyElement/>**; note the position of the slash.

b) Elements may contain ***attributes***, for example: **<Fruit color="yellow"**>. There can only be one instance of a given attribute name in any tag.

2. XML is case-sensitive, so that **<Apple>**, **<apple>** and **<APPLE>** are all different tags.
3. Comments in XML are delimited like this:
   <!-- This is a comment -->
4. An XML document can be handled by both human and computer. If a human sees the document, then the XML is *presented*, usually by means of a style language file, or through a filter. If a computer sees the document, there is an API to control a parser of the document.

B. Object Model of XML

1. When a XML file is read by an application program through the LIGO_LW API, the hierarchical structure of the file is parsed to a hierarchical *base object*, which is then made available to the application. The LIGO_LW software layer then extracts from the base object the first **<XSIL>** element, which is returned to the application. Thus LIGO_LW can be mixed in a straightforward way with other kinds of XML or HTML, such as Math Markup Language, Chemistry Markup, or other XML languages.
2. As well as the parser, the LIGO_LW API provides a rich set of methods to extract objects of given element-type, objects which have given attributes, given Name or Type, and so on.
3. Information can then be extracted from objects. For example once a Param object of given name has been found, the string which is its value can then be extracted. In this way we have a dictionary of name-value pairs. Similarly the rows and colums of a Table, or the start and end time of an IGWDFrame object are available to the application.

III. Beginning the LIGO_LW file

1. All XML files, which includes all LIGO_LW files, have the following as their first line:
   <?xml version="1.0"?>
2. The second line of an XML file may make a reference to an externally defined Document Type Definition (DTD), which defines the syntax of LIGO_LW files. Thus every LIGO_LW file has the following as its second line:
   <!DOCTYPE XSIL SYSTEM "http://www.cacr.caltech.edu/ligo/XSIL/LIGO_LW.dtd">
   This file is presented in section VI.
3. There may follow other XML elements, but the LIGO_LW parser will ignore these until it finds a XSIL element. Only the first LIGO_LW element is then considered, so there should never be more than one in a file. Thus, in general, the third line of a pure LIGO_LW file is:
   <XSIL>
   and the last line of the file closes this element:
   </XSIL>

IV. Object Name and Type

1. Any of the LIGO_LW elements may have a Name attribute, for example:
   <Table Name="Bake_Out_Temperature">
   While the names may be useful in presentation of the LIGO_LW, the major intended use is in navigating the object model, through methods that find an object of a given name.
2. Any of the LIGO_LW elements may have a Type attribute, which may also be used for navigating the object model through queries that we might paraphrase as: *find the Table object whose Type attribute is 'Channel Descriptions'*.
3. We also expect the Type attribute to be used for extending the basic LIGO_LW types in a

very simple and straightforward way to create so-called Extension Objects (see section V.B.).

A. The Container Object

1. **XSIL**
   In the LIGO_LW language, there is a generic **<XSIL>** element which is a container for other elements, including other **<XSIL>** elements, thus inducing a hierarchy. Each of these may have a **Name** attribute, to provide hierarchical naming that is visible from the API. When the LIGO_LW API parses the file, it returns a representation of the first XSIL element that it finds in the document. Here is a LIGO_LW fragment that consists of a container hierarchy with an array at the second level and a parameter at the third level:

```
<XSIL Name="Fruit">
    <XSIL Name="YellowFruit">
        <Array><Dim>7</Dim></Array>
        <XSIL Name="Banana">
            <Param Name="Inductance">1.34</Param>
        </XSIL>
    </XSIL>
</XSIL>
```

2. The XSIL object is the only one that can contain other XSIL objects. All the others (below) can only contain only:
   a) the elements explicitly defined in their description, or
   b) a Stream object from which the data comes that fills the object structure.

B. Elementary Objects

1. **Comment**
   A comment object in LIGO_LW is not meant to be interpreted or parsed. It appears only in the presentation of the document. For example:

```
<Comment>The data that follows is probably wrong</Comment>
```

2. **Param**
   A parameter in LIGO_LW is an association between a name and a value. In addition, it may have a Unit attribute. For example:

```
<Param Name="Fruit_Mass" Unit="kg">0.387</Param>
```

   The meaning here is "Fruit_mass = 0.387 kg", which is the kind of thing usually found in "parameter files" or "header files" in scientific computing. If the element is empty, then the value is read from the corresponding stream. Note that the value of the parameter is not typed as float, int, etc, but rather it is string-valued.

3. **Time**
   In the LIGO observatory, as with many other experiments in physical science, it is critical that timing information be not only accurate, but also easy to understand. The **<Time>** element in LIGO_LW can represent either "natural" time (ISO-8601 standard, YYYY-MM-DD HH:MM:SS.mmmuuunnn), or GPS time, or "Unix time" (seconds since 1/1/1970). The different formats are differentiated by the **Type** attribute in the tag:

```
<Time Type="ISO-8601">1998-11-08 17:40:00.032</Time>
<Time Type="GPS">594582000.032</Time>
<Time Type="Unix">910546800.032</Time>
```

   The default for the type is ISO-8601.

4. **Table**
   A table is an unordered set of *records*, each of the same format, where a record is an ordered list of values. The contents of a record are defined by column headings, each of which may have a unit and a type. This definition of a table should be thought of as similar to the table

object that is found in a relational database; we should point out that this is *not* the complex and exotic typographical beast of TeX or HTML.

The only tag specific to the Table object is **<Column>**, which specifies the name, type, and possibly units associated with one of the columns of the table. It can be thought of as the heading of a column in a table. Shown below is an example table that includes a stream from which the table can be populated.

```
<Table>
    <Column Name="ChannelName" Type="string"/>
    <Column Name="Site"      Type="float"   Unit="meter"/>
    <Column Name="Clock"     Type="float"   Unit="hour"/>
    <Column Name="Description" Type="string"/>
    <Stream Delimiter=",">
        "History Channel", 405.0, 3.7, "Another Channel"
        "Math Channel", 307.0, 2.1, "Channel about Math"
    </Stream>
</Table>
```

5. **Array**

An array is a collection of numbers (or other primitive type) referenced by subscripts, which is a list of integers whose maximum values are given by the list of dimensions of the array. This definition is very close conceptually to a Fortran or C array, with the **Type** attribute of the **<Array>** tag specifying which primitive type is contained in the array (float, int, etc.). The Dim element specifies the dimensions of the array, but it does not specify the subscript ranges. For a dimension of 5, a Fortran binding of the API would label subscripts from 1 to 5, but a Java or C++ binding would have subscripts from 0 to 4.

As with other LIGO_LW objects, the Array tag may have **Name** and **Type** attributes. The Type attribute is interpreted as the data-type for the data in the aray. The only element specific to this class is **<Dim>**, which may have a Name attribute to specify the name of that dimension of the data for presentation purposes. For example:

```
<Array Type="int">
    <Dim Name="X-axis">5</Dim>
    <Dim Name="Y-axis">3</Dim>
</Array>
```

which specifies a 5x3 array of integers, with the last dimension changing fastest. The presumption is that 15 integers may be read from the Stream associated with this Array.

6. **IGWDFrame**

This element defines the metadata associated with an IGWDFrame object, which is the principle data format adopted by the LIGO and VIRGO gravitational wave obeservatories. While it is possible in principle to represent all of the Frame data in LIGO_LW, this would not seem effective given the efficiency and quality of the existing Frame format and its associated software. Here we intend to represent metadata, together with links (remote streams) to the Frame file itself.

The IGWDFrame element consists of a header section, a detector definition, a history section, and a number of AdcData sections.

```
<Header Name="new">
    <run>1</run>
    <frameNumber>433383</frameNumber>
    <Time Unit="GPS">572397599.942967000</Time>
    <leapS>0</leapS>
    <dt>1</dt>
</Header>

<Detector Name="real">
    <longitude>0.0.0</longitude>
```

```
    <latitude>0.0.0</latitude>
    <elevation>6.3714e+06</elevation>
    <azimuth>0.0</azimuth>
    <armLength>0</armLength>
</Detector>

<History>
    This section contains the history records
</History>

<AdcData Name="ASC_point_SEGs">
    <status>0</status>
    <rate>16379.6</rate>
    <nData>16</nData>
    <nBits>16</nBits>
    <offset>0.0</offset>
    <type>1</type>
</AdcData>
```

7. **Event**
   This is TBD.

C. Extension Objects

As discussed above, we can easily extend LIGO_LW in an informal, perhaps personal way, simply by creating a collection object with "well-known" parameter names. In this case, we expect the application that is reading the file to understand the special significance of the word **Time Series**, and to know the names of the parameters that it expects to find. Both time-series and frequency series are defined below.

1. **Time Series**
   The following example illustrates the collection of parameter names that would be appropriate for a time-series object. This set of names matches closely those chosen for the basic signal processing toolkit. The dimension of the array should be a multiple of the numberSamples parameter.

```
<XSIL Type="Time Series" Name="My Time Series">
    <Comment>A sample time series</Comment>
    <Param Name="startTime">.....</Param>
    <Param Name="status">.....</Param>
    <Param Name="duration">.....</Param>
    <Param Name="numberSamples">.....</Param>
    <Param Name="sampleRate">.....</Param>
    <Param Name="heterodyneShift">.....</Param>
    <Param Name="scaleFactor">.....</Param>
    <Param Name="amplitudeOffset">.....</Param>
    <Array><Dim>....</Dim></Array>
</XSIL>
```

2. **Frequency Series**
   The following example illustrates the collection of parameter names that would be appropriate for a frequency-series object. This set of names matches closely those chosen for the basic signal processing toolkit. The dimension of the array should be a multiple of the numberSamples parameter.

```
<XSIL Type="Frequency Series" Name="My Frequency Series">
    <Comment>A sample frequency series</Comment>
    <Param Name="packingType">.....</Param>
    <Param Name="status">.....</Param>
    <Param Name="numberSamples">.....</Param>
    <Param Name="minFrequency">.....</Param>
    <Param Name="bandwidth">.....</Param>
```

```
    <Param Name="sampleRate">.....</Param>
    <Param Name="decimationFactor">.....</Param>
    <Param Name="scaleFactor">.....</Param>
    <Param Name="amplitudeOffset">.....</Param>
    <Array><Dim>....</Dim></Array>
</XSIL>
```

V. Streams

A. Connecting Streams and LIGO_LW Objects

1. Some of the objects from the previous section are self-contained, for example Param and Comment. Others (XSIL, Table, Array, IGWDFrame) define the structure of a data object, without necessarily making available the data itself. The Stream object provides the input that allows these structures to be filled.
2. A LIGO_LW document may define many streams and many LIGO_LW objects (Table, Array etc.). In general a stream may contain the data for many different objects. We use the collection mechanism induced by the XSIL tag to provide this association.
3. For a stream to be available to a LIGO_LW object, either:
   a) The object contains the stream explicitly between its start and end tags, or
   b) the collection in which the object resides must contain exactly one Stream or Stream-Mirror. The objects in the collection will then be read sequentially from the Stream, unless the object has an explicit stream.

B. Stream Type: Local and Remote

1. The stream may have Type **Local** — the data is contained in the LIGO_LW file itself), or it may be **Remote** (i.e. not Local). Here is an example of a local stream:

```
<Stream Delimiter=",">
    4.76,5.77,8.99,3.44,2.11,0.93
</Stream>
```

   This is the default type for a stream.

2. If the stream has type **Remote**, then text in this element is interpreted as a URL-type locator for the data that this stream represents. A local filename would be preceded by file:// also the ftp:// and http:// protocols are supported. There may also be unregistered, idiosyncratic ways to specify the data location, such as tape://12345/657009. Here is an example of a remote stream, where again the data is text, delimited by commas:

```
<Stream Type="Remote" Encoding="Bigendian">
    file://usr/local/fred/datafile1.dat
</Stream>
```

C. Mirrored Streams

1. There is a special object used to contain multiple, equivalent streams, called a StreamMirror. When multiple streams are collected like this, there is an assumption that exactly one of the child streams is chosen (perhaps arbitrarily) from which to get the data.

```
<StreamMirror>
    <Stream Type="Remote" >
        file://myfile.dat
    </Stream>
    <Stream Type="Remote" Timeout="600">
        tape://54983
    </Stream>
</StreamMirror>
```

D. Stream Content

1. The LIGO_LW data stream may have the Content attribute set to “Typed”, meaning that it consists of a sequence of primitive data types, where these types are a subset of the Java primitive types, listed in the following (with the corresponding number of bits):
   - **boolean (1)**
   - **byte (8)**
   - **double (64)**
   - **float (32)**
   - **int (32)**
   - **long (64)**
   - **short (16)**

   a) An additional type that is allowed here is the String type, whose binary representation is a number of character bytes, followed by a null (zero) byte, as in the C-language.

   b) Another additional type is doubleComplex, meaning a pair of 64-bit doubles that represent the real and imaginary parts of a complex number.

   Typed content is the default for a stream.
2. The LIGO_LW data stream may consist of a stream of bytes that is not interpreted by the LIGO_LW/XML library. In this case the Content attribute is set to Raw:
   <Stream Content=”Raw”>
3. The LIGO_LW data stream may consist of a MIME-typed data stream, delivering to the application the MIME type, the content length, the content ID, as well as the byte stream itself. In this case the Content attribute is set to MIME:
   <Stream Content=”MIME”>

E. Stream Encoding

1. The **Encoding** attribute specifies how the data in a Typed stream is encoded. It is a comma-separated list chosen from the values:
   - **Text, Binary, uuencode, base64, BigEndian, LittleEndian, Delimiter**
2. The Text attribute is assumed by default for Local streams, the Binary attribute for Remote streams.
3. When the stream is uuencode or Base64, it is decoded by the LIGO_LW library. When the stream specifies an Endian order, it is converted to the Endian order appropriate to the current machine before being delivered to the application code.
4. The other attribute define in this section is **Delimiter**: It is only relevant in the case of a Text stream. The characters in the attribute are appended to a delimiter string that already contains newline and tab.

F. Accessing the Stream Data

1. The following attributes concern other features of the stream.
2. **Actuate**: the value of this attribute determines when the application open the stream and begin to fetch data.

   a) **Delay**: Data is not read from the stream until the latest possible moment. Caching is discouraged.

   b) **Immediate**: Data is prefetched aggressively, to minimize any possible access latency. If possible, the entire content of the stream is read into memory.

   c) Cache policy (TBD)
3. **Access**: Permission attributes for the stream, maybe username/password pairs.
4. **Timeout**: The time in seconds to wait for data to be begin to flow from the link before

signalling failure.

5. **Presentation**: Information about how the link looks when written on a page for human consumption.

VI. The DTD for LIGO_LW

1. An XML file may be associated with a Document Type Definition (DTD) which defines the allowed tag names in the document, and how these fit together: which elements may contain which other elements, and how many of each element there may be.

```
<!ELEMENT XSIL ((XSIL|Comment|Param|Time|Table|Array|IGWDFrame|Event|Stream)*)>
<!ATTLIST XSIL Name CDATA "" Type CDATA "">

<!ELEMENT Comment (#PCDATA)>

<!ELEMENT Param (#PCDATA)>
<!ATTLIST Param Name CDATA "" Type CDATA "" Unit    CDATA "" >

<!ELEMENT Time (#PCDATA)>
<!ATTLIST Time Name CDATA "" Type CDATA "">

<!ELEMENT Array (Dim*, Stream?)>
<!ATTLIST Array Name CDATA "" Type CDATA "">
<!ELEMENT Dim (#PCDATA)>
<!ATTLIST Dim Name CDATA "">

<!ELEMENT Table (Column*, Stream?)>
<!ATTLIST Table Name CDATA "" Type CDATA "">
<!ELEMENT Column EMPTY>
<!ATTLIST Column Name CDATA "" Type CDATA "" Unit CDATA "">

<!ELEMENT IGWDFrame (Header, Detector, History, AdcData*, Stream?)>
<!ELEMENT Header (run, frameNumber, Time, leapS, dt)>
<!ATTLIST Header Name CDATA "">
<!ELEMENT Detector (longitude, latitude, elevation, azimuth, armLength)>
<!ATTLIST Detector Name CDATA "">
<!ELEMENT longitude (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>
<!ELEMENT elevation (#PCDATA)>
<!ELEMENT azimuth (#PCDATA)>
<!ELEMENT armLength (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT AdcData (status,rate,nData,nBits,offset,type)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT rate (#PCDATA)>
<!ELEMENT nData (#PCDATA)>
<!ELEMENT nBits (#PCDATA)>
<!ELEMENT offset (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ATTLIST AdcData Name CDATA "">

<!ELEMENT Stream(#PCDATA)>
<!ATTLIST Stream Name CDATA "" Type CDATA "" >
<!ATTLIST Stream Content CDATA ""Encoding CDATA "" Delimiter CDATA "">
<!ATTLIST Stream Actuate CDATA "" Access CDATA "">
<!ATTLIST StreamTimeout CDATA "" Presentation CDATA "">
<!ATTLIST Stream    Immediate CDATA "">

<!ELEMENT StreamMirror (Stream)+>
```