XidML 2 Handbook

An open standard for an e**X**tensible **I**nstrumentation **D**ata exchange **M**ark-up **L**anguage

26 January 2006

www.XidML.org

CONTACT DETAILS

E-mail: mail@xidml.org

Website: www.xidml.org

Contents

About this book

This handbook assumes you are already familiar with XML and the advantages of using it as an open standard. For more information visit www.XML.com

XidML 2 (pronounced Sid-ML two) is an XML method of storing information on how data is acquired, processed and packaged for transmission, storage or reproduction.

In particular, XidML 2 is designed to store and exchange information between multiple vendors and user-groups about complex instrumentation gathering thousands of parameters.

Because XML schemas can be self-documenting, this handbook is not intended to be a complete description of XidML 2, rather it is hoped that it will enthuse you and explain some of the schema decisions made.

Using the XidML 2 Handbook

A “What’s new?” section at the beginning of each chapter details differences between XidML 2.2 and XidML 2.1

The symbol NEW appears on the left-hand-side of the page where content has been added to the XidML 2 Handbook since its last issue.

Suggested reading paths

The goal of the first two chapters is introduce the goals and concepts behind the XidML 2 schema	Chapter 1 - Introducing XidML 2 Chapter 2 - Introducing XidML 2 schema
Outlines key elements of the Documentation Schema. **The documentation schema is less complex than the other schema and so serves as a good introduction to XidML 2 schema.**	Chapter 3 - Creating documentation
Explains the schema for parameters.	Chapter 4 - Defining parameters
Explains the schema for packages.	Chapter 5 - Creating packages and DataLinks
Explains the schema for algorithms.	Chapter 6 - Creating algorithms
Explains the schema for instrumentation.	Chapter 7 - Defining instrumentation settings
Shows a complete example.	Chapter 8 - Looking at a complete example
These appendices section comprises: **A-1 XidML 2 synopsis** **A-2 Schemas available from xidml.org**	After reading the XidML 2 Handbook, the only references required is A-2

1 - Introducing XidML 2

Overview

XidML 2 is a vendor-neutral XML-based data storage architecture developed for large and complex instrumentation systems.

This chapter introduces the motivation behind XidML 2, the goals, the xidml.org website and future directions of XidML 2. It also includes some suggested reading on other related standards.

1.1. Why XidML 2?

1.1.0.1. Imagine a modern aircraft test scenario where there are thousands of signals being gathered via instrumentation from tens of different vendors all around the aircraft.

- Some of the temperature parameters are sampled at rates as low as 1 or 2Hz, and some of the structural integrity parameters are being sampled at tens of kHz.
- Some data is packaged for transmission on fly-by-wire avionics busses, some for large storage devices and some for transmission to the ground.
- Some data is analyzed in real-time by pilots and some data is analyzed hours, days or years later by engineers and scientists.

1.1.0.2. The purpose of XidML 2 is to document for each vendor and user group, where each parameter comes from, how the instrumentation was configured, and how the data was packaged and processed at each stage.

1.1.0.3. XidML 2 contains Documentation elements where details on "why", "when", and "who" can be stored along with descriptions.

1.1.0.4. It also contains Parameter elements that list the type of parameter (e.g. temperature) and the range it measures (e.g. -100 to 1000°C) and where it originates (e.g. thermocouple sensor #23 on engine) and how it is packaged for transmission to ground or archival.

1.1.0.5. It contains information on any Algorithms used to process the data (e.g. the thermocouple linearization table that converted voltage to temperature) and information on the Instrumentation settings required to configure the instrumentation.

1.2. Design goals of XidML 2

1.2.0.1. When defining the XidML 2 schema there were many choices of how to perform various tasks. The following are the key goals or rules that informed those choices.

1.2.0.2. XidML 2 must:

- be vendor-neutral and easy to learn
- be future-proof and evolve with new standards and technologies
- facilitate instrumentation-data archival for analysis and tracking many years later
- document large numbers of different devices
- accommodate very large parameter lists and complex sampling strategies

1.3. Applying these goals

1.3.0.1. XML was an obvious choice when growth (eXtensible) was a criterion.

1.3.0.2. Breaking XidML 2 into multiple schemas is necessary for multiple vendors. Each setup can share the same parameter and packaging schema. If there are Instrumentation settings unique to one vendor then only a limited number of sub-elements of the Instrumentation schema are affected.

1.3.0.3. The need to archive instrumentation-data alongside acquired data means that the file must be compact. Therefore a frequently used linearization table of 64kbytes should not be repeated for each instance of the sensor.

1.3.0.4. The need to archive also means the XidML 2 file must be complete. This has implications for the use of library elements in files. For example, if the packet structure points to a library of structures, this library should be archived with the data, especially as the data may not be analyzed until a few years after it has been gathered.

- Many instruments acquiring 1000s of parameters at many rates, to be stored or transmitted in different packet types, led to the concept of an Acquisition Cycle. This concept is further explained in Section Key XidML 2 elements.
- An attribute can be used if it helps identifying, sorting or finding instrumentation-data but it cannot be used to define instrumentation-data

1.3.0.5. Setting an Acquisition Cycle (please see Section 2.1.3 AcquisitionCycle for more details) allows you to co-ordinate or synchronize the operation of an avionics bus sending messages in 60Hz cycles, with a radio transmitter to ground sending 128 packets per second and a solid state storage device storing multiple parameters at 20,000 samples per second.

1.4. Using xidml.org

1.4.0.1. More information on XidML 2 can be found at www.xidml.org including:

- Links to relevant standards (W3C, OMG Sensor ML and so on)
- Schemas for common algorithms and packages
- Documentation e.g. this handbook

1.4.0.2. At xidml.org/documentation/designingtools.html we discuss some XidML 2 tools that could be developed. We also discuss the difference between schema validation and vendor validation. A valid and well-formed XidML 2 file can have a range of ±9V which the schema validation accepts but the vendor validation may not.

1.5. Where next for XidML 2?

1.5.0.1. Where XidML 2 goes from here depends largely on you. Each paragraph in this book is numbered to help you to reference your feedback. What follows in this section are some thoughts on where it may go.

NEW 1.5.0.2. One method of feeding back ideas id via the Feedback button on the xidml.org website.

1.5.1 Supporting libraries

1.5.1.1. XidML 2 supports auxiliary files (please see Section 2.1.2 Auxiliary files for more information) and it is assumed that all data will be archived. However we may introduce the use of library elements for popular parameter types, algorithms and packages. These libraries elements could be assumed to be archived elsewhere.

1.5.2 Compressing XidML

1.5.2.1. If instrumentation-data is to be stored or transmitted with data then the file size may have be compressible. We understand that some work has been done "binary" XML that might help.

1.5.2.2. Another advantage of a compressed XidML file is that it could be stored in the instrument or sensor, so that years later how the instrument was set up can be viewed via an open standard file.

1.5.3 Defining equipment interfaces

1.5.3.1. XidML 2 is designed for the data used in instrumentation setup and processing, i.e. the data needed to program equipment and acquire and analyze data. However, frequently asked questions include:

- Could an XML file contain connector details such as type, manufacture part number and the pin numbers for each channel to aid the installation process?
- Could an XML file contain details of an instrument's power consumption, input impedance, or accuracy to aid the equipment selection process?
- What if it also contained details of sensors, such as the resistance, linearization, and excitation required so that XidML data could be automatically defined?

1.5.3.2. These are exciting ideas, however, to help focus XidML 2 development we suggest that these interface (i/f) details be discussed separately, under another heading (perhaps XifML).

1.6. Further reading

On XML: The XML specification and general information can be found at XML www.xml.org

On OMG space domain task force: The latest submission to the OMG from NASA/ESA can be found at http://www.omg.org/space/RFP_1/RFP_1.htm

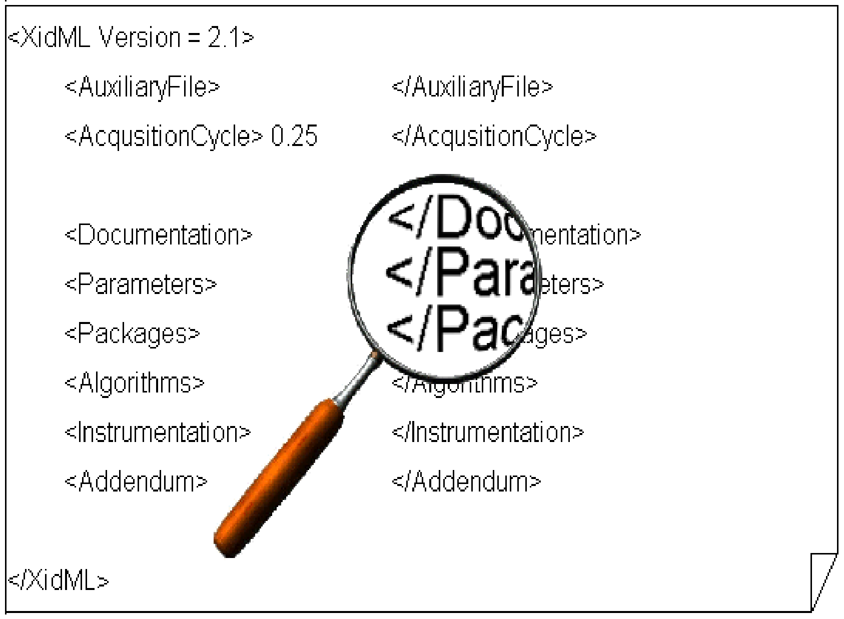
On MathML: The MathML specification can be found at www.w3.org/math More general information can be found at http://www.mathmlcentral.com/

On sensorML: The SensorML specification and general information can be found at http://www.opengeospatial.org/groups/

On TMATS: For more information on TMATS please see IRIG-106 Ch. 9.

On SI units: http://www.physics.nist.gov/cuu/Units/

2 - Introducing XidML 2 schema



```
<XidML Version = 2.1>
      <AuxiliaryFile>                </AuxiliaryFile>
      <AcqusitionCycle> 0.25         </AcqusitionCycle>

      <Documentation>                </Documentation>
      <Parameters>                   </Parameters>
      <Packages>                     </Packages>
      <Algorithms>                   </Algorithms>
      <Instrumentation>              </Instrumentation>
      <Addendum>                     </Addendum>

</XidML>
```

Overview

This chapter introduces some of the key elements of the XidML 2 schema.

In particular Documentation, Parameters, Packages, Algorithms, Instrumentation, and Addendum elements are introduced here, and then discussed in more detail in later chapters. For complete schemas please see www.xidml.org

This chapter also introduces some of the conventions and ideas used throughout the schema.

2.1. Introducing key elements of the XidML 2 schema

2.1.0.1. Example 2.1 shows key element names at the "top" level of the XidML 2 schema. These are each discussed below.

```
<XidML Version = "2.1">
    <AuxiliaryFile>                                  </AuxiliaryFile>
    <AcqusitionCycle>                              </AcqusitionCycle>

    <Documentation>                                  </Documentation>
    <Parameters>                                        </Parameters>
    <Packages>                                            </Packages>
    <Algorithms>                                        </Algorithms>
    <Instrumentation>                              </Instrumentation>
    <Addendum>                                            </Addendum>



</XidML>
```

Example 2.1. *Key elements of the XidML 2 schema*

2.1.1 XidML

2.1.1.1. This must be the "root element" for each file and has the version as an attribute.

2.1.2 Auxiliary files

2.1.2.1. Because "meta-data tracing", i.e. tracing the source of information, is critical to the XidML 2 philosophy; the use of libraries is not supported.

2.1.2.2. However XidML 2 does allow for instrumentation-data to be stored in multiple XML files providing the following rules are obeyed:

- AuxiliaryFile elements must be named at the top of your main file.
- Each auxiliary file must agree with the XidML 2 schema rules.

NEW 2.1.2.3. In particular it is recommended that XidML 2 parsers will simply "copy and paste" auxiliary files, pointed to by the URI, into a single file before running. In more detail, a XidML parser will take an entire section in an auxiliary file (e.g. the Parameters section) and place it at the end of the equivalent section in main XidML file. Auxiliary files will be “copied and pasted” in the order that they appear in the Auxiliary files section.

```
<AuxiliaryFiles>
    <AuxiliaryFile ArchiveThisTime="True">MyPackages.xml</AuxiliaryFile>
</AuxiliaryFiles>
```

2.1.3 AcquisitionCycle

2.1.3.1. The AcquistionCycle is analogous to the time taken to transmit a major-frame in PCM systems. In XidML 2 this element is mandatory with a value in seconds. The reasoning behind the use of this concept is discussed in Section 2.2.5 Using defaults.

```
<AcquisitionCycle>125e-3</AcquisitionCycle>
```

2.1.3.2. XidML 2 strongly recommends that this element be in the main file, however it may also be repeated in auxiliary files if the value is the same.

2.1.4 Documentation

2.1.4.1. This element is used to record who did what, when, why, and how. Documentation elements can exist for the files and as children of each parameter, package, algorithm, and instrument.

2.1.4.2. The who and when are recorded as author and date elements. The why and how can be described with long and short descriptions.

2.1.4.3. This element is discussed in Chapter 3 - Creating documentation.

2.1.5 Parameters

2.1.5.1. This element is used to define what a parameter measures, what format it is in, where it comes from, and all the places it is going to.

2.1.5.2. The parameters element is discussed in *Chapter 4 - Defining parameters*.

2.1.6 DataLinks

NEW

2.1.6.1. This element is used to define properties common to all packages such as bit-rate (on an RS-422 bus), bits per word (on a CAIS bus), or TTL v.s RS-422 (on a PCM link). The DataLink element is discussed in *Chapter 5 - Creating packages and DataLinks.*

2.1.7 Packages

2.1.7.1. This element is used to define how data is transmitted on busses, networks, or links of any kind. Packages can also be used to agree how data is stored or archived.

2.1.7.2. The Package element is discussed in Chapter 5 - Creating packages and DataLinks.

2.1.8 Algorithms

2.1.8.1. Algorithms are used to define linearization tables, conversion formula, processing, embedded parameters, and alarms.

2.1.8.2. The Algorithms element is described in Chapter 6 - Creating algorithms.

2.1.9 Instrumentation

2.1.9.1. This element is used to define settings for each instrument (real or virtual) such as excitation voltage for a strain gage, or misses-to-loss for a PCM decoder.

2.1.9.2. While XidML 2 has generic schemas for some common instruments this is the one element where vendors and user-groups may have to develop their own schemas.

2.1.9.3. The Instrumentation element is described in Chapter 7 - Defining instrumentation settings

2.1.10 Addendum

2.1.10.1. This element allows a vendor or user-group to store in a "free-style" manner, information that is only needed by that vendor or user-group.

```
<Addendum>
    <FTIRUs: HumidityOnTestDay>95<FTIRUs: HumidityOnTestDay>
</Addendum>
```

2.1.10.2. This data can be encrypted if need be, as it will only be validated and parsed by the schema or parser from one vendor or user-group. In this example the schema used by the flight test instrumentation group is indicated by FTIRUs.

2.2. Themes and conventions throughout XidML 2

2.2.1 Naming conventions

2.2.1.1. Element names such as MaximumRange, DataType or WordSize are used rather than short, vague, or ambiguous names such as Max, Type or Size.

2.2.1.2. In example schemas we recommend using element identifiers starting with 'My' such as MyExamplePackage.

2.2.1.3. In sample schemas for XidML 2 where vendor names are required for illustration, fictitious names such as MetersRUs or SensorsRUs are used to reinforce the vendor neutrality of XidML 2.

2.2.2 Base units and parameter types

2.2.2.1. In Chapter 4 - Defining parameters, we will see the introduction of a rigorous use of units - all referenced to SI base units. Also the choice of data formats is equally rigorous and "up-front", this ensures greater flexibility in the design of algorithms and packages and more rigor on instrumentation outputs.

2.2.3 Use of XML attributes

2.2.3.1. XidML 2 follows the convention that attributes should only be used to sort, find and link data and should not contain important data. For more discussion on this convention, please see:
http://xmlfiles.com/xml/xml_attributes.asp
http://www.w3schools.com/xml/xml_attributes.asp
http://justicexml.gtri.gatech.edu/elements_versus_attributes.html
http://www.businesscentricmethodology.com/CMS/documents/guidance/DFAS.XMLBestPractices.2002-12-12.ppt

2.2.3.2. In particular attributes are only used in the following three situations:

- A Name attribute is used to uniquely identify the major components or the schema. In particular, Parameters, Packages, Algorithms and Instrumentation will be uniquely identified using a Name attribute
- Array pointers. If an array of items is identified by a unique number, an attribute can be used. For example, channels on an eight-channel data acquisition module would have Index values 0 to 7 while busses on an a eight-bus ARINC-420 bus monitor would be have Number attribute values or 1 to 8.
- The root element of all XidML 2 files <XidML> must have a version attribute.
- Attributes can be used as instructions for processing tools. For example, the Auxiliaryfile element

```
<Parameter Name="MyTemp">
```

has an "ArchiveThisTime" attribute to indicate that the auxiliary file should be archived and in Parameter definition the "BalanceSettings" and "ShuntSettings" elements respectively have "BalanceThisTime"and "ShuntThisTime" attributes.

2.2.3.3. No other attributes have any meaning in XidML 2. However XidML 2 allows user-defined attributes.

2.2.4 Defining instrumentation-data in a single location

2.2.4.1. Data defined in multiple locations causes conflict and ambiguity. Instrumentation-data must be defined in a single location for settings.

2.2.4.2. How do we resolve a conflict if the instrumentation settings specify 30 samples per second; if for example, four samples of a parameter are transmitted in a packet sent 10 times per second? This is resolved by not having the sample rate as a setting for instrumentation.

2.2.4.3. Where we specify a range of -8V to +8V for a parameter and the vendor requests a gain and offset setting for the instrumentation, a gain and offset setting is meaningless unless we know the input range of the Analog/Digital. In XidML 2, we resolve this conflict by not allowing the vendor to specify gain and offset. The vendor must program the instrumentation with the appropriate gain and offset to meet the range or else warn the user at compilation that the range is not available.

2.2.5 Using defaults

2.2.5.1. Some elements which need assigned values have defaults to aid quick development of schema. For example DataSize defaults to 16 bits if the element is empty or not present.

2.2.5.2. A-1 XidML 2 synopsis, lists all the defaults for reserved XidML 2 elements.

2.2.5.3. Some defaults are recommendations for parser tools rather than defaults for schema validators.

2.2.6 Understanding the acquisition cycle

2.2.6.1. Background

- An analysis engineer is told that one parameter is sampled at 1000 samples per second and another parameter at 8 samples per second. How can these be graphed on the same plot? In other words how do we time-correlate parameters at different rates?
- How do we correlate samples taken on multiple airframes (located far apart) to ±1μs?
- How do we synchronize the operation of an avionics bus sending messages in 60Hz cycles with a radio transmitter to ground sending 128 packets per second and a solid state storage device storing multiple parameters at 20,000 samples per second?

2.2.6.2. If the sampling strategy changes because you are less interested in parameters from the undercarriage at 20,000 feet then you were during takeoff, when does all the equipment switch strategies?

2.2.6.3. In XidML 2 these issues are resolved using the concept of an acquisition cycle, which can be loosely associated with a major-frame. Let's look at the two examples above

- Sample and storage rates of 8, 1000 and 20,000 samples per second
- Packets transmitted at 60 and 128 packets per second

2.2.6.4. All these operations can be synchronized by insisting that all signals were sampled at the same time at the start of a second, and this time was also used to synchronize building of packets for transmission and storage.

2.2.6.5. XidML 2 does not support multiple acquisition cycles as this greatly simplifies the rules for format switching.

2.2.6.6. Sampling and transmitting data with respect to acquisition cycle significantly reduces the number of time tags associated with each packet and sample - one time tag tags all.

2.2.6.7. To reduce the time to recover from errors or brownouts or to switch between sampling strategies a shorter acquisition cycle is often a good idea. For the example above, if 250ms (4Hz) is chosen then the maximum delay to a format switch is 250ms, and the worst-case delay to receipt of a good major-frame after a brownout is also reduced.

2.2.6.8. XidML 2 also strongly recommends that this value be an integer number of micro-seconds as this greatly simplifies synchronization with respect to certain IRIG time codes, GPS 1Mpps outputs, CAIS command busses, etc.

2.2.7 Multiple sampling strategies and IBIT

2.2.7.1. In flight-test where the amount of radio bandwidth is limited the concept of format switching is often used. Parameters are sampled at different rates depending on the test phase (for example take-off, cruising or landing).

2.2.7.2. Remember that in XidML 2, sampling rates are functions of the packages containing the samples. So different formats or strategies are handled via different package contents. To greatly simplify package definition for different formats, XidML 2 insists the acquisition cycle be the same for each format.

2.2.7.3. One of the design goals for XidML 2 is that it would be a complete file for systems with 1000s of parameters. Often IBIT (Initiated Built In Test) procedures must be developed that support automatic checking of whether the instrumentation is present, functioning, and within tolerance. XidML 2 supports this automation by allowing datum values and tolerances to be defined as balance settings and within alarm algorithms.

2.2.8 Using a flat structure

2.2.8.1. Due to the need for simplicity with the large numbers of instruments and parameters it was decided that a "flat schema" would be easier to structure, describe, and parse. For example, when pointing to the source instrument for a parameter, it is simpler to point to an instrumentation name rather than use a path structure.

2.2.8.2. All packages are described as sub-elements directly under Packages. To allow reuse and to facilitate quick parsing of XidML 2 files each package has a unique name. This rule also applies to Parameters and Algorithms.

2.2.8.3. Documentation elements can exist as sub-elements to any of the key elements, for example you might want to describe a particular package and who created it and why and when it was last edited.

2.2.8.4. Each instrument exists as a separate child of instrumentation. In particular in a chassis, rack, or slice-of-bread system each module is a separate instrument with a unique name. Even "virtual instruments" such as display screens exist as separate instruments. Please see Chapter 7 - Defining instrumentation settings for an illustration of this.

2.2.8.5. However to facilitate grouping and sub-grouping of algorithms, parameters, packages, or instrumentation according to location, function etc. XidML 2 contains group elements for each. This is shown for parameters in Chapter 4 - Defining parameters.

2.2.9 Mapping parameters

2.2.9.1. Packages and algorithms that contain parameters can have other parameter names mapped on to them so as to encourage reuse of packages and algorithms.

2.2.9.2. For example a packet for RF transmission might have "MyP1" super-commutated in each minor-frame. In the parameter schema, the parameter "MyLeftWingTemperature5" can be mapped to "MyP1" without the package being rewritten.

```
<Parameter Name="MyLeftWingTemperature5">
    <Source>
        <Package>
            <InstrumentReference>MyRFTransmitter</InstrumentReference>
            <ParameterMap>MyP1</ParameterMap>
            <PackageReference>MyPackage</PackageReference>
        </Package>
    </Source>
</Parameter>
```

2.3. Complete example

2.3.0.1. Please see Section 8 - Looking at a complete example, for a complete XidML 2 definition for the system in Figure 2.1.

2.3.0.2. In this example a remote terminal is the source of a parameter (MyAltitudeInFt) that is read by a 1553 bus controller and then transmitted to ground where it is graphed on a real-time strip-chart.

2.3.0.3. Another parameter (MyGage1234) is also monitored and forms part of Go/NoGo algorithm that controls an LED for the pilot.

2.3.0.4. Some of the parameters, packages, algorithms, and instrumentation in this example are also discussed throughout this document (you may want to print Figure 2.1 out as a reference)

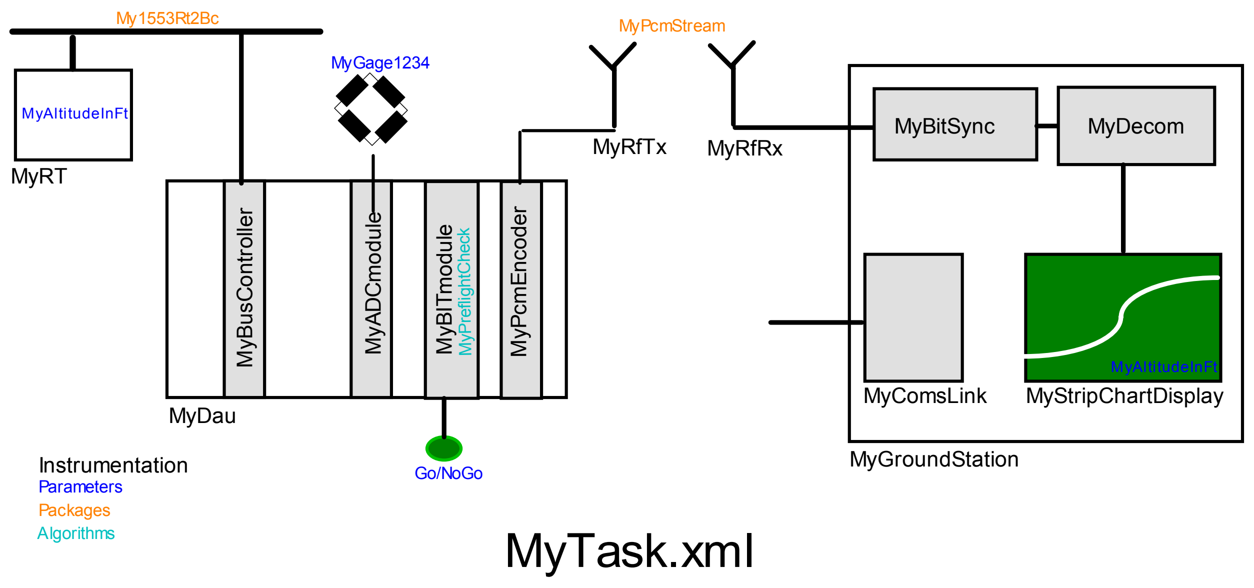
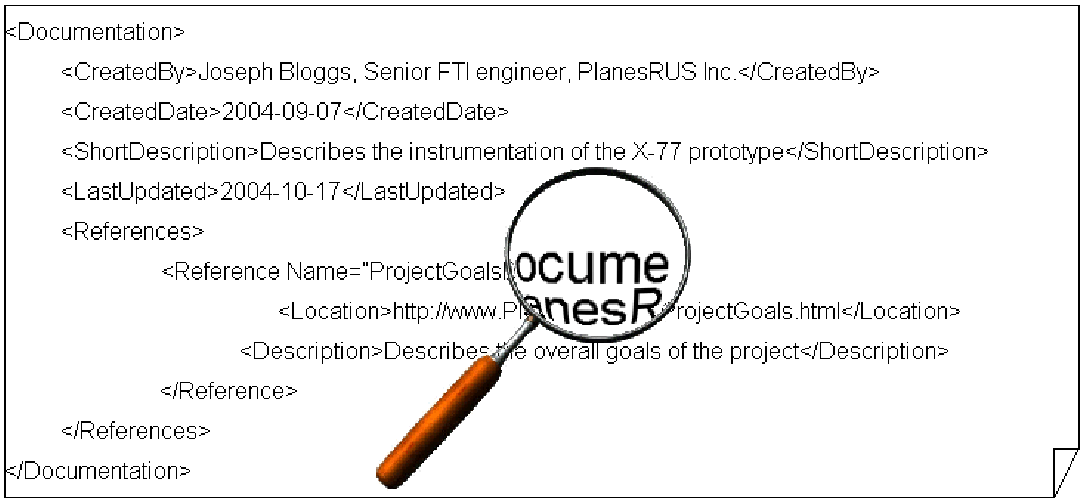


Figure 2.1: *MyTask.xml*

3 - Creating documentation



```
<Documentation>
    <CreatedBy>Joseph Bloggs, Senior FTI engineer, PlanesRUS Inc.</CreatedBy>
    <CreatedDate>2004-09-07</CreatedDate>
    <ShortDescription>Describes the instrumentation of the X-77 prototype</ShortDescription>
    <LastUpdated>2004-10-17</LastUpdated>
    <References>
        <Reference Name="ProjectGoals
            <Location>http://www.Pl  rojectGoals.html</Location>
          <Description>Describes  e overall goals of the project</Description>
        </Reference>
    </References>
</Documentation>
```

ocume
lanesR

Overview

The documentation section of the schema allows you to add comments and descriptions to your XidML 2 files. It also allows you to track when additions were made, who made the changes and why. In addition to this information, references to external documents containing other relevant information can also be included as part of the documentation section.

The documentation element can be included at many levels of the document. Its primary purpose is to aid the documentation of instance documents and to simplify document archiving and retrieval.

What's new?

The documentation section of the XidML 2.2 schema does not differ from that of the XidML 2.1 schema. However, some text in the schema examples may have been updated in this chapter since the last revision of the *XidML Handbook.*

3.1. Using the documentation element

3.1.0.1. The Documentation element has five child elements. These child elements are listed in Appendix - Key XidML 2 elements. It is recommended that document authors should provide short and long descriptions to make this section more meaningful and informative, and to facilitate the automatic retrieval of high-level information.

Documentation example

3.1.0.2. Example 3.1 shows how the documentation element may be used. The Documentation element in this example includes information about who created the document and when it was created, as well as a short description.

```
<Documentation>
    <CreatedBy>Joseph Bloggs, Senior FTI engineer, PlanesRUs Inc.</CreatedBy>
    <CreatedDate>2004-09-07</CreatedDate>
    <ShortDescription>Describes the instrumentation of the X-77 prototype</ShortDescription>
    <LongDescription>This describes the instrumentation of the X-77 prototype. The X-77 prototype is part of the
BDD project - serial number 124.99-11.</LongDescription>
    <LastUpdated>2004-10-17</LastUpdated>
    <References>
        <Reference>
            <Location>http://www.PlanesRUs.com/ProjectGoals.html</Location>
            <Alias>ProjectGoalsDocument</Alias>
            <Description>Describes the overall goals of the project</Description>
        </Reference>
        <Reference>
            <Location>http://www.PlanesRUs.com/TechnicalSpecs.html</Location>
            <Alias>TechnicalSpecifications</Alias>
            <Description>Contains technical specifications for the project</Description>
        </Reference>
        <Reference>
            <Location>http://www.PlanesRUs.com/RevisionHistory.html</Location>
            <Alias>RevisionHistory</Alias>
            <Description>Revision history for this document</Description>
        </Reference>
    </References>
</Documentation>
```

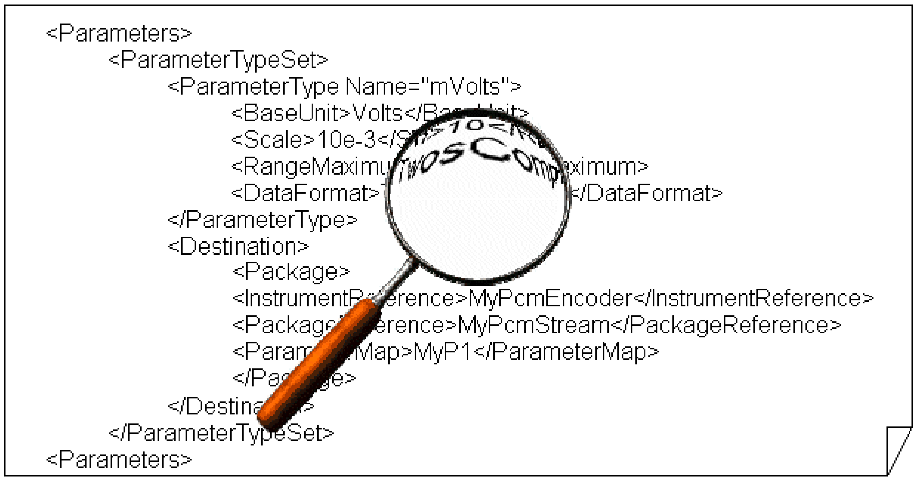
Example 3.1 *Documentation element example*

3.1.0.1. The Documentation element also references external documentation. In this case three other documents are referenced. They are documents containing information about the overall project goals (http://www.PlanesRUs.com/ProjectGoals.html), a full description for the technical specifications for the project in (http://www.PlanesRUs.com/TechnicalSpecs.html), and a revision history of the document along with the associated role of each engineer in (http://www.PlanesRUs.com/RevisionHistory.html).

3.1.0.2. Document references need not be links to web resources, but can be a reference to a file on a local machine or network. Any valid URI reference is acceptable. Also, the documents referenced in this section are not considered to be part of the document (i.e. they are not auxiliary files). It is up to the author of the XidML 2 file to verify that the referenced resources exist and they are archived if necessary.

3.1.0.3. Please see *Appendix - Key XidML 2 elements* for a formal definition of these elements.

4 - Defining parameters



```
<Parameters>
    <ParameterTypeSet>
        <ParameterType Name="mVolts">
            <BaseUnit>Volts</B[illegible]
            <Scale>10e-3</S[illegible]10<R[illegible]
            <RangeMaximu[illegible]ximum>
            <DataFormat>T[illegible]</DataFormat>
        </ParameterType>
        <Destination>
            <Package>
            <InstrumentR[illegible]erence>MyPcmEncoder</InstrumentReference>
            <Package[illegible]erence>MyPcmStream</PackageReference>
            <Param[illegible]Map>MyP1</ParameterMap>
            </Pa[illegible]e>
        </Destina[illegible]>
    </ParameterTypeSet>
<Parameters>
```

Overview

In this section we first define different parameter types, and then explain how these are used to define a particular parameter. In particular, we discuss how to name a parameter and define the units it measures, the range it measures, and the format for the data. We also discuss how to define where the parameter comes from and goes to.

This section also discusses how to define and report balance and shunt conditions.

Some new concepts are introduced in this chapter, in particular the idea of not using terms such as "raw data", "physical units", "engineering units" or "instrumentation units".

DON'T PANIC! It is only nine more pages and once you have grasped the concepts the succeeding chapters are easier. As you read this chapter it may help to remember the two guiding principles of the Parameters schema

- Define as much as possible in a way that is vendor-neutral
- Force a rigor (early on) on data-types which will pay off in our discussion on algorithms and packages later

What’s new?

The BalanceSettings and ShuntSettings elements have been removed from the ParameterType element in ParameterTypeSet.

4.1. Introducing parameter types

4.1.0.1. Over the years, many conventions have evolved with respect to raw units, instrumentation units, physical units, and engineering units. With XidML 2 we have decided to force a rigorous use of units "up-front and early". With this approach, there is no chance of confusion of, for example, miles and kilometers, even though both can be used as all algorithms and conversions etc. reference a base type.

4.1.0.2. Example 4.1 shows a parameter type (MyFahrenheit) that measures a range of -100F to +100F and is transmitted as a 12-bit 2's complement number.

```
<Parameters>
    <ParameterTypeSet>
        <ParameterType Name="MyFahrenheit">
                <BaseUnit>Celsius</BaseUnit>
                <Scale>0.55555</Scale>
                <Offset>-17.777778</Offset>
                <RangeMaximum>100</RangeMaximum>
                <DataFormat>TwosComplement</DataFormat>
                <SizeInBits>12</SizeInBits>
        </ParameterType>
    </ParameterTypeSet>
</Parameters>
```

Example 4.1 *MyFahrenheit parameter type*

4.1.0.3. For this parameter type the range maximum is actually 37.78°C (100x(5/9)-32/1.8). In particular, any value set for this parameter in terms of balance targets, tolerances, or alarm conditions will be interpreted in Fahrenheit then converted to Celsius.

4.1.0.4. This consistent reference to a base unit is powerful, as algorithms can be written independently of the parameter type chosen by the user.

4.1.0.5. Example 4.2 shows a parameter type (MyAltitudeInFt) that measures a range of 0 to 65535 feet and is transmitted as a 16-bit offset binary number.

```
<Parameters>
    <ParameterTypeSet>
        <ParameterType Name="MyAltitudeInFt">
            <BaseUnit>Meters</BaseUnit>
            <Scale>0.3048</Scale>
            <RangeMaximum>65535</RangeMaximum>
            <RangeMinimum>0</RangeMinimum>
        </ParameterType>
    </ParameterTypeSet>
</Parameters>
```

Example 4.2 *MyAltitudeinFt*

4.1.0.6. In XidML 2 RangeMinimum defaults to the negative of RangeMaximum. DataFormat defaults to Offset Binary, DataSize defaults to 16 bits, and Offset defaults to 0; so in this case they need not be specified.

4.1.0.7. XidML 2 has chosen the 63 SI units, some of which are listed below, as its base units. Remember XidML 2 is a storage and archive mechanism not a graphical user interface. In other words, You are free to enter data in whatever form you prefer, e.g. data can be entered in feet and stored in feet as long as feet is referenced with respect to meters.

4.1.0.8. XidML 2 supports all 63 SI units, some of which are listed below (the full list is available from xidml.org)

Unit	Quantity	Origin	Symbol
Meter	length	SI base unit	m
Kilogram	mass	SI base unit	kg
Second	time	SI base unit	s
Ampere	electric current	SI base unit	A
SquareMeter	area	SI derived unit	m^2
CubicMeter	volume	SI derived unit	m^3
MeterPerSecond	speed	SI derived unit	m/s
MeterPerSecondSquared	acceleration	SI derived unit	m/s^2
:			
Hertz	frequency	SI special	Hz
Watt	power, radiant flux	SI special	W
Volt	electric potential difference, electromotive force	SI special	V
Celsius	temperature	SI special	ºC
NewtonMeter	moment of force	SI special derived	N•m
RadianPerSecond	angular velocity	SI special derived	rad/s
RadianPerSecondSquared	angular acceleration	SI special derived	rad/s^2

Table 4.1 *Some of the XidML 2 base types with a SI origin*

4.1.0.9. For these base types the DataFormat can be offset binary, 2's complement, sign plus magnitude or IEEE single and double precision floating point values

4.1.0.10. The DataSize is optional. For example, a 14-bit A/D value can be transmitted as 12 bits if the MSBs are transmitted or stored as 16 bits if the two LSBs are made 0. In this case if data size was fixed we would have had to define three different parameters not necessary if everyone follows the same convention.

4.1.0.11. RangeMaximum is mandatory.

4.1.1 Non SI base types

4.1.1.1. Even with SI units as base types we are still left with the issue of unitless parameters such as %, parts per million, με, etc. Also, what are the units for an MPEG-4 stream? To help with these and other issues XidML 2 introduces the six base types listed in Table 4.2:.

Unit	Quantity / Application	Defaults
Ratio	Base unit for %, ppm, με, user-defined etc.	Same as SI base types
Count	Base unit for events, counters	Same as SI base types
Bytes	Binary digits, base unit for memory size	Same as SI base types
Boolean	Return from alarm algorithms etc.	Data format = Boolean DataSize does not apply
BitVector	Discretes, status, concatenated parameters	Data format does not apply DataSize default is 16-bits RangeMaximum and RangeMinimum does not apply
BitSequence	MPEG4, MPEG2, CVSD etc.	Data format indicates the algorithm used DataSize is mandatory, default is 16-bits RangeMaximum and RangeMinimum does not apply

Table 4.2: *The six XidML 2 base types that are non-SI*

4.1.1.2. The reasoning behind these six non-SI base types is shown in Example 4.3:

```
<Parameters>
    <ParameterTypeSet>
        <ParameterType Name="MyMicroStrain">
                <BaseUnit>Ratio</BaseUnit>
                <Scale>10e-6</Scale>
                <RangeMaximum>2000</RangeMaximum>
                <RangeMinimum>0</RangeMinimum>
        </ParameterType>
        <ParameterType Name="MyFuelLevelPercentage">
                <BaseUnit>Ratio</BaseUnit>
                <Scale>10e-2</Scale>
                <RangeMaximum>100</RangeMaximum>
                <RangeMinimum>0</RangeMinimum>
        </ParameterType>
        <ParameterType Name="MyEvents">
                <BaseUnit>Count</BaseUnit>
                <RangeMaximum>65535</RangeMaximum>
                <RangeMinimum>1000</RangeMinimum>
    </ParameterType>
        <ParameterType Name="MyFreeDiskSpace_MiB">
                <BaseUnit>Bytes</BaseUnit>
                <Scale>1048576</Scale>
                <RangeMaximum>80</RangeMaximum>
                <RangeMinimum>0</RangeMinimum>
        </ParameterType>
        <ParameterType Name="MyGoNogoCondition">
                <BaseUnit>Boolean</BaseUnit>
        </ParameterType>
    <ParameterType Name="MyDiscreteInputs">
                <BaseUnit>BitVector</BaseUnit>
                <SizeInBits>14</SizeInBits>
        </ParameterType>
        <ParameterType Name="MyMPEG4video">
                <BaseUnit>BitVector</BaseUnit>
                <DataFormat>BitStream</DataFormat>
        </ParameterType>
    </ParameterTypeSet>
</Parameters>
```

Example 4.3: *Non-SI base type*

4.2. Grouping parameters

4.2.0.1. Parameters, packages, algorithms and displays can be grouped for viewing or sorting with software tools as shown in Example 4.4:

```
<Parameters>
    <ParameterGroupSet>
        <Group Name="MyAircraftParameters">
            <Group Name="MyLeftWingParameters">
              <Group Name="MyEngineParameters">
                  <Reference>MyTemperature1002</Reference>
                  <Reference>MyTemperature1003</Reference>
                  <Reference>MyTemperature1004</Reference>
              </Group>
              <Group Name="MyFuelParameters">
                  <Reference>MyTankPressure</Reference>
                  <Reference>MyTankLevel</Reference>
              </Group>
            </Group>
        </Group>
        <Group Name="MyGroundParameters">
            <Reference>MyBitSyncStatus</Reference>
            <Reference>MyDecomStatus</Reference>
        </Group>
    </ParameterGroupSet>
</Parameters>
```

Example 4.4: *Grouping parameters*

4.2.0.2. Group reference points to a named parameter. Group and Reference can not both be children of the same parent.

4.3. Defining a parameter

4.3.0.1. Example 4.5: shows a parameter (MyAnalogInput) read from Channel 1 of an instrument (MyADC) and transmitted via a PCM stream (MyPcmStream) in PCM locations defined for parameter MyP1.

4.3.0.2. This parameter is also destined for a built in test (BIT) module that has a predefined alarm algorithm running on a parameter MyIn1

```
<Parameters>
    <ParameterTypeSet>
        <ParameterType Name="mVolts">
            <BaseUnit>Volt</BaseUnit>
            <Scale>1e-3</Scale>
        </ParameterType>
    </ParameterTypeSet>
    <ParameterSet>
        <Parameter Name="MyAnalogInput">
            <ParameterProperties>
                <ParameterTypeReference> My_mVolts</ParameterTypeReference>
            </ParameterProperties>
            <Source>
                <Signal>
                    <InstrumentReference>MyADC</InstrumentReference>
                    <VendorMap>Channel1</VendorMap>
                </Signal>
            </Source>
            <Destination>
                <Package>
                    <InstrumentReference>MyPcmEncoder</InstrumentReference>
                    <PackageReference>MyPcmStream</PackageReference>
                    <ParameterMap>MyP1</ParameterMap>
                </Package>
            </Destination>
            <Destination>
                <Algorithm>
                    <InstrumentReference>MyBitModule</InstrumentReference>
                    <AlgorithmReference>MyStuckAtCheck</AlgorithmReference>
                    <ParameterMap>MyInput1</ParameterMap>
                </Algorithm>
            </Destination>
        </Parameter>
    </ParameterSet>
</Parameters>
```

Example 4.5: *Defining a parameter*

4.3.0.3. We will discuss in Chapter 5 how the PCM stream package is defined.

4.3.0.4. We will discuss in Chapter 6 how the built in test algorithm is defined.

4.3.0.5. We will discuss in Chapter 7 how the instruments are setup with respect to excitation etc.

4.3.0.6. We assume that the instrument MyADC is capable of measuring the range specified for the parameter type mVolts and formatting the data as specified. In other words the vendor will check the range and set the gain and offset accordingly when programming the instrumentation.

4.3.0.7. Likewise we assume that the instrument MyPcmEncoder is capable of transmitting the PCM stream as described in the package MyPcmStream.

4.3.0.8. If the package already contains the parameter MyAnalogInput then the package map need not be used. However package maps allow generic package definitions to be reused.

4.3.0.9. A parameter may have only one source but may go to multiple packages, for example a PCM stream and a IRIG-106 chapter 4 storage packet.

4.3.0.10. A parameter may go to multiple algorithms, for example one algorithm might return a strict GO/NOGO Boolean as part of a preflight check another might return the value in other units.

4.3.0.11. In the previous example we assume the data can be transported from MyADC to MyPcmEncoder in a way that we do not need to define via a package. Perhaps they are both from the same vendor, in a rack or chassis with a backplane.

4.3.0.12. The next example assumes that the parameter is transported via another PCM stream (MyOtherPcmStream) to a decoder on the ground (MyPcmDecoder).

```
<Parameters>
    <ParameterSet>
        <Parameter Name="MyAnalogInput">
            <ParameterProperties>
                <ParameterTypeReference>mVolts</ParameterTypeReference>
            </ParameterProperties>
            <Source>
                <Signal>
                    <InstrumentReference>MyADC</InstrumentReference>
                    <VendorMap>Channel1</VendorMap>
                </Signal>
            </Source>
            <Transport>
                <InstrumentReference>MyOtherPcmEncoder</InstrumentReference>
                <InstrumentReference>MyPcmDecoder</InstrumentReference>
                <PackageReference>MyOtherPcmStream</PackageReference>
            </Transport>
            <Destination>
                <Package>
                    <InstrumentReference>MyPcmEncoder</InstrumentReference>
                    <PackageReference>MyPcmStream</PackageReference>
                    <ParameterMap>MyP1</ParameterMap>
                </Package>
            </Destination>
            <Destination>
                <Algorithm>
                    <InstrumentReference>MyBitModule</InstrumentReference>
                    <AlgorithmReference>MyStuckAtCheck</AlgorithmReference>
                    <AlgorithmInstance>MyInstance</AlgorithmInstance>
                    <ParameterMap>MyInput1</ParameterMap>
                </Algorithm>
            </Destination>
        </Parameter>
    </ParameterSet>
</Parameters>
```

Example 4.6: *Source/destination*

4.3.0.13. This example also assumes that MyStuckAtCheck algorithm is run on multiple parameters separately. To link inputs and outputs to multiple instance of a single algorithm the AlgorithmInstance element is used.

4.4. Introducing balancing

4.4.0.1. After bonding strain gages, for example, it is possible that the bridge is so unbalanced that for the gain applied the reading is "off-the-scale". XidML 2 supports the automatic balancing of each gage as shown in Example 4.7:

4.4.0.2. In Example 4.7: the analysis engineer suggests that for the loading involved for the aircraft under test this gage should read between 45 and 55$\mu\varepsilon$ (target ± tolerance).

4.4.0.3. In Example 4.7: the vendor supports Current Shunting as a method of balancing where a current (in mA) is applied to balance the bridge (in Example 4.7: 5.001mA was required to balance to 49$\mu\varepsilon$).

```
<Parameters>
    <ParameterSet>
        <Parameter Name="MyStrainGageNo9876">
            <ParameterProperties>
                <ParameterTypeReference>MyMicroStrain</ParameterTypeReference>
                <BalanceSettings BalanceThisTime="true">
                    <BalanceType>CurrentShunt</BalanceType>
                    <Target>50</Target>
                    <Tolerance>5</Tolerance>
                    <Actual>-200</Actual>
                    <Achieved>49</Achieved>
                    <Applied>5.001</Applied>
                </BalanceSettings>
            </ParameterProperties>
            <Source>
                <Signal>
                    <InstrumentReference>Myreference</InstrumentReference>
                    <VendorMap>Parameter1</VendorMap>
                </Signal>
            </Source>
        </Parameter>
    </ParameterSet>
</Parameters>
```

Example 4.7: *Setting balance conditions for a strain gage*

4.4.0.4. The reading with zero current applied is noted under Actual for data tracking purposes.

4.4.0.5. If the bridge is not to be balanced this time, enable is set to False.

4.4.0.6. Other vendors might support Resistor Shunting where a resistor is permanently shunted across on arm. For this type of shunting the shunt resistor value would be recorded under Applied.

4.4.0.7. For other gages Range Shunting can be used where essentially the range is offset to achieve the target reading. In this case the offset added would be recorded under Applied (in $\mu\varepsilon$ or whatever).

4.4.0.8. Defaults for balance settings (enable, target, tolerance) can be set when defining a parameter type.

4.5. Introducing shunting

4.5.0.1. For calibration or wiring checking it is often desirable to shunt a resistor or current across a bridge, or change the excitation or shunt in a known voltage instead of the sensor and check the value then measured. XidML 2 supports the automatic shunting of each gage as shown in Example 4.8:

4.5.0.2. In Example 4.8: the vendor supports changing the excitation as a method of checking the wiring. In this example the excitation is halved and the output should also halve. For a full discussion on shunting and balancing please see xidml.org

```
<Parameters>
    <ParameterSet>
        <Parameter Name="MyStrainGageNo9876">
            <ParameterProperties>
                <ParameterTypeReference>MyMicroStrain</ParameterTypeReference>
            </ParameterProperties>
            <ShuntSettings ShuntThisTime="True">
                <ShuntType>CurrentShunt</ShuntType>
                <ShuntValue>0.1</ShuntValue>
                <Tolerance>5</Tolerance>
                <TargetDeflection>500</TargetDeflection>
                <AchievedDeflection>495</AchievedDeflection>
            </ShuntSettings>
        </Parameter>
    </ParameterSet>
</Parameters>
```

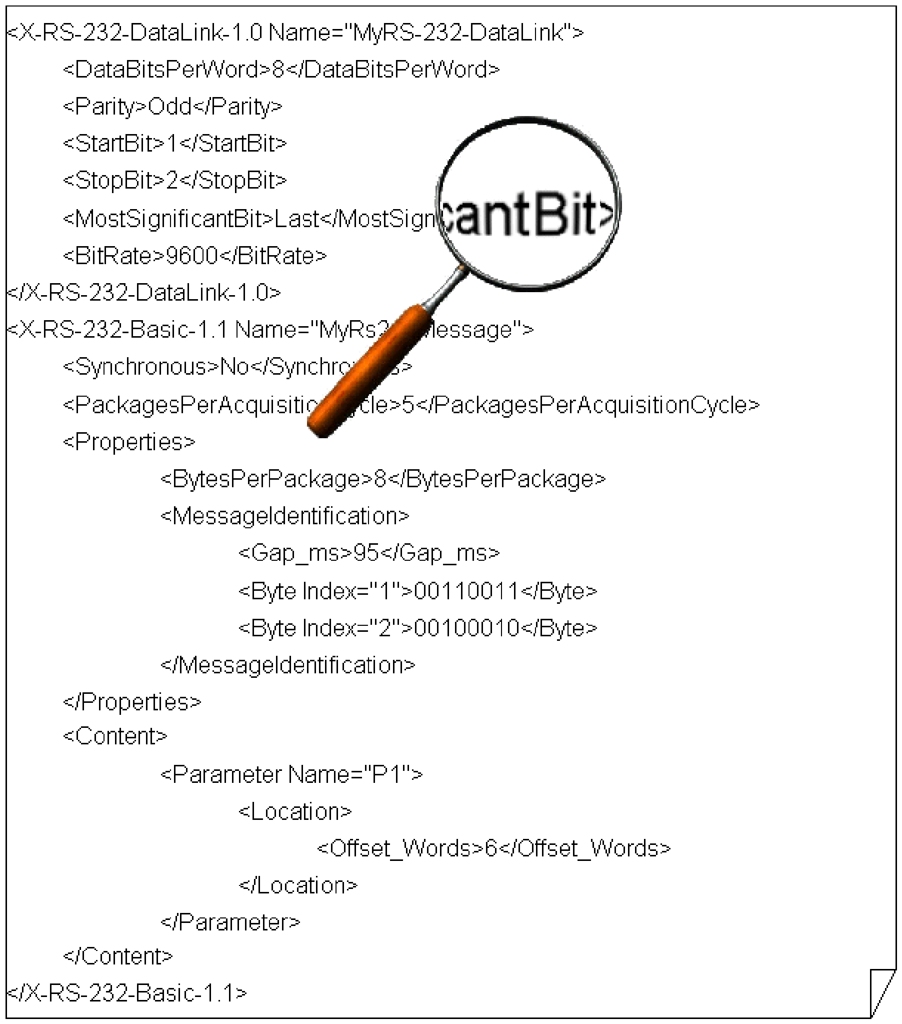
Example 4.8: *Setting shunt conditions for a strain gage*

4.5.0.3. If the bridge is not to be shunted this time enable is set to False.

4.5.0.4. Defaults for shunt settings (enable, shunt value, expected change, tolerance) can be set when defining a parameter type.

THIS PAGE IS INTENTIONALLY BLANK

5 - Creating packages and DataLinks



```
<X-RS-232-DataLink-1.0 Name="MyRS-232-DataLink">
        <DataBitsPerWord>8</DataBitsPerWord>
        <Parity>Odd</Parity>
        <StartBit>1</StartBit>
        <StopBit>2</StopBit>
        <MostSignificantBit>Last</MostSign cantBit>
        <BitRate>9600</BitRate>
</X-RS-232-DataLink-1.0>
<X-RS-232-Basic-1.1 Name="MyRs2   lessage">
        <Synchronous>No</Synchro    >
        <PackagesPerAcquisitio   cle>5</PackagesPerAcquisitionCycle>
        <Properties>
                <BytesPerPackage>8</BytesPerPackage>
                <MessageIdentification>
                        <Gap_ms>95</Gap_ms>
                        <Byte Index="1">00110011</Byte>
                        <Byte Index="2">00100010</Byte>
                </MessageIdentification>
        </Properties>
        <Content>
                <Parameter Name="P1">
                        <Location>
                                <Offset_Words>6</Offset_Words>
                        </Location>
                </Parameter>
        </Content>
</X-RS-232-Basic-1.1>
```

Overview

Packages are used to define how data is sent and stored.

What's new?

All the recommended XidML packages have been divided into four sections called Source, Destination, Properties and Content. Not every package definition will have all four sections.

A DataLink element has been added at the same level as parameters.

5.1. Structure of a package definition

5.1.0.1. All the recommended XidML packages have the same basic structure and are divided into four sections called Source, Destination, Properties, and Content. The Content sections of all package definitions also have the same overall structure containing a parameter with its name and location and the number of times a parameter occurs in a package. Please note that for Occurrences with a value greater than 1, it is assumed that each value is evenly spaced in a package, i.e. with equal time intervals between each sample.

5.1.0.2. The Source element is used to define the source of a data stream defined by a particular package definition. For example, in a MIL-STD-1553 package the Source element specifies a remote terminal ID.

5.1.0.3. The Destination element is used to define the destination of a data stream defined by a particular package definition. For example, in an Ethernet UDP package the Destination element specifies the IP address of the destination for the data packet.

5.1.0.4. The Properties element describes the global characteristics of a particular package. For example, in an IRIG PCM package the Properties section contains the number of minor-frames in a major-frame, the number of bits in a minor-frame and so on.

5.1.0.5. The Content element contains a list of parameters transmitted by a package. Each parameter has a name and a location. The information contained in a location element depends on the package type. For example, the location element in an RS-232 package contains the offset in terms of the number of Words from the beginning of the RS-232 data stream.

5.1.0.6. Not every package definition has all four sections; a package definition only has those sections that make sense in its context. For example, an Ethernet UDP package definition has Source and Destination sections which contain the IP addresses of the origin and destination of the data packet, but IRIG-106 chapter 4 PCM package definitions do not contain this type of information.

5.2. A simple RS-232 message

```
<DataLinks>
    <DataLinkSet>
            <X-RS-232-DataLink-1.0 Name="MyRS-232-DataLink">
                <BitRate_Hz>9600</BitRate_Hz>
                <DataBitPerWord>8</DataBitPerWord>
                <Parity>Odd</Parity>
                <StartBits>1</StartBits>
                <StopBits>2</StopBits>
                <MostSignificantBit>Last</MostSignificantBit>
            </X-RS-232-DataLink-1.0>
    </DataLinkSet>
</DataLinks>
<Packages>
    <PackageSet>
            <X-RS-232-Basic-1.1 Name="MyRs232Message">
                <Synchronous>No</Synchronous>
                <PackagesPerAcquisitionCycle>5</PackagesPerAcquisitionCycle>
                <Properties>
                    <MessageIdentification>
                        <BytesPerPackage>8</BytesPerPackage>
                        <Gap_ms>95</Gap_ms>
                        <StartPattern>
                            <Byte Index="0">33</Byte>
                            <Byte Index="1">22</Byte>
                        </StartPattern>
                    </MessageIdentification>
                </Properties>
                <Content>
                    <Parameter Name="P1">
                        <Location>
                            <Offset_Words>6</Offset_Words>
                        </Location>
                    </Parameter>
                    <Parameter Name="P2">
                        <Location>
                            <Offset_Words>7</Offset_Words>
                        </Location>
                    </Parameter>
                </Content>
            </X-RS-232-Basic-1.1>
    </PackageSet>
</Packages>
```

Example 5.2. *A simple RS-232 message*

5.2.0.1. X-RS-232-Basic is an XidML 2 recommended package for RS-232 messages.

5.2.0.2. Because in this example it is not synchronous; the packages per cycle is just a guide to help in the building of read rates from monitors.

NEW 5.2.0.3. These messages are identified by a gap (of at least 95ms) followed (in this case) by two specific words (0x33 followed by 0x22).

5.2.0.4. If a gap is not defined then identification is by the first two bytes only.

5.2.0.5. If the message identification bytes are not defined then a single message is identified by a gap only.

5.2.0.6. Note the use of the convention "Index" to indicate that the first byte after the identifier is called Byte 0.

5.2.0.7. Two bytes can be combined into one parameter using a bit map algorithm as discussed in Section 6.5.Bit maps.

5.3. Reading data from a MIL-STD-1553 RT to BC message

5.3.0.1. Example 5.3. shows a parameter (MyAltitudeInFt) being read from a MIL-STD-1553 bus monitor (MyBusMonitor).

```
<Packages>
    <PackageSet>
        <X-MIL-STD-1553-Message-1.1 Name="My1553RT2BC">
            <Synchronous>Yes</Synchronous>
            <PackagesPerAcquisitionCycle>128</PackagesPerAcquisitionCycle>
            <RemoteTerminalToBusController>
                <Properties>
                    <NumberOfDataWords>10</NumberOfDataWords>
                </Properties>
                <Source>
                    <RemoteTerminal>1</RemoteTerminal>
                    <SubAddress>
                        <Address>1</Address>
                    </SubAddress>
                    <SubAddressMap>
                        <Map>1</Map>
                        <Map>2</Map>
                    </SubAddressMap>
                </Source>
                <Content>
                    <Parameter Name="P1">
                        <Location>
                            <Offset_Words> 5 </Offset_Words>
                        </Location>
                    </Parameter>
                    <Parameter Name="MyAltitudeInFt">
                        <Location>
                            <Offset_Words>7</Offset_Words>
                        </Location>
                    </Parameter>
                </Content>
            </RemoteTerminalToBusController>
        </X-MIL-STD-1553-Message-1.1>
    </PackageSet>
</Packages>
```

Example 5.3. *Transmitting a MIL-STD-1553 word in a PCM stream*

5.3.0.1. The 1553 package in Example 5.3. defines data transfer from remote terminal 1, sub-address 1 to a bus controller. The message is transmitted synchronously to the bus monitor's operation 128 times per acquisition cycle.

5.3.0.2. <NumberOfWords> is optional. However, when it is specified it allows the bus monitor to distinguish between messages on basis of their length.

5.3.0.3. The transmitting remote terminal supports Mode Code 17 expanded sub-addresses. You can have either a <SubAddressMap> child as shown in the example, or a <CycleNumber> child.

5.4. Simple IRIG-106 chapter 4 PCM stream

```
<Packages>
    <PackageSet>
            <X-IRIG-106-Ch-4-1.1 Name="MyPcmStream">
                  <Synchronous>Yes</Synchronous>
                  <PackagesPerAcquisitionCycle>8</PackagesPerAcquisitionCycle>
                  <Properties>
                        <MajorFrameProperties>
                              <BitsPerMinorFrame>160</BitsPerMinorFrame>
                              <MinorFramesPerMajorFrame>10</MinorFramesPerMajorFrame>

                        <FillPattern>AAAA</FillPattern>
                              <DefaultParity>None</DefaultParity>
                              <DefaultDataBitsPerWord>16</DefaultDataBitsPerWord>
                              <DefaultMostSignificantBit>Last</DefaultMostSignificantBit>
                        </MajorFrameProperties>
                        <SynchronizationStrategy>
                              <SubframeSynchronisationStrategy>
                                    <URC>
                                    <Pattern>0</Pattern>

                                                <NumberOfBits>16</NumberOfBits>
                                          <Location>
                                                <MinorFrameNumber>9</MinorFrameNumber>
                                                <Offset_Words>0</Offset_Words>
                                          </Location>
                                    </URC>
                              </SubframeSynchronisationStrategy>
                              <SyncWord>00001111000011110000111100001111</SyncWord>
                        </SynchronizationStrategy>
                        <Modulation>
                              <PcmCode>NRZ-L</PcmCode>
                              <PCMPolarity>True</PCMPolarity>
                              <DclkPhase>180</DclkPhase>

            </Modulation>
                  </Properties>
                  <Content>
                        <Parameter Name="P1">
                              <Location>
                                    <MinorFrameNumber>8</MinorFrameNumber>
                                    <Offset_Words>16</Offset_Words>
                                    <Occurrences>8</Occurrences>
                              </Location>
                  </Parameter>
                        <Parameter Name="P2">
                              <Location>
                                    <MinorFrameNumber>1</MinorFrameNumber>
                              <Offset_Words>48</Offset_Words>
                                    </Location>
                        </Parameter>
                        <Parameter Name="P3">
                            <Location>
                                <MinorFrameNumber>1</MinorFrameNumber>
                                <Offset_Words>80</Offset_Words>
                            </Location>
                        </Parameter>
                    </Content>
                </X-IRIG-106-Ch-4-1.1>
    </PackageSet>
</Packages>
```

Example 5.4. *A simple IRIG-106 PCM stream*

5.4.0.1. X-IRIG-106-Ch4-Basic-1.0 is a XidML 2 recommended package for PCM streams where the commutation is even and the data word size is always the same (X-IRIG-106-Ch4-Complete-1.0 allows non-even events and word lengths.

5.4.0.2. Words per minor-frame include the SyncWord.

5.4.0.3. In *Example 5.4. A simple IRIG-106 PCM stream*, P3 appears only once per major-frame. Since the <Occurrences> element is optional and its default value is 1, the <Occurrences> element does not need to be included. P1 is in every minor-frame in words 8, 16, 24, etc. (first data word after SyncWord is word 1).

5.4.0.4. Please note that SyncWord mask and tolerance are not defined here, they are defined in the instrumentation settings for the decoder. This is because the mask and tolerance capabilities are functions of the decoder only and legal values may vary from vendor to vendor.

5.4.0.5. Please see *Chapter 8 - Looking at a complete example* for a more complete example.

5.5. Ethernet packet

```
<Packages>
    <PackageSet>
        <X-IENA-Ethernet-UDP-Basic-1.1 Name="MyStdPkt">
            <Synchronous>Yes</Synchronous>
            <PackagesPerAcquisitionCycle>1024</PackagesPerAcquisitionCycle>
            <Properties>
                <Key>1234</Key>
                <Status>3F</Status>
                <NumberOfWords>32</NumberOfWords>
                <EndWord>AABB</EndWord>
            </Properties>
            <Source>
                <IPAddress>192.126.1.167</IPAddress>
                <MACAddress>00-12-23-34-56-78</MACAddress>
                <Port>52000</Port>
            </Source>
            <Destination>
                <IPAddress>192.126.1.168</IPAddress>
                <MACAddress>66-45-35-23-89-00</MACAddress>
                <Port>51000</Port>
            </Destination>
            <Content>
                <Parameter Name="P1">
                    <ID>4455</ID>
                    <Delay>Yes</Delay>
                    <Location>
                        <Offset_Words>0</Offset_Words>
                        <Occurrences>2</Occurrences>
                    </Location>
                </Parameter>
                <Parameter Name="P2">
                    <ID>1221</ID>
                    <Delay>Yes</Delay>
                    <Location>
                        <Offset_Words>1</Offset_Words>
                        <Occurrences>8</Occurrences>
                    </Location>
                </Parameter>
            </Content>
        </X-IENA-Ethernet-UDP-Basic-1.1>
    </PackageSet>
</Packages>
```

Example 5.5. *Ethernet packet*

5.5.0.1. Example 5.5. shows a XidML 2 recommended package for IENA Ethernet packets over UDP/IP standard parameters only.

5.5.0.2. In the first section, the global properties of the package are defined. In particular, you can specify whether the transmission of the packets is synchronous to the operation of the instrumentation that originates the packets. <PacketsPerAcquisitionCycle> defines the transmission rate. The second part of the package named <Properties> defines all information that is specific to this packet. <Source> and <Destination> contain the UDP/IP setup information for the package. The placement of parameters within the frame is defined in the <Content> section. For each sample placeholder, the package allows to specify optionally ID and Delay. Note the ID is not an alternative to the Name attribute, but rather represents an ID that is prepended to the sample within the frame. When <Occurrences> is greater than 1 (Note: the Occurrences element is optional and one is the default value), the placement of the sample is “implicit” and an application will place the first instance of the sample at the given index and equally space the remaining occurrences throughout the packet.

5.6. DataLinks and InterConnects

5.6.0.1. DataLinks can be used to describe attributes and settings that are common to a particular data link. DataLinks are defined in the DataLinks section of the XidML file. There are a number of predefined DataLinks in XidML but the ability to add 3rd party defined data links is also provided.

5.6.0.2. A named DataLink can be optionally referenced from a package definition but a package is more usually associated with a DataLink using an InterConnect. This allows DataLink settings to be reused by more than one package. A bus or network datalink can be "connected" to an instrument using the interconnect element. Packages are typically "connected" to an instrument when defining parameters read or written to the instrument. Thus datalink settings can be reused for more than one package.

5.6.0.3. Example 8.1 below gives an example of an Ethernet package that has a Source and a Destination in addition to examples of a DataLink and an InterConnect.

```
<Instrumentation>
    <InstrumentSet>
           <X-Module-Ethernet-1.0 Name="MyEthernetModule">
                 <Manufacturer>
                       <Name>ACRA CONTROL</Name>
                       <PartReference>KAD/ETH/001/B</PartReference>
                 </Manufacturer>
                 <Location>MyDAU</Location>
                 <SubLocation>3</SubLocation>
                 <InterConnect>MyEthernetDataLink</InterConnect>
                 <Settings>
                       <Module-Ethernet-1.0>
                             <IPAddress>111.222.1.101</IPAddress>
                             <Port>1300</Port>
                             <TimeToLive>1</TimeToLive>
                       </Module-Ethernet-1.0>
                 </Settings>
           </X-Module-Ethernet-1.0>
    </InstrumentSet>
</Instrumentation>
<DataLinks>
    <DataLinkSet>
           <X-Ethernet-UDP-DataLink-1.0 Name="MyEthernetDataLink">
                 <Type>BaseT</Type>
           </X-Ethernet-UDP-DataLink-1.0>
    </DataLinkSet>
</DataLinks>
<Packages>

<PackageSet>
           <X-IENA-Ethernet-UDP-Basic-1.1 Name="MyEthernetPackage">
                 <Synchronous>Yes</Synchronous>
                 <PackagesPerAcquisitionCycle>2</PackagesPerAcquisitionCycle>
                 <Properties>
                       <Key>1F</Key>
                       <Status>7F</Status>
                       <NumberOfWords>1</NumberOfWords>
                       <EndWord>FFFF</EndWord>
                       </Properties>
```

```
                    <Destination>
                              <IPAddress>112.223.1.102</IPAddress>
                              <Port>1300</Port>
                              <MACAddress>00-00-00-00-00-00</MACAddress>
                    </Destination>
                    <Content>
                              <Parameter Name="P1">
                                        <Location>
                                                  <Offset_Words>1</Offset_Words>
                                        </Location>
                              </Parameter>
                    </Content>
          </X-IENA-Ethernet-UDP-Basic-1.1>
    </PackageSet>
</Packages>
```

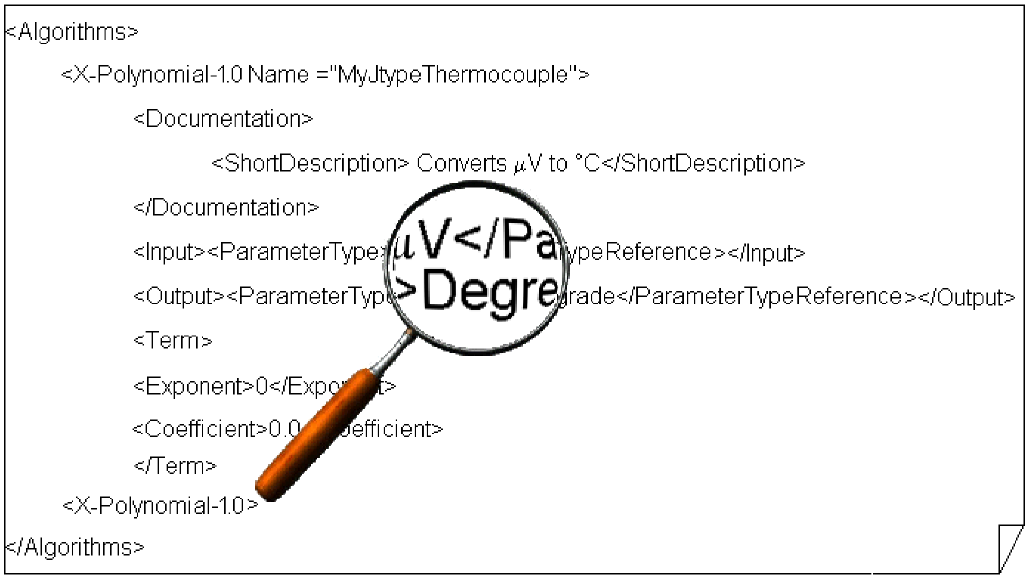
Example 5.6. *Example showing DataLink and InterConnect*

5.6.0.1. *Table 5.1 DataLinks and packages* gives examples of how DataLinks and Packages are used with other packages.

Package type	**Package defined**	**Datalink defined**
IRIG-106 PCM	Number of minor frames Bits per minor frame Fill word Format identifier	Type (Rs-422/RS-422/TTL/RF)
1553	Number of data words Mode code	Maximum response time Retries on fail
RS-232	Type Bytes per package Message identification	Termination (i.e. Direct or Transformer coupled)
IENA Ethernet UDP	KeyStatus Number of words End word Time to live	Type (BaseT, TX, FX etc.) Bit-rate

Table 5.1 *DataLinks and packages*

6 - Creating algorithms



```
<Algorithms>
    <X-Polynomial-1.0 Name ="MyJtypeThermocouple">
        <Documentation>
            <ShortDescription> Converts µV to °C</ShortDescription>
        </Documentation>
        <Input><ParameterType µV</Pa ypeReference></Input>
        <Output><ParameterTyp >Degre rade</ParameterTypeReference></Output>
        <Term>
        <Exponent>0</Expo t>
        <Coefficient>0.0 oefficient>
        </Term>
    <X-Polynomial-1.0>
</Algorithms>
```

Overview

The Algorithm schema is used to define conversion functions, alarm algorithms, embedded and concatenated parameters and general math functions carried out on parameters.

What's new?

The algorithms section of the XidML 2.2 schema does not differ from that of the XidML 2.1 schema. However, some text in the schema examples may have been updated in this chapter since the last revision of the *XidML Handbook.*

6.1. Polynomials

6.1.0.1. Polynomials are a common way of specifying conversion functions. Example 6.1: shows how μV can be converted to °C for a J-Type thermocouple.

```
<Algorithms>
        <AlgorithmSet>
                <X-Polynomial-1.0 Name="MyJtypeThermocouple">
                        <Documentation>
                                <ShortDescription>Converts mV to Degrees C</ShortDescription>
                        </Documentation>
                        <Input>
                                <ParameterTypeReference>mV</ParameterTypeReference>
                        </Input>
                        <Output>
                                <ParameterTypeReference>DegreesCentigrade</ParameterTypeReference>
                        </Output>
                        <Term>
                                <Exponent>0</Exponent>
                                <Coefficient>0.0</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>1</Exponent>
                                <Coefficient>1.9528268e-2</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>2</Exponent>
                                <Coefficient>-1.2286185e-6</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>3</Exponent>
                                <Coefficient>-1.0752178e-9</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>4</Exponent>
                                <Coefficient>-5.9086933e-13</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>5</Exponent>
                                <Coefficient>-1.7256713e-16</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>6</Exponent>
                                <Coefficient>-2.8131513e-21</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>7</Exponent>
                                <Coefficient>-2.3963370e-24</Coefficient>
                        </Term>
                        <Term>
                                <Exponent>08</Exponent>
                                <Coefficient>-8.3823321e-29</Coefficient>
                        </Term>
                </X-Polynomial-1.0>
        </AlgorithmSet>
</Algorithms>
```

Example 6.1: *Conversion polynomial for a J-Type thermocouple*

6.1.0.2. This can be written as

$$°C = \sum_{j=0}^{8} Term \cdot \mu V^{Exponent}$$

6.1.0.3. Undefined exponents are assumed to be zero.

6.1.0.4. The Polynomial schema uses attributes to facilitate sorting and finding of instrumentation-data. It also makes it easier to read.

6.1.0.5. Using units in InputParameters is optional. An algorithm writer may use units if the units of a parameter are important.

6.2. Tables

6.2.0.1. Another common (and powerful) way of defining conversion functions is to use Lookup tables (or arrays). Example 6.2: shows part of the inverse function (°C to mV) for the J-Type thermocouple discussed in Section 6.1. Polynomials.

```
<Algorithms>
        <AlgorithmSet>
                <X-Table-1.0 Name="MyJtypeThermocoupleTo_mV">
                        <Documentation>
                                <ShortDescription>Converts °C to mV</ShortDescription>
                        </Documentation>
                        <Input>
                                <ParameterTypeReference>DegreesCentigrade</ParameterTypeReference>
                        </Input>
                        <Output>
                                <ParameterTypeReference>mV</ParameterTypeReference>
                        </Output>
                        <Entry>
                                <Input>0</Input>
                                <Output>-8.095</Output>
                        </Entry>
                        <Entry>
                                <Input>1</Input>
                                <Output>-8.076</Output>
                        </Entry>
                        <Entry>
                                <Input>2</Input>
                                <Output>-8.057</Output>
                        </Entry>
                        <Entry>
                                <Input>3</Input>
                                <Output>-8.057</Output>
                        </Entry>
                        <Entry>
                                <Input>1399</Input>
                                <Output>69.496</Output>
                        </Entry>
                </X-Table-1.0>
        </AlgorithmSet>
</Algorithms>
```

Example 6.2: *Conversion table for a J-type thermocouple*

6.2.0.2. For an input of -198°C the output would be -8.057V.

6.2.0.3. For an input of 1200°C the output would be -69.553μV.

6.2.0.4. Example 6.2: should have 1401 entries but for clarity we left most out.

6.2.0.5. If steps are undefined, or a value in between steps is required, then the recommended interpolation convention would be a straight-line fit between the higher point and the lower point.

6.2.0.6. If values are outside the defined range, then the recommended convention would be a straight-line fit through the first and last points.

6.3. N samples in one out algorithms

6.3.0.1. Often there is a need to take only the average or maximum over a large number of samples, perhaps to reduce the data transmitted.

```
<X-NsamplesInOneOut-Maximum-1.0 Name="MyMaximum">
      <NumberOfSamples>200</NumberOfSamples>
</X-NsamplesInOneOut-Maximum-1.0>
```

Example 6.3: *Maximum of large number of samples*

6.3.0.2. This example returns the maximum of the last 200 values written to the device.

6.3.0.3. Other similar algorithms are:

- X-NsamplesInOneOut-Minimum-1.0, Minimum of the samples written to the device
- X-NsamplesInOneOut-Amplitude-1.0, Difference between the largest and smallest values written to the device
- X-NsamplesInOneOut-Average-1.0, Average of the samples written to the device
- X-NsamplesInOneOut-RMS-1.0, RMS of the samples written to the device

6.4. MathML

6.4.0.1. MathML is a good choice for describing mathematical functions.

6.4.0.2. Example 6.4: describes a function to obtain the average of two parameters.

```
<Algorithms>
<AlgorithmSet>
      <X-MathML-1.0 Name="MyMathML">
          <Documentation>
                  <ShortDescription>Out=(In1+In2)/2</ShortDescription>
          </Documentation>
          <Input Name="In1">
                  <ParameterTypeReference>Feet</ParameterTypeReference>
          </Input>
          <Input Name="In2">
                  <ParameterTypeReference>Feet</ParameterTypeReference>
          </Input>
          <Output Name="Out1">
                  <ParameterTypeReference>Feet</ParameterTypeReference>
          </Output>
          <mml:apply>
                  <mml:divide/>
                  <mml:apply>
                          <mml:plus/>
                          <mml:cn>In1</mml:cn>
                          <mml:cn>In2</mml:cn>
                  </mml:apply>
                  <mml:cn>2</mml:cn>
          </mml:apply>
      </X-MathML-1.0>
</AlgorithmSet>
</Algorithms>
```

Example 6.4: *MathML function to get the average of two inputs*

6.4.0.3. MathML is not so human readable - although many editors are available to help.

6.4.0.4. There are two important rules for mathematical functions (however they are defined).

6.4.0.5. **Rule 1:** Implement the function in terms of the units, not the data format. For example if temperature (-100 to 1000°C) is expressed in Offset Binary and we want to get the mean of two samples (e.g. -100 +1000°C).

$$rms_{units} = \sqrt{\frac{-100^2 + 1000^2}{2}} = 704$$

$$rms_{offsetbinary} = \sqrt{\frac{-0^2 + 65536^2}{2}} \cdot \frac{1100}{65536} - 100 = 677$$

6.4.0.6. **Rule 2:** Before multiplying or dividing two parameters the output units must be comparable. For example power (W) = Voltage (V) x Current (A) or the unit-less ratio (δ) = V/V.

6.5. Bit maps

6.5.0.1. Often there is a need to concatenate bits of multiple parameters or extract embedded data from a parameter. Sometimes we simply want to reverse bit orders or invert bits. Example 6.5: shows a 12-bit 2's complement parameter embedded across two parameters being mapped onto a 16-bit offset binary parameter as described by the logic equation below
Out(15 downto 0) <= not In1(3) & In1(2 downto 0) & In(15 to 8) & 0000;

```
<Algorithms>
    <AlgorithmSet>
        <X-Bitmap-1.0 Name="My12Bit2scomplementTo16BitOffsetBinary">
            <MapInverse>
                <OutParameter>Out</OutParameter>
                <OutStartIndex>15</OutStartIndex>
                <InParameter>In1</InParameter>
                <InStartIndex>3</InStartIndex>
            </MapInverse>
            <MapTrue>
                <OutStartIndex>12</OutStartIndex>
                <OutEndIndex>14</OutEndIndex>
                <InParameter>In1</InParameter>
                <InStartIndex>2</InStartIndex>
                <InEndIndex>0</InEndIndex>
            </MapTrue>
            <MapTrue>
                <OutStartIndex>4</OutStartIndex>
                <OutEndIndex>11</OutEndIndex>
                <InParameter>In2</InParameter>
                <InStartIndex>15</InStartIndex>
                <InEndIndex>8</InEndIndex>
            </MapTrue>
            <MapFixed>
                <OutStartIndex>0</OutStartIndex>
                <OutEndIndex>3</OutEndIndex>
                <FixedValue>0000</FixedValue>
            </MapFixed>
        </X-Bitmap-1.0>
    </AlgorithmSet>
</Algorithms>
```

Example 6.5: *Converting an embedded 12-bit 2's complement to a 16-bit offset binary*

6.5.0.2. The end indices are not needed for single bit maps.

6.5.0.3. The OutParameters and InParameters are only needed if more than one input or output is being defined.

6.5.0.4. MapInverse inverts all the bits indexed.

6.5.0.5. MapFixed is used to assign fixed values to all bits.

6.6 Alarms

6.6.0.1. In a system with 1000s of parameters it may be necessary to set conditions to be met for the test to start (a Preflight check) or additionally limits are set for safety during a test.

6.6.0.2. Common requirements are that parameters are within a window or that they are not stuck at a value indicating a faulty meter.

6.6.0.3. The algorithm below returns a True if any of the conditions are met.

```
<Algorithms>
    <AlgorithmSet>
        <X-Alarm-1.0 Name="MyPreFlightCheckStopConditions">
                <Window>
                        <InParameter>In1</InParameter>
                        <OKMaximum>5.4</OKMaximum>
                        <OKMinimum>3.2</OKMinimum>
                </Window>
                <Window>
                        <InParameter>In2</InParameter>
                        <OKMaximum>0</OKMaximum>
                </Window>
                <Stuck>
                        <InParameter>In3</InParameter>
                        <Readings>200</Readings>
                </Stuck>
        </X-Alarm-1.0>
    </AlgorithmSet>
</Algorithms>
```

Example 6.6: *Defining preflight checks*

6.6.0.4. In this example the algorithm will return TRUE if

(In1 > 5.4) or (In1 < 3.2)

or (In2 > 0)

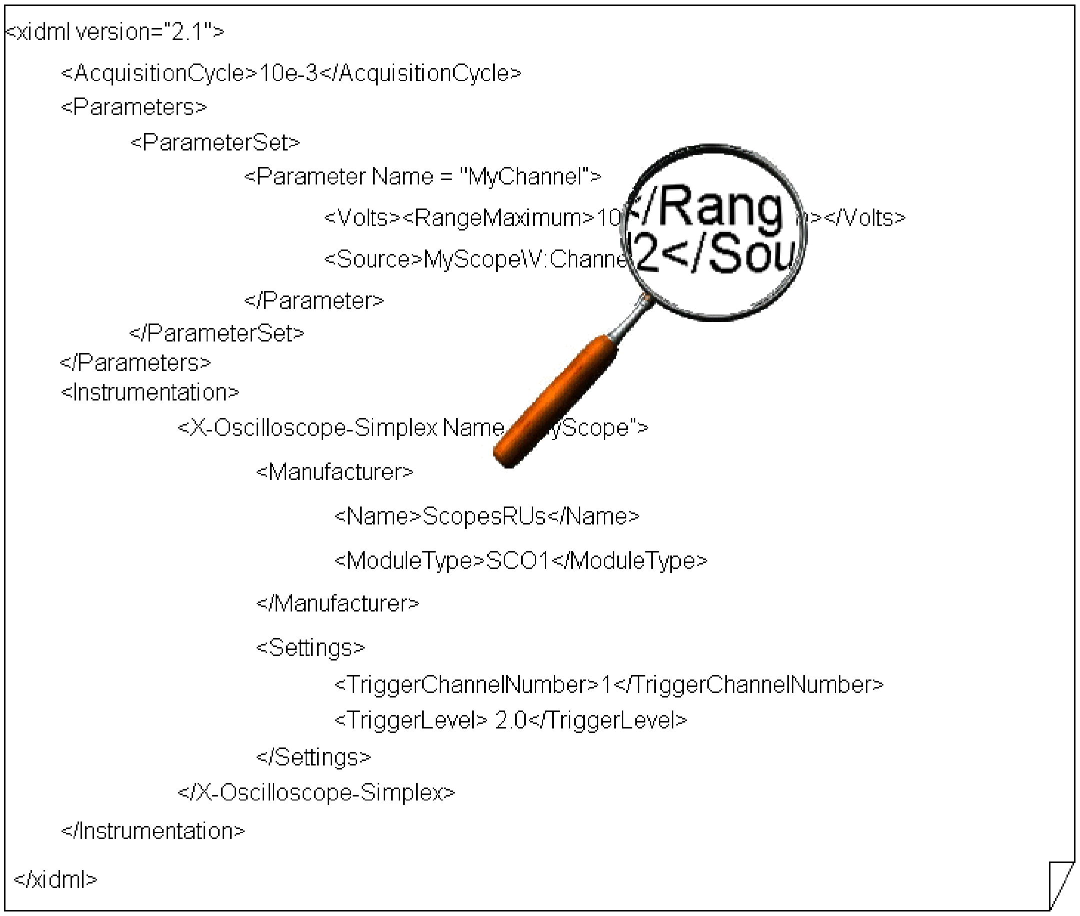
or In3 stays the same for 200 readings

6.6.0.5. Another algorithm can be used to return TRUE for warning conditions.

6.6.0.6. Other algorithms can be used to define error and warning conditions in flight.

THIS PAGE IS INTENTIONALLY BLANK

7 - Defining instrumentation settings



```
<xidml version="2.1">
      <AcquisitionCycle>10e-3</AcquisitionCycle>
      <Parameters>
             <ParameterSet>
                         <Parameter Name = "MyChannel">
                                 <Volts><RangeMaximum>10</Rang      ></Volts>
                                 <Source>MyScope\V:Channe2</Sou
                         </Parameter>
             </ParameterSet>
      </Parameters>
      <Instrumentation>
                  <X-Oscilloscope-Simplex Name      yScope">
                          <Manufacturer>
                                  <Name>ScopesRUs</Name>
                                  <ModuleType>SCO1</ModuleType>
                          </Manufacturer>
                          <Settings>
                                  <TriggerChannelNumber>1</TriggerChannelNumber>
                                  <TriggerLevel> 2.0</TriggerLevel>
                          </Settings>
                  </X-Oscilloscope-Simplex>
      </Instrumentation>
</xidml>
```

Overview

Imagine an analyst requesting a parameter measuring temperatures between ±100°C using a PT100 from the instrumentation engineer. The instrumentation engineer talks to the meter vendor who says "sure, our meter can do that, just specify the linearization to use, the excitation you want, the input impedance you would like, and where to set the filter".

They agree to define these things in the Instrumentation section of the XidML 2 schema. The Instrumentation schema is where all the information specific to the instrumentation is defined.

What's new

An extra element has been added under each of the Settings elements in all instrumentation schemas. This element allows settings to be reused for instruments which combine the functionality of multiple modules.

7.1. Simple oscilloscope

7.1.0.1. 7.1 outlines a complete XidML 2 file for a simple oscilloscope

```
<AcquisitionCycle>10e-3</AcquisitionCycle>
        <Documentation>
                <ShortDescription>This is a complete example of a 2 channel oscilloscope</ShortDescription>
        </Documentation>
        <Parameters>
                <ParameterTypeSet>
                        <ParameterType Name="MyVolts">
                                <BaseUnit>Volt</BaseUnit>
                                <RangeMaximum>10</RangeMaximum>
                        </ParameterType>
                        <ParameterType Name="MyMilliVolts">
                                <BaseUnit>Volt</BaseUnit>
                                <Scale>1e-3</Scale>
                                <RangeMaximum>100</RangeMaximum>
                        </ParameterType>
                </ParameterTypeSet>
                <ParameterSet>
                        <Parameter Name="MyChannel1">
                                <ParameterProperties>
                                        <ParameterTypeReference>MyVolts</ParameterTypeReference>
                                </ParameterProperties>
                                <Source>
                                        <Signal>
                                                <InstrumentReference>MyScope</InstrumentReference>
                                                <VendorMap>Channel1</VendorMap>
                                        </Signal>
                                </Source>
                        </Parameter>
                        <Parameter Name="MyChannel2">
                                <ParameterProperties>
                                        <ParameterTypeReference>MyMilliVolts</ParameterTypeReference>
                                </ParameterProperties>
                                <Source>
                                        <Signal>
                                                <InstrumentReference>MyScope</InstrumentReference>
                                                <VendorMap>Channel2</VendorMap>
                                        </Signal>
                                </Source>
                        </Parameter>
                </ParameterSet>
        </Parameters>
        <Instrumentation>
                <InstrumentSet>
                <Instrument Name="MyScope">
                        <Manufacturer>
                                <Name>ScopesRUs</Name>
                                <PartReference>SCO1</PartReference>
                        </Manufacturer>
                        <scopeRUS:Settings>
                              <scopeRUS:TriggerChannelNumber>1</scopeRUS:TriggerChannelNumber>
                              <scopeRUS:TriggerLevel> 2.1</scopeRUS:TriggerLevel>
                        </scopeRUS:Settings>
                </Instrument>
        </InstrumentSet>
</Instrumentation>
```

Example 7.1: *The instrumentation schema is where you define settings particular to the instrument*

7.1.0.1. The time-base has been set using the acquisition cycle to 10ms.

7.1.0.2. The gain for each channel has been set using RangeMaximum. For this oscilloscope RangeMinimum = -RangeMaximum so it need not be specified.

7.1.0.3. Also, there are limited ranges available (Volts = 10, 1, 0.1, 0.01 and 0.001) but this is not enforceable in the schema definition. It is up to the instrumentation vendor to make sure that the values specified are legal.

7.1.0.4. In the source definition MyScope is the name given to the instrument.

7.1.0.5. The vendor allows two parameters to be read from the oscilloscope that the vendor calls: Channel1 and Channel2.

7.2. Bus monitor

7.2.0.1. 7.2 shows four parameters sourced from a MIL-STD-1553 bus monitor. The example shows the different ways to specify a parameter source.

- The first parameter "MyDATA0Param" is sourced from the package "MyMessage" that is used by the "MyMonitor" instrument. Because there is no alias specified (i.e. no ParameterReference element), the parameter name itself is used to determine the placement of the parameter in the 1553 message package definition.
- The second parameter "MyDATA1Param" is also sourced from the package "MyMessage" but in this case the alias "P1" is used to determine the placement of the parameter in the package.
- The fourth parameter "MySnarferParam" is sourced directly from the "MySnarferDefinitionPackage" package definition

NEW 7.2.0.2. The “MaximumResponseTime” for the 1553 data link is set to 4 microseconds (please see *Chapter 5 - Creating packages and DataLinks* for more details on DataLinks).

```
<Parameters>
    <ParameterSet>
            <Parameter Name="MyDATA0Param">
                <ParameterProperties>
                    <ParameterTypeReference>MyDataType</ParameterTypeReference>
                </ParameterProperties>
                <Source>
                    <Package>
                      <InstrumentReference>My1553Monitor</InstrumentReference>
                      <PackageReference>MyMessage</PackageReference>
                    </Package>
                </Source>
            </Parameter>
            <Parameter Name="MyDATA1Param">
                <ParameterProperties>
                    <ParameterTypeReference>MyDataType</ParameterTypeReference>
                </ParameterProperties>
                <Source>
                    <Package>
                      <InstrumentReference>My1553Monitor</InstrumentReference>
                      <PackageReference>MyMessage</PackageReference>
                      <ParameterMap>P1</ParameterMap>
                    </Package>
                </Source>
            </Parameter>
            <Parameter Name="MySnarferParam">
                <ParameterProperties>
                    <ParameterTypeReference>MyDataType</ParameterTypeReference>
                </ParameterProperties>
                <Source>
                    <Signal>
                      <InstrumentReference>My1553Monitor</InstrumentReference>
                      <VendorMap>SNARFER_LO</VendorMap>
                    </Signal>
                </Source>
            </Parameter>
    </ParameterSet>
</Parameters>
<Instrumentation>
```

```
    <InstrumentSet>
            <X-Module-1553-Monitor-1.2 Name="My1553Monitor">
                    <Manufacturer>
                            <Name>BusMonitorsRUs</Name>
                            <PartReference>Bus8000</PartReference>
                    </Manufacturer>
                    <InterConnect>My1553DataLink</InterConnect>
            </X-Module-1553-Monitor-1.2>
    </InstrumentSet>
</Instrumentation>
<DataLinks>
    <DataLinkSet>
            <X-1553-DataLink-1.0 Name="My1553DataLink">
                    <MaximumResponseTime>4</MaximumResponseTime>
            </X-1553-DataLink-1.0>
    </DataLinkSet>
</DataLinks>
```

Example 7.2: *An example of parameters sourced from a 1553 message*

NEW

7.3. Message buffers

7.3.0.1. Some monitor schemas, including the X-Module-1553-Monitor schema, allow multiple messages to be read from the same buffer.

7.3.0.2. *Example 7.3: Example showing message buffering* shows an excerpt from a XidML file that shows how to setup a MIL-STD-1553 bus monitor to store MIL-STD-1553 bus traffic. The example shows a message buffer called “MyMessageBuffer” that stores traffic (i.e. data plus any tags) from two messages called “Message_1” and “Message_2”.

7.3.0.3. Other standard module schemas also allow a user to define message or packets to be sequenced or buffered.

```
<Instrumentation>
    <InstrumentSet>
        <X-Module-1553-Monitor-1.2 Name="My1553Monitor">
            <Manufacturer>
                <Name>BusMonitorsRUs</Name>
                <PartReference>Bus8000</PartReference>
            </Manufacturer>
            <Settings>
                <Module-1553-Monitor-1.2>
                    <MessageBuffers>
                        <MessageBuffer Name="MyMessageBuffer">
                          <MessageReference Index="0">MyMessage1</MessageReference>
                          <MessageReference Index="2">MyMessage2</MessageReference>
                        </MessageBuffer>
                    </MessageBuffers>
                </Module-1553-Monitor-1.2>
            </Settings>
        </X-Module-1553-Monitor-1.2>
    </InstrumentSet>
</Instrumentation>
```

Example 7.3: *Example showing message buffering*

7.4. Defining a generic ADC module

7.4.0.1. 7.4 shows an analog module that uses an algorithm to define linearization. All channels use the same linearization algorithm (i.e. "MyPT100VoltsToDegreesC"). The linearization algorithm itself is included in an auxiliary file.

NEW 7.4.0.2. The filter cutoff frequency is defined as a percentage of the sampling frequency.

```
<AuxiliaryFiles>
        <AuxiliaryFile>MyAlgorithms</AuxiliaryFile>
</AuxiliaryFiles>
        <Parameters>
                <ParameterSet>
                        <Parameter Name="MyChannel1Parameter">
                                <ParameterProperties>
                                        <ParameterTypeReference>MyVolts</ParameterTypeReference>
                                </ParameterProperties>
                                <Source>
                                        <Signal>
                                                <InstrumentReference>MyA2D</InstrumentReference>
                                                <VendorMap>Channel1</VendorMap>
                                        </Signal>
                                </Source>
                        </Parameter>
                </ParameterSet>
        </Parameters>
        <Instrumentation>
                <InstrumentSet>
                        <X-Module-Analog-In-1.1 Name="MyA2D">
                                <Manufacturer>
                                        <Name>A2DRUs</Name>
                                        <PartReference>A2D300</PartReference>
                                </Manufacturer>
                                <Settings>
                                        <Module-Analog-In-1.1>
                                                <Channel Index="0">
                                                        <Linearization>MyPT100VoltsToDegreesC</Linearization>
                                                        <FilterCutoff>0.25</FilterCutoff>
                                                </Channel>
                                                <Channel Index="1">
                                                        <Linearization>MyPT100VoltsToDegreesC</Linearization>
                                                        <FilterCutoff>0.25</FilterCutoff>
                                                </Channel>
                                                <Channel Index="2">
                                                        <Linearization>MyPT100VoltsToDegreesC</Linearization>
                                                        <FilterCutoff>0.5</FilterCutoff>
                                                </Channel>
                                                <Channel Index="3">
                                                        <Linearization>MyPT100VoltsToDegreesC</Linearization>
                                                        <FilterCutoff>0.5</FilterCutoff>
                                                </Channel>
                                        </Module-Analog-In-1.1>
                                </Settings>
                        </X-Module-Analog-In-1.1>
                </InstrumentSet>
        </Instrumentation>
```

Example 7.4: *An example of how sourced parameters could be linearized*

7.5. Decoder

7.5.0.1. This example illustrates a generic XidML 2 schema for a PCM decoder. It shows one decoder (i.e. "MyDecoder1") with two busses and a stream defined for each. The decoder also defines various instrument-specific setup information for each of the busses.

```
<Parameters>
        <ParameterSet>
                <Parameter Name="MyParameter1">
                        <ParameterProperties>
                                <ParameterTypeReference>MyVolts</ParameterTypeReference>
                        </ParameterProperties>
                        <Source>
                                <Package>
                                        <InstrumentReference>MyDecoder1</InstrumentReference>
                                        <PackageReference>MyStreamDef</PackageReference>
                                </Package>
                        </Source>
                </Parameter>
                <Parameter Name="MyParameter2">
                        <ParameterProperties>
                                <ParameterTypeReference>MyVolts</ParameterTypeReference>
                        </ParameterProperties>
                        <Source>
                                <Package>
                                        <InstrumentReference>MyDecoder1</InstrumentReference>
                                        <PackageReference>MyOtherStreamDef</PackageReference>
                                </Package>
                        </Source>
                </Parameter>
        </ParameterSet>
</Parameters>
<Instrumentation>
        <InstrumentSet>
                <X-Module-PCM-In-1.1 Name="MyDecoder1">
                        <Manufacturer>
                                <Name>DecodersRUs</Name>
                        </Manufacturer>
                        <Settings>
                                <Module-PCM-In-1.1>
                                        <Bus Name="myPCMIn1">
                                                <MatchesToLock>3</MatchesToLock>
                                                <MissesToLoss>1</MissesToLoss>
                                                <SyncWordMask>1001011010</SyncWordMask>
                                                <SyncWordErrorTolerance>4</SyncWordErrorTolerance>
                                                <MinorFrameBitTolerance>3</MinorFrameBitTolerance>
                                        </Bus>
                                        <Bus Name="myPCMIn2">
                                                <MatchesToLock>2</MatchesToLock>
                                                <MissesToLoss>1</MissesToLoss>
                                                <SyncWordMask>1001011010</SyncWordMask>
                                                <SyncWordErrorTolerance>4</SyncWordErrorTolerance>
                                                <MinorFrameBitTolerance>3</MinorFrameBitTolerance>
                                        </Bus>
                                </Module-PCM-In-1.1>
                        </Settings>
                </X-Module-PCM-In-1.1>
        </InstrumentSet>
</Instrumentation>
```

Example 7.5: *An example of two decoders sourcing two different streams*

7.6. Encoder

7.6.0.1. One of the advantages of using (and reusing) generic packages and algorithms is how little is then left to be defined in instrumentation. *Example 7.6: An example of an encoder, a decoder and a parameter sourced from an algorithm* shows the instrumentation setting for a PCM encoder and an RF transmitter for the PCM stream discussed in *Section 7.5. Decoder*

```
<Instrumentation>
        <InstrumentSet>
                <X-Module-PCM-In-1.1 Name="MyEncoder">
                        <Manufacturer>
                                <Name>EncodersRUs</Name>
                                <PartReference>PCM5000Encoder</PartReference>
                                <SerialNumber>54654</SerialNumber>
                        </Manufacturer>
                        <Location>MyDAU</Location>
                        <SubLocation>3</SubLocation>
                        <InterConnect>MyPcmToRfLink </InterConnect>
                        <Settings>
                                <Module-PCM-In-1.1>
                                        <Bus Name="MyPCMIn1">
                                            <SyncWordMask>11111110011010110100100000100000</SyncWordMask>
                                            <MatchesToLock>2</MatchesToLock>
                                            <MissesToLoss>1</MissesToLoss>
                                        </Bus>
                                </Module-PCM-In-1.1>
                        </Settings>
                </X-Module-PCM-In-1.1>
                <X-RF-Tx-1.1 Name="MyRfTx">
                        <Manufacturer>
                                <Name>RfLinksRUs</Name>
                                <PartReference>RF5000</PartReference>
                        </Manufacturer>
                        <InterConnect>MyPcmToRfLink</InterConnect>
                        <Settings>
                                <RF-Tx-1.1>
                                        <TransmitterID>12</TransmitterID>
                                        <BasebandCompositeBandwidth>1e3</BasebandCompositeBandwidth>
                                        <PreModulationFilter>
                                                <BandWidth>10e3</BandWidth>
                                                <Slope>0.9</Slope>
                                                <Type>CA - Constant Amplitude</Type>
                                        </PreModulationFilter>
                                        <Frequency>488.2</Frequency>
                                </RF-Tx-1.1>
                        </Settings>
                </X-RF-Tx-1.1>
        </InstrumentSet>
</Instrumentation>
```

Example 7.6: *An example of an encoder, a decoder and a parameter sourced from an algorithm*

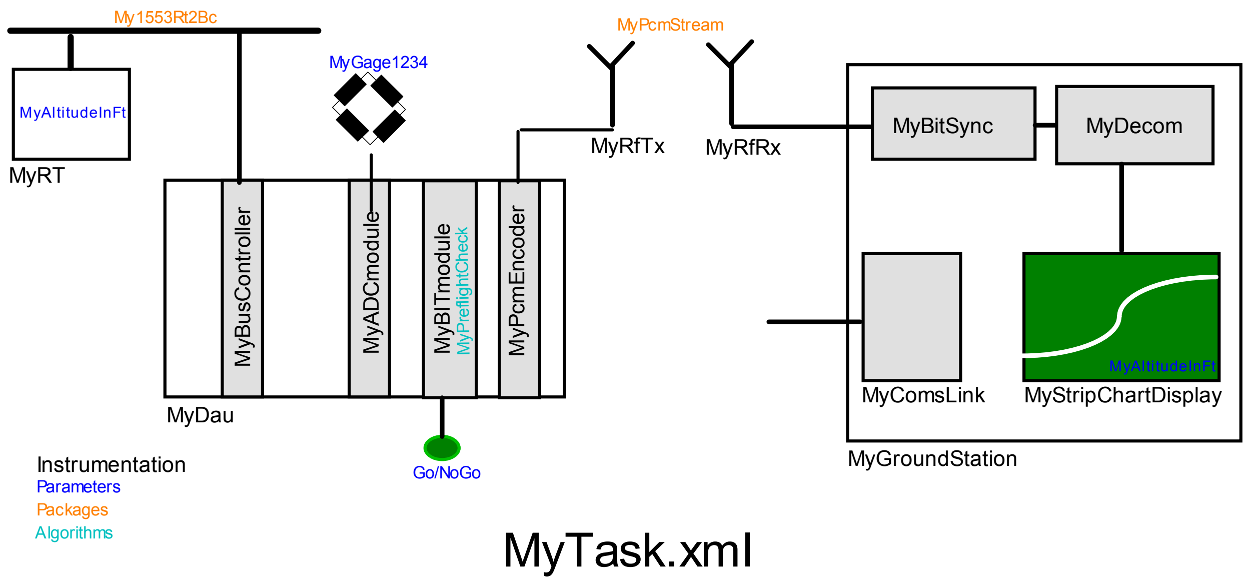
7.6.0.1. Interconnect are optional indicators of what hardware is connected to what.

7.6.0.2. For the encoder the PCM code etc. is defined in the package being transmitted which is indicated in the destination or transport setting for any parameter in the stream.

7.6.0.3. For both devices the bit-rate is a product of the bits per major-frame and major-frames per acquistion cycle divided by the acquisition cycle time.

7.6.0.4. Please note that having the encoder as a child of settings allows future modules to have multiple encoders (or even an encoder and decoder) on the same module, and reuse the Settings children.

8 - Looking at a complete example



Overview

In this chapter we describe a complete XidML 2 definition for the system above. For clarity the information is broken down into files less than a page long and each file is discussed below it or on the opposite page. It may help to print this chapter out, or at least the block diagram above.

The task is to obtain a parameter from the MIL-STD-1553 remote terminal and transmit it to ground via an IRIG-106 PCM stream where it will be displayed on a strip chart.

This task also includes monitoring a strain gage bridge and defining the conditions for a GO/NOGO LED for the pilot.

For simplicity it is assumed that all instruments are programmed (before the flight) via a Ethernet port on a ground station PC.

What's new?

The documentation, parameter, and algorithm sections of the XidML 2.2 schema do not differ from that of the XidML 2.1 schema.

All the recommended XidML packages have been divided into four sections called Source, Destination, Properties and Content. Not every package definition will have all four sections. A DataLink element has been added at the same level as parameters.

An extra element has been added under each of the Settings elements in all instrumentation schemas. This element allows settings to be reused for instruments which combine the functionality of multiple modules.

8.1. The main file (MyTask.xml)

```
<AuxiliaryFiles>
    <AuxiliaryFile>c:\MyTask\MyParameters.xml</AuxiliaryFile>
    <AuxiliaryFile>c:\MyPackages\MyPcmStream.xml</AuxiliaryFile>
    <AuxiliaryFile>c:\MyPackages\My1553RT2BC.xml</AuxiliaryFile>
    <AuxiliaryFile>c:\MyPackages\MyOtherFile.xml</AuxiliaryFile>
    <AuxiliaryFile>c:\MyTask\MyAircraft.xml</AuxiliaryFile>
    <AuxiliaryFile>c:\MyTask\MyGroundStation.xml</AuxiliaryFile>
</AuxiliaryFiles>
<AcquisitionCycle>20e-3</AcquisitionCycle>
<Documentation>
    <CreatedBy>The author of the XidML 2 Handbook</CreatedBy>
    <CreatedDate>2004-9-31</CreatedDate>
    <ShortDescription>Describes the system on first page of chapter 8</ShortDescription>
    <LastUpdated>2004-10-1</LastUpdated>
    <References>
       <Reference>
       <Alias>Handbook</Alias>
       <Location>http://www.xidml.org/documentation/handbook.html</Location>
       <Description>The XidmL 2 Handbook</Description>
       </Reference>
    </References>
</Documentation>
<Parameters>
    <ParameterTypeSet>
       <ParameterType Name="MyMicroStrain">
             <BaseUnit>Ratio</BaseUnit>
             <Scale>10e-6</Scale>
             <RangeMaximum>2000</RangeMaximum>
       </ParameterType>
       <ParameterType Name="MyGoNogoCondition">
             <BaseUnit>Boolean</BaseUnit>
             <RangeMaximum>1</RangeMaximum>
             <RangeMinimum>0</RangeMinimum>
       </ParameterType>
       <ParameterType Name="MyStatus">
             <BaseUnit>BitVector</BaseUnit>
       </ParameterType>
       <ParameterType Name="MyFeet">
             <BaseUnit>Meter</BaseUnit>
             <Scale>0.3048</Scale>
             <RangeMaximum>65535</RangeMaximum>
       </ParameterType>
    </ParameterTypeSet>
</Parameters>
```

Example 8.1: *MyTask.xml*

8.1.0.1. All files start (and end) with the XidML 2 element with the version number as an attribute.

8.1.0.2. Only the main file has auxiliary files. Each of the auxiliary files are discussed in this chapter one page at a time.

8.1.0.3. In this task we are reusing packages already defined in a directory (folder) called MyPackages. This is good practice.

8.1.0.4. The acquisition cycle is 20ms which means that, in this example, the rate at which the MIL-STD-1553 bus operates and the PCM major frames are sent is a multiple of 50Hz. This means that all parameters are also sampled at a multiple of 50Hz.

8.1.0.5. The creation and last updated dates follow the recommended ISO 8601 date format (yr/month/day).

NEW 8.1.0.6. The strain gages all measure the range ±2000με, and when they are balanced the default target is 50με (We will overwrite these defaults later for one of the channels). Please see example in *Section 8.6. Rest of parameter definition (MyParameters.xml).*

8.1.0.7. There is a built in test module (MyBITmodule) in MyDAU running level checks on each of the strain gage modules. An LED warns the pilot if any are out of range.

8.1.0.8. The MIL-STD-1553 bus controller (MyBusController) and the built in test module (MyBITmodule) both have 16-bit status parameters with bits to indicate an error occurred, and other bits to indicate what caused the error (RT did not respond, parameter out of range etc.). Therefore BitVector was chosen as the base type.

8.1.0.9. One of the parameters to be read from the MIL-STD-1553 remote terminal is altitude (which the pilots insist must be displayed in feet).

NEW 8.1.0.10. All strain gages are to be checked that they are within ±1000με and are not stuck at the same value for more than 200 readings as defined in the alarm algorithm. Please see example in *Section 8.6. Rest of parameter definition (MyParameters.xml).*

8.1.0.11. Another algorithm is defined to extract the six bits indicating the error code from the 16-bit status word.

8.1.0.12. Auxiliary file elements have an optional attribute called ArchiveThisTime. Its default value is true and is a value used by tools to determine if an auxiliary file needs to be archived.

8.2. MyPCMStream.xml

```
<Packages>
    <PackageSet>
        <X-IRIG-106-Ch-4-1.1 Name="MyPcmStream">
                <Synchronous>Yes</Synchronous>
                <PackagesPerAcquisitionCycle>2</PackagesPerAcquisitionCycle>
                <Properties>
                        <MajorFrameProperties>
                                <DefaultDataBitsPerWord>16</DefaultDataBitsPerWord>
                                <DefaultParity>None</DefaultParity>
                                <DefaultMostSignificantBit>Last</DefaultMostSignificantBit>
                                <MinorFramesPerMajorFrame>16</MinorFramesPerMajorFrame>
                                <BitsPerMinorFrame>256</BitsPerMinorFrame>
                                <FillPattern>FFFF</FillPattern>
                        </MajorFrameProperties>
                        <SynchronizationStrategy>
                                <SubframeSynchronisationStrategy>
                                    <SyncWordComplement>
                                        <FCC>
                                                <MinorFrameNumber>125</MinorFrameNumber>
                                        </FCC>
                                    </SyncWordComplement>
                                </SubframeSynchronisationStrategy>
                                <SyncWord>00001111000011110000111100001111</SyncWord>
                        </SynchronizationStrategy>
                        <Modulation>
                                <PcmCode>NRZ-L</PcmCode>
                                <PCMPolarity>True</PCMPolarity>
                                <DclkPhase>180</DclkPhase>
                        </Modulation>
                </Properties>
                <Content>
                        <Parameter Name="MyP1">
                                <Location>
                                    <MinorFrameNumber>1</MinorFrameNumber>
                                    <Offset_Words>25</Offset_Words>
                                    <Occurrences>8</Occurrences>
                                </Location>
                        </Parameter>
                        <Parameter Name="MyP2">
                                <Location>
                                    <MinorFrameNumber>1</MinorFrameNumber>
                                    <Offset_Words>8</Offset_Words>
                                    <Occurrences>2</Occurrences>
                                </Location>
```

```
                    </Parameter>
                <Parameter Name="MyP3">
                    <Location>
                      <MinorFrameNumber>1</MinorFrameNumber>
                      <Offset_Words>8</Offset_Words>
                    </Location>
                </Parameter>
            </Content>
        </X-IRIG-106-Ch-4-1.1>
    </PackageSet>
</Packages>
```

Example 8.2: *MyPCMStream.xml*

8.3. MyOtherFile.xml

```
<Algorithms>
<AlgorithmSet>
    <X-Alarm-1.0 Name="MyStrainGageCheck">
        <Window>
                <OKMaximum>1000</OKMaximum>
                <OKMinimum>-1000</OKMinimum>
        </Window>
        <Stuck>
                <Readings>200</Readings>
        </Stuck>
    </X-Alarm-1.0>
    <X-Bitmap-1.0 Name="MyBitErrorCode">
        <MapTrue>
                <OutStartIndex>5</OutStartIndex>
                <OutEndIndex>0</OutEndIndex>
                <InStartIndex>7</InStartIndex>
                <InEndIndex>2</InEndIndex>
        </MapTrue>
    </X-Bitmap-1.0>
</AlgorithmSet>
</Algorithms>
<Packages>
<PackageSet>
    <X-MIL-STD-1553-Message-1.1 Name="My1553RT2BC">
        <Synchronous>Yes</Synchronous>
        <PackagesPerAcquisitionCycle>4</PackagesPerAcquisitionCycle>
        <RemoteTerminalToBusController>
                <Source>
                        <RemoteTerminal>1</RemoteTerminal>
                        <SubAddress>
                                <Address>1</Address>
                        </SubAddress>
                        <SubAddressMap>
                                <Map>01</Map>
                                <Map>00</Map>
                        </SubAddressMap>
                </Source>
                <Content>
                        <Parameter Name="MyAltitudeInFt">
                                <Location>
                                    <Offset_Words>7</Offset_Words>
                                </Location>
                        </Parameter>
                </Content>
        </RemoteTerminalToBusController>
    </X-MIL-STD-1553-Message-1.1>
```

```
</PackageSet>
</Packages>
```

Example 8.3: *Packages*

8.3.0.1. These files are auxiliary and as such do not include auxiliary files.

8.3.0.2. Both packages are synchronous with respect to the acquisition cycle. In particular the bus controller will read the RT synchronously with respect to when the data is transmitted in the PCM stream.

8.3.0.3. From the main file we know there are 50 acquisition cycles per second. For the PCM stream there are two packets (major-frames) per acquisition cycle and 16 minor-frames per major-frame and 128x16 bits per minor-frame so the bit-rate will be

$$50 \times 2 \times 16 \times 128 \times 16 = 3.2768\text{Mbps}.$$

8.3.0.4. We use MinorFramesPerMajorFrame to indicate clearly that the (32) syncword bits are not included. Similarly we use DefaultDataBitsPerWord to indicate that this value does not include parity.

8.3.0.5. Later we will be reading the parameter MyAltitudeInFt from the remote terminal via the bus controller, and transmitting it as MyP2 in the PCM stream. The parameter is read four times per acquisition cycle via MIL-STD-1553, and is transmitted four times per acquisition cycle (2 x 2 = 4). Being synchronous means we don't have to worry about triple buffering.

8.4. Instrumentation (MyAircraft.xml)

```
<Instrumentation>
    <InstrumentSet>
        <X-RF-Tx-1.1 Name="MyRfTx">
                <Manufacturer>
                        <Name>RfRUs</Name>
                </Manufacturer>
                <InterConnect>TxLink</InterConnect>
                <Settings>
                        <RF-Tx-1.1>
                                <TransmitterID>655</TransmitterID>
                                <BasebandCompositeBandwidth>333</BasebandCompositeBandwidth>
                                <Frequency>106.6e10</Frequency>
                        </RF-Tx-1.1>
                </Settings>
        </X-RF-Tx-1.1>
        <Instrument Name="MyRT">
                <Manufacturer>
                        <Name>RTRUs</Name>
                </Manufacturer>
                <InterConnect>TxLink</InterConnect>
                <RtRUs:RT>
                        <RtRUs:Settings>
                                <RtRUs:RemoteTerminal>0x2</RtRUs:RemoteTerminal>
                        </RtRUs:Settings>
                </RtRUs:RT>
        </Instrument>
        <X-Module-1553-Monitor-1.2 Name="MyBusMonitor_1">
                <Manufacturer>
                        <Name>RTRUs</Name>
                </Manufacturer>
                <Location>MyDau</Location>

                <SubLocation>3</SubLocation>
                <InterConnect>MyTxLink</InterConnect>
        </X-Module-1553-Monitor-1.2>
        <X-Module-Analog-In-1.1 Name="MyADCModule">
                <Manufacturer>
                        <Name>ADCRUs</Name>
                </Manufacturer>
                <Location>MyDau</Location>
                <SubLocation>4</SubLocation>
                <InterConnect>TxLink</InterConnect>
                <Settings>
                        <Module-Analog-In-1.1>
                                <Channel Index="0">
                                    <FilterCutoff>0.25</FilterCutoff>
                                </Channel>
                        </Module-Analog-In-1.1>
                </Settings>
        </X-Module-Analog-In-1.1>
        <Instrument Name="MyBusMonitor_2">
            <Manufacturer>
                <Name>BitRUs</Name>
            </Manufacturer>
            <Location>MyDau</Location>
            <SubLocation>5</SubLocation>
        </Instrument>
        <X-Module-1553-Monitor-1.2 Name="MyBusMonitor_3">
            <Manufacturer>
                <Name>BusMonitorRUs</Name>
            </Manufacturer>
            <Location>MyDau</Location>
            <SubLocation>6</SubLocation>
            <InterConnect>MyTxLink</InterConnect>
            <Settings>
                <Module-1553-Monitor-1.2>
                    <AcceptMessageWithNoStatus>Yes</AcceptMessageWithNoStatus>
                </Module-1553-Monitor-1.2>
            </Settings>
        </X-Module-1553-Monitor-1.2>

        <X-Module-1553-Monitor-1.2 Name="MyBusMonitor_4">
            <Manufacturer>
                <Name>BusMonitorRUs</Name>
            </Manufacturer>
            <Location>MyGroundStation</Location>
            <SubLocation>6</SubLocation>
            <InterConnect>MyTxLink</InterConnect>
            <Settings>
                <Module-1553-Monitor-1.2>
                    <ModeCode17>
                      <DefaultMapOnPowerUp>7C</DefaultMapOnPowerUp>
                    </ModeCode17>
                </Module-1553-Monitor-1.2>
            </Settings>
        </X-Module-1553-Monitor-1.2>
    </InstrumentSet>
</Instrumentation>
<DataLinks>
    <DataLinkSet>
        <X-1553-DataLink-1.0 Name="MyTxLink">
            <MaximumResponseTime>13</MaximumResponseTime>
        </X-1553-DataLink-1.0>
    </DataLinkSet>
</DataLinks>
```

Example 8.4: *Aircraft instrumentation*

8.5. Instrumentation (MyGroundStation.xml)

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:RfRus="http://www.RfRus.com/schemas"
xmlns:BsRUs="http://www.BsRUs.com/schemas"
xmlns:CommsRUs="http://www.CommsRUs.com/schemas"
xmlns:ChartsRUs="http://www.ChartsRUs.com/schemas"
xmlns:DRUs="http://www.DRUs.com/schemas"
xsi:noNamespaceSchemaLocation="R:\src\SATURN\XidML\WebsiteFiles\Schemas\X-Essential\XidML-2.2.xsd">
<Instrumentation>
    <InstrumentSet>
        <Instrument Name="MyRfRx">
            <Manufacturer>
                <Name>RfRUs</Name>
            </Manufacturer>
            <InterConnect>RxLink</InterConnect>
            <RfRus:RFTx>
            <RfRus:Band>S</RfRus:Band>
            </RfRus:RFTx>
        </Instrument>
        <Instrument Name="MyBitSync">
            <InterConnect>RxLink</InterConnect>
            <BsRUs:BitSync>
                <BsRUs:InputResistance>High</BsRUs:InputResistance>
                <BsRUs:LoopBandwidth>5</BsRUs:LoopBandwidth>
            </BsRUs:BitSync>
        </Instrument>
        <X-Module-Ethernet-1.0 Name="MyCommsLink_1">
            <Manufacturer>
                <Name>CommsRUs</Name>
            </Manufacturer>
            <Settings>
            <Module-Ethernet-1.0>
                <IPAddress>192.126.1.168</IPAddress>
                <Port>1023</Port>
                <MACAddress>00-12-23-34-56-78</MACAddress>
            </Module-Ethernet-1.0>
            </Settings>
        </X-Module-Ethernet-1.0>
        <X-Module-Ethernet-1.0 Name="MyCommsLink_2">
            <Manufacturer>
                <Name>CommsRUs</Name>
            </Manufacturer>
        <Settings>
            <Module-Ethernet-1.0>
                <IPAddress>192.126.1.169</IPAddress>
                <Port>1024</Port>
                <MACAddress>00-12-23-34-57-78</MACAddress>
            </Module-Ethernet-1.0>
        </Settings>
        </X-Module-Ethernet-1.0>
        <Instrument Name="MyStripChartDisplay">
            <Manufacturer>
                <Name>ChartsRUs</Name>
            </Manufacturer>
            <DRUs:Display>
            <DRUs:Legend>Altitude_Feet</DRUs:Legend>
```

```
                <DRUs:Type>StripChart</DRUs:Type>
                <DRUs:BackgroundColor>Green</DRUs:BackgroundColor>
                <DRUs:TraceColor>White</DRUs:TraceColor>
                </DRUs:Display>
        </Instrument>
    </InstrumentSet>
</Instrumentation>
```

8.5.0.1. The radio transmitter and receiver are from RfRUs Inc and hence they have chosen RfRUs: to indicate elements to be defined for them.

8.5.0.2. The encoders and decoders do not require much setup as most of the information is defined in the packages. By looking at the parameter definition of any parameter transported via the encoder or decoder we can find the PCM packet and hence the bit-rate.

8.5.0.3. There are multiple PCM encoders on board, so an interconnect element (TxLink) is used to indicate what PCM encoder is connected to the transmitter. The bit-rate can hence be found as it was for the encoder or decoder.

8.5.0.4. All instrumentation is programmed via the Ethernet port so we only have to identify the port we are to use.

8.5.0.5. The display settings have also been defined via XidML 2. This means that when we view information years after its being archived, we can see what the pilots saw as they viewed the data.

8.6. Rest of parameter definition (MyParameters.xml)

```
<Parameters>
    <ParameterTypeSet>
        <ParameterType Name="MyFeet">
            <BaseUnit>Meter</BaseUnit>
            <Scale>3.3</Scale>
        </ParameterType>
    </ParameterTypeSet>
    <ParameterSet>
        <Parameter Name="MyAltitudeInFt">
            <ParameterProperties>
                <ParameterTypeReference>MyFeet</ParameterTypeReference>
            </ParameterProperties>
            <Source>
                <Signal>
                    <InstrumentReference>MyRT</InstrumentReference>
                    <VendorMap>Channel1</VendorMap>
                </Signal>
            </Source>
            <Transport>
                <InstrumentReference>MyRT</InstrumentReference>
                <InstrumentReference>MyBusController</InstrumentReference>
                <PackageReference>My1553message</PackageReference>
            </Transport>
            <Transport>
                <InstrumentReference>MyPcmEncoder</InstrumentReference>
                <InstrumentReference>MyDecom</InstrumentReference>
                <PackageReference>MyPcmStream</PackageReference>
                <ParameterMap>MyP2</ParameterMap>
            </Transport>
            <Destination>
                <Signal>
                    <InstrumentReference>MyStripChartDisplay</InstrumentReference>
                    <VendorMap>Strip1</VendorMap>
                </Signal>
            </Destination>
        </Parameter>
        <Parameter Name="MyStrainGage1234">
            <ParameterProperties>
                <ParameterTypeReference>MyMicroStrain</ParameterTypeReference>
                <BalanceSettings>
                    <BalanceType>CurrentShunt</BalanceType>
                    <Target>50</Target>
                    <Tolerance>5</Tolerance>
                </BalanceSettings>
            </ParameterProperties>
            <Source>
                <Signal>
                    <InstrumentReference>MyADC</InstrumentReference>
                    <VendorMap>Channel1</VendorMap>
                </Signal>
            </Source>
            <Transport>
                <InstrumentReference>MyPcmEncoder</InstrumentReference>
                <InstrumentReference>MyDecom</InstrumentReference>
                <PackageReference>MyPcmStream</PackageReference>
                <ParameterMap>MyP3</ParameterMap>
            </Transport>
            <Destination>
                <Algorithm>
                    <InstrumentReference>MyBitModule</InstrumentReference>
                    <AlgorithmReference>MyStrainGageCheck</AlgorithmReference>
                </Algorithm>
            </Destination>
        </Parameter>
      </ParameterSet>
</Parameters>
```

Example 8.5: *Remainder of parameter definition*

THIS PAGE IS INTENTIONALLY BLANK

A-1 XidML 2 synopsis

Please see A-2 Schemas available from xidml.org, for a full list of schemas available from xidml.org, A full list of base units and SI prefixes are also available from xidml.org

Note 1. In Table A-1: Key XidML 2 elements, the following apply:

Singleton implies that the element can only exist once in a file or under parent.

Reference points to the section in the *XidML 2 Handbook* where the element is first discussed.

Elements that must be uniquely named have "Name = in the example column.

Complex elements with children end in "\" in the element column.

Some elements have a choice of substitutes.

Note 2. AcquistionCycle (recommendations not mandatory)

The acquisition cycle should be kept short (< 1 second) to ensure quick recovery from brown-outs and faster format switching. Also it should be an integer multiple of 1μs, so that 1Mbps outputs from GPS card, 1MHz IRIG time, or CAIS busses can be used to synchronize operation.

XidML 2 allows the acquisition element to be repeated in auxiliary files for independent checking of files, providing the value agrees with the value in the main file.

Note 3. RangeMaximum, RangeMinimum, DataFormat and DataSize

RangeMaximum and RangeMinimum are not used for Boolean, BitVector and BitStream data types. RangeMinimum will default to minus RangeMaximum.

Values set for RangeMaximum, Balance tolerance etc. when defining the parameter type can be overwritten if required for each parameter.

DataSize is mandatory for BitVector and BitStream data types and is not used for Boolean data types

XidML 2 recommends that 16-bit Offset Binary be used where possible as opposed to 2's complement or sign + magnitude as it can be used with unipolar and bipolar and offset ranges.

XidML 2 recommends that when truncating Offset Binary or 2's complement or sign + magnitude parameters that the MSBs be used. Also if expanding these that the LSBs be padded with zeros. This will ensure that the maximum and minimum ranges will still be achieved and that DataSize need not always be specified.

Note 4. Descriptions (recommendations not mandatory)

Short descriptions should be kept to less than 50 characters and on one line as they are often used as part of inventory lists.

Note 5. Names (recommendations not mandatory)

Names should be kept short (<20 characters), have no white space, bold or italic characters or contain any of the following five characters "/><\. Also, it should be noted that specific vendors may not support the full range of characters that are allowed in XidML 2 files. Document authors should check with individual vendors to see which character sets their software supports.

SI prefixes should be used in parameter type names where appropriate (e.g. mVolts).

Non-SI parameter types should be reflected in the parameter type name (e.g My_Ft) and in the parameter name (MyAltitude_Ft)

The prefix "X-" is reserved for Schemas available from xidml.org (e.g. X-MathML), and the prefix "My" is used in examples to clearly indicate a name that can be changed by the user (e.g. MyP1).

The prefix "Data" is used to denote bits and words containing parameters (e.g. DataBitsPerWord or DataWordsPerFrame).

Clear names should be used rather then abbreviations (e.g. RangeMaximum rather than Max).

The word or suffix "Index" should be used to indicate a numbering scheme starting at 0, and the suffix "number" should be used if it starts with 1 and there is a risk of confusion (e.g. DataWordIndex or MinorFrameNumber).

The suffix "Tag" should be reserved for parameters associated with a package but not actual traffic (e.g. TimeTag or StatusTag).

Note 6. BalanceTypes and ShuntTypes (recommendations not mandatory)

One of the following balance types should be used: ResistorShunt, CurrentShunt, or RangeAdjust (where the range is adjusted with a shunt resistor, current, or offset).

One of the following shunt types should be used: ResistorShunt, CurrentShunt, or ExcitationShunt (e.g. the excitation is halved) or Substitution (e.g. a voltage rather than the measurand is multiplexed).

Note 7. Group and Reference

Group can be a child, grandchild etc. of group. Group and Reference can not be children of the same parent.

Key XidML 2 elements

Element	Reference	Mandatory	Type	Singleton	Example	Default	Notes
XidML	§2.1	•		•	Version = "2.2"		The XidML element is the root element of each file and must have a Version attribute
AuxiliaryFiles	§2.1		Complex	•			
AuxiliaryFile	§2.1		URI		C:\Myfile.xml		This element has an optional "ArchiveThisTime" attribute
AcquisitionCycle	§2.1	•	Decimal	•	20e-3	1.0	Please see Note §2
Documentation\	§3.1			•			
CreatedDate	§3.1		See notes	•	2004-12-31		Documentation can also be a single child of parameters, packages, algorithms and instrumentation
CreatedBy	§3.1		Strings	•	Joe Bloggs		
ShortDescription	§3.1		Strings	•	It is nice		XidML 2 uses the ISO 8601 convention for date
LongDescription	§3.1		Strings	•	Four score and ten years		Please see Note §4
LastUpdated	§3.1		See notes	•	2004-12-31		XidML 2 uses the ISO 8601 convention for date
Documentation\References\	§3.1			•			
Documentation\References\Reference\	§3.1		Strings		Name = "RevisionHistory"		One use would be for recording revision changes of a file
Location	§3.1	•	URI	•	http://www.xidml.org		Please see Note §7
ShortDescription	§3.1		Strings	•	It contains technical specs		Please see Note §7
Parameters\	§4.1			•			
Parameters\ParameterGroupSet\Group\	§4.2		Strings		Name = "MyAnalog"		
Group\	§4.2				Name = "MyLeftWing"		
Reference	§4.2		Named parameter		MyLeftWingTemperature		
Parameters\ParameterTypeSet\ParameterType	§4.1				Name = "MyFeet"		Each ParameterType has a unique name attribute
BaseUnit	§4.1	•	See notes	•	Meters	Ratio	
Scale	§4.1		Decimal	•	0.3048	1	User-defined unit is multiplied by scale to get base unit
Offset	§4.1		Decimal	•	0	0	After the user-defined unit is multiplied by the scale the offset is added to get the base unit
RangeMaximum	§4.1	•	Decimal	•	65535		Please see Note §3
RangeMinimum	§4.1		Decimal	•	0	-RangeMaximum	Please see Note §3
DataFormat	§4.1		See notes	•	Integer	OffsetBinary	Please see Note §3
SizeInBits	§4.1		Integer	•	12	16	Please see Note §3
Parameters\ParameterSet\Parameter	§4.3				Name = "MyAltitude"		
\Parameter\Properties\	§4.3			•			
ParameterTypeReference	§4.3	•	Named type	•	MyFeet		
BalanceSettings\	§4.4		Complex				This element has a "BalanceThisTime" attribute
Target	§4.4	•	Decimal	•	100	Mid-range	Target is the value that channel should be balanced to
Tolerance	§4.4	•	Decimal	•	5	Full-scale	Tolerance is the acceptable error with respect to target
Actual	§4.4	•	Decimal	•	200	None	Please see Note §6
Achieved	§4.4	•	Decimal	•	49		
Applied	§4.4	•	Decimal	•	5.001		
BalanceType	§4.4	•	String	•	Current		
ShuntSettings\	§4.5		Complex	•			This element has a "ShuntThisTime" attribute
ShuntValue	§4.5		Decimal	•	0.1		Desired value or change of value when shunt is applied
ExpectedChange	§4.5		Decimal	•	500		
ActualChange	§4.5		Decimal	•	495		
Tolerance	§4.5		Decimal	•	10		Tolerance is the acceptable error with respect to expected input
ShuntType	§4.5		Strings	•	Substitution	None	Please see Note §6
RangeMaximum	§4.3		Decimal	•	100	As set in type	

Element	Reference	Mandatory	Type	Singleton	Example	Default	Notes
RangeMinimum	§4.3		Decimal	•	-100	As set in type	
DataFormat	§4.3		DataFormats	•	TwosComplement	As set in type	
SizeInBits	§4.3		Integer	•	16	As set in type	
Parameters\ParameterSet\Parameter\Source\	§4.3	•	Substitute 1 of 3	•			
Parameters\ParameterSet\Parameter\Source\Signal\	§4.3			•			Signal means the parameter originated in this instrument
InstrumentReference	§4.3	•	Named Instrument	•	MyMeter		The instrument the signal comes from
VendorMap	§4.3	•	Vendor Name	•	Channel1		Name the vendor has given to the parameter to be read
Parameters\ParameterSet\Parameter\Source\Package\	§4.3			•			Package means the parameter originates in a package
InstrumentReference	§4.3	•	Named Instrument	•	MyDecoder		The instrument receiving the package
PackageReference	§4.3	•	Named Package	•	MyPackage		The name of the package
ParameterMap	§4.3		Name in Package	•	MyP1		Only if the parameter has another name in the package
Parameters\ParameterSet\Parameter\Source\Algorithm\	§4.3			•			Algorithm means the parameter originates from an algorithm output
InstrumentReference	§4.3	•	Named Instrument	•	MyMathModule		The instrument running the algorithm
AlgorithmReference	§4.3	•	Named Algorithm	•	MyAlarm		The algorithm name
AlgorithmInstance	§4.3		Strings	•	My1		Only if the same algorithm is running on multiple inputs/outputs
ParameterMap	§4.3		Name in Algorithm	•	MyP1		Only if the algorithm has multiple outputs
Parameters\ParameterSet\Parameter\Transport\	§4.3	•				16	A packet is sent and the receiver is a second source
InstrumentReference	§4.3	•	Named Instrument		MyTransmitter		One instrument is the sender
InstrumentReference	§4.3	•	Named Instrument		MyReceiver		One instrument (at least) is the receiver
PackageReference	§4.3	•	Named Package	•	MyPackage		The name of the package
ParameterMap	§4.3		Name in Package	•	MyP1		Only if the parameter has another name in the package
Parameters\ParameterSet\Parameter\Destination\Signal\	§4.3		Substitute 1 of 3			16	Signal means that parameter goes to, for example, a dial or D/A (not a package or algorithm)
InstrumentReference	§4.3	•	Named Instrument	•	MyStripChart		The instrument the signal goes to
VendorMap	§4.3	•	Vendor Name	•	Channel1		Name the vendor has given to where the parameter goes
Parameters\ParameterSet\Parameter\Destination\Package\	§4.3						Package means the parameter is transmitted or stored
InstrumentReference	§4.3	•	Named Instrument	•	MyEncoder		The instrument transmitting or storing the package
PackageReference	§4.3	•	Named Package	•	MyPackage		The name of the package
ParameterMap	§4.3		Name in Package	•	MyP1		Only if the parameter has another name in the package
Parameters\ParameterSet\Parameter\Destination\Algorithm\	§4.3						Algorithm means the parameter goes to an algorithm input
InstrumentReference	§4.3	•	Named Instrument	•	MyMathModule		The instrument running the algorithm
AlgorirthmReference	§4.3	•	Named Algorithm	•	MyAlarm		The algorithm name
AlgorithmInstance	§4.3		Strings	•	MyP1		Only if the same algorithm is running on multiple inputs/outputs
ParameterMap	§4.3		Name in Algorithm	•	MyP1		Only if the algorithm has multiple inputs
Parameters\ParameterSet\Parameter\Balance Settings\	§4.4			•			
BalanceThisTime	§4.4		Boolean	•	True	As set in type	
TargetValue	§4.4		Decimal	•	100	As set in type	
Tolerance	§4.4		Decimal	•	105	As set in type	
Actual	§4.4		Decimal	•	105		What is actually read with zero balance applied
Achieved	§4.4		Decimal	•	99		The best balancing could achieve
Applied	§4.4		Decimal	•	5.2		
BalanceType	§4.4		Strings	•	Current		

Element	Reference	Mandatory	Type	Singleton	Example	Default	Notes
Parameters\ParameterSet\Parameter\ShuntSettings\	§4.5			•			
ShuntThisTime	§4.5		Boolean	•	True	As set in type	If set this overwrites value set for the parameter type
TargetDeflection	§4.5		Decimal	•	100	As set in type	If set this overwrites value set for the parameter type
Tolerance	§4.5		Decimal	•	105	As set in type	If set this overwrites value set for the parameter type
AchievedDeflection	§4.5		Decimal	•	105		An output from the automated shunt check
ShuntType	§4.5		Strings		Substitution	As set in type	
Packages\	§5.3			•			
Packages\PackageGroupSet\Group\	§4.2		Strings		Name = "MyFTIpackages"		Please see Note §7
Group\	§4.2				Name = "MyPCM"		Please see Note §7
Reference	§4.2		Named Package		MyAlarm		Please see Note §7
Packages\PackageSet\Package\	§5.4		Substitution Group		Name = "MyPackage"		
Synchronous	§5.4	•	Boolean	•	False	True	True means transmission is synchronous with the acquisition cycle
PackagesPerAcquistionCycle	§5.4	•	Integer	•	5	1	
Packages\PackageSet\Package\Source	§5.5				IP address		The source of a data package
Packages\PackageSet\Package\Destination	§5.5				Remote terminal ID		The destination for a data package
Packages\PackageSet\Package\Properties	§5.5				Minor-frames per major-frame		Settings global to a particular package
Packages\PackageSet\Package\Content\Parameter	§5.4				Name = “MyP1”		
\Location\	§5.4						Package specific parameter location information
\Occurrences	§5.4		Integer	•	20	1	The number of samples per package
DataLinks\DataLinkSet\DataLink	§5.6						
Algorithms\	§6.1			•	Name = "MyAlarm"		
Algorithms\AlgorithmGroupSet\Group\	§4.2		Strings		Name = "MyFtiAlgorithms"		Please see Note §7
Group\	§4.2				Name = "MyAlarms"		Please see Note §7
Reference	§4.2		Named Algorithm		Name = "MyDau"		Please see Note §7
Algorithms\AlgorithmSet\Algorithm\Input\	§6.1				Name = "MyP1"		Mandatory if more than one input
ParameterType	§6.1		See notes	•	MyFeet	Ratio	User-defined ParameterType derived from base units
Algorithms\AlgorithmSet\Algorithm\Output\	§6.1				Name = "MyP1"		Mandatory if more than one output
ParameterType	§6.1		See notes	•	MyFeet	Ratio	User-defined ParameterType derived from base units
Instrumentation\	§7.1			•			
Instrumentation\InstrumentgroupSet\Group\	§4.2				Name = "MyFTIequipment"		Please see Note §7
Group\	§4.2				Name = "MyCabinStuff"		Please see Note §7
Reference	§4.2		Named Instrument		MyDau		Please see Note §7
Instrumentation\InstrumentSet\Instrument\	§7.1				Name = "MyMathModule"		Can also be a virtual instrument (e.g. a Display in a PC)
Manufacturer	§7.6		String	•	MyManufacturer		
Location	§7.6		Named Instrument	•	MyDau		(Optional) chassis, DAU, rack, or PC that contains the instrument
SubLocation	§7.6		String/Integer	•	3		Where in the DAU or PC the instrument is
InterConnect	§7.6		Strings		Name = “MyCommsLink”		Allows us to specify a link not defined by a transport element
Settings	§7.6		Complex				May be vendor-specific with a vendor schema
Addendum	§2.1		Complex	•			User or vendor-specific information

Table A-1: *Key XidML 2 elements*

A-2 Schemas available from xidml.org

XidML 2 schemas are separated into files as below. All schemas have a verbose and basic example.

	Name	Description
XidML	XidML-2.2	Validates all XidML 2.2 elements except the children of documentation, parameters, algorithms, packages, and instrumentation
Essential	X-Addendum-1.0	"Free-style" storage of information that is only needed by that vendor or user-group
	X-AlgorithmGlossary-1.0	Contains the list of officially released algorithm schemas
	X-AlgorithmTypes-1.0	
	X-BaseUnits-1.0	Validates all elements associated with Base Units
	X-DataFormats-1.0	Validates all elements associated with Data Formats
	X-DataLinks-1.0	
	X-DataLinkGlossary-1.0	Contains the list of officially released data link schemas.
	X-DataLinkTypes-1.0	
	X-Documentation-1.0	Validates all elements associated with documentation
	X-Groups-1.0	Validates all child elements associated with groups of parameters, algorithms, packages and instrumentation
	X-Instrumentation-1.0	
	X-InstrumentGlossary-1.0	Contains the list of officially released instrumentation schemas
	X-InstrumentTypes-1.1	
	X-Packages-1.1	
	X-PackageGlossary-1.0	Contains the list of officially released package schemas
	X-PackageTypes-1.1	
	X-Parameters-1.0	Validates all child elements associated with Parameters except groups
Packages	X-CCDL-1.0	Defines data on a CCDL bus
	X-CAIS-1.1	Defines a CAIS package
	X-IENA-Ethernet-UDP-Basic-1.1	Ethernet standard parameter package definition for UDP
	X-IRIG-106-Ch-4-1.1	Defines an IRIG-106 Ch 4 PCM stream
	X-IRIG-106-Ch-8-1.0	How monitored 1553 data could be packed into a FIFO
	X-Firewire-1.0	Defines data on a firewire bus
	X-Memory-Storage-1.1	Defines data storage on a memory storage device
	X-MIL-STD-1553-Message-1.0	Defines a MIL-STD-1553 message.
	X-RS-232-Basic-1.1	Message identified by bytes at start of message
	X-Snarfer-1.0	Defines a generic filtered FIFO
Algorithms	X-Alarm-1.0	AlgorithmDescribes basic alarm conditions
	X-Algorithms-1.0	
	X-BitMap-1.0	Describes a generic BitMap algorithm
	X-MathML-1.0	Describes a generic Math algorithm using MathML
	X-NsamplesInOneSampleOut-Amplitude-1.0	Returns difference between largest and smallest of N samples
	X-NsamplesInOneSampleOut-Average-1.0	Returns an average of N samples
	X-NsamplesInOneSampleOut-Maximum-1.0	Returns the largest of N samples
	X-NsamplesInOneSampleOut-Minimum-1.0	Returns the smallest of N samples
	X-NsamplesInOneSampleOut-RMS-1.0	Returns the mean of N samples
	X-Polynomial-1.0	Describes a generic polynomial algorithm
	X-Boolean-Simple-1.0	Performs basic Boolean tests on individual bits from one or more parameters
	X-Table-1.0	Maps one or more input values to an output value
Instrumentation	X-DAU	Generic DAU
	X-DAU-Controller-CAIS-1.0	Generic CAIS DAU backplane controller
	X-DAU-Controller-PCM-1.1	Generic PCM DAU backplane controller
	X-Module-1553-BC-1.1	Generic 1553 bus controller module
	X-Module-1553-Monitor-1.2	Generic 1553 bus monitor module
	X-Module-1553-RT-1.0	Generic 1553 remote terminal
	X-Module-Analog-In-1.1	Generic multi-channel A2D module with excitation, linearization, and filtering
	X-Module-Analog-Out-1.1	Generic multi-channel digital to analog converter.
	X-Module-Arinc-429-Monitor-1.0	Generic Arinc-479 monitor
	X-Module-Arinc-573-Monitor-1.0	Generic Arinc-573 monitor

	Name	Description
	X-Module-Arinc-429-Transmitter-1.0	Generic Arinc-479 transmitter
	X-Module-Arinc-573-Transmitter-1.0	Generic Arinc-573 transmitter
	X-Module-Audio-In-1.1	Generic Audio compression module
	X-Module-BitSynchronizer-1.1	A generic bit synchronization module
	X-Module-Built-In-Test	An instrument that continuously checks parameters are valid
	X-Module-CAIS-BC-1.0	Generic CAIS bus controller.
	X-Module-CCDL-Monitor-1.0	Generic CCDL bus monitor
	X-Module-Discrete-In-1.1	Generic multi-channel module with discrete inputs plus various counters
	X-Module-Ethernet-1.0	Generic Ethernet module
	X-Module-ExternalMultiplexer-Controller-1.1	Generic module for controlling an external multiplexer
	X-Module-Firewire-Monitor-1.0	Generic Firewire bus monitor
	X-Module-IRIGTime-InOut-1.1	Generic IRIG time generator and translator
	X-Module-PCM-In-1.1	Generic multi-channel IRIG-106-Ch4-Decoder
	X-Module-PCM-Out-1.1	Generic multi-channel IRIG-106-Ch4-Encoder
	X-Module-Recorder-1.2	Generic solid-state or other memory storage device
	X-Module-RS-232-Monitor-1.0	Generic RS-232/422 module
	X-Module-RS-232-Uplink	An airborne RS-232 receiver as part of an RF uplink
	X-Module-Video-In-1.2	Generic Video + Audio compression module
	X-RF-Tx-1.1	GenericRF transmitter
	X-RF-Rx-1.1	Generic RF receiver
DataLinks	X-1553-DataLink-1.0	Generic 1553 data link
	X-Arinc-429-DataLink-1.0	Generic Arinc-429 data link
	X-Arinc-573-DataLink-1.0	Generic Arinc-573 data link
	X-CAIS-DataLink-1.0	Generic CAIS data link
	X-Ethernet-UDP-1.0	Generic Ethernet UDP data link
	X-Memory-Storage-DataLink-1.1	Generic a data link to a memory storage device
	X-PCM-DataLink-1.0	Generic PCM data link
	X-RS-232-DataLink-1.0	Generic RS-232/422 data link