The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

# XML TELEMETRIC AND COMMAND EXCHANGE (XTCE)—ELEMENT DESCRIPTION

INFORMATIONAL REPORT

CCSDS 660.1-G-2

GREEN BOOK

August 2021

The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

# XML TELEMETRIC AND COMMAND EXCHANGE (XTCE)—ELEMENT DESCRIPTION

**INFORMATIONAL REPORT**

**CCSDS 660.1-G-2**

**GREEN BOOK**

**August 2021**

# AUTHORITY

| | |
|---|---|
| Issue: | Informational Report, Issue 2 |
| Date: | August 2021 |
| Location: | Washington, DC, USA |

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and reflects the consensus of technical panel experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4).

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
Email: secretariat@mailman.ccsds.org

# FOREWORD

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Report is therefore subject to CCSDS document management and change control procedures, which are defined in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be sent to the CCSDS Secretariat at the email address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Science Policy Office (BELSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- China Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Hellenic Space Agency (HSA)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Netherlands Space Office (NSO)/The Netherlands.
- Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

# DOCUMENT CONTROL

| Document | Title | Date | Status |
|---|---|---|---|
| CCSDS 660.1-G-1 | XML Telemetric and Command Exchange (XTCE)—Element Description, Informational Report, Issue 1 | May 2012 | Original issue, superseded |
| CCSDS 660.1-G-2 | XML Telemetric and Command Exchange (XTCE)—Element Description, Informational Report, Issue 2 | August 2021 | Current issue |

# CONTENTS

# CONTENTS (continued)

# CONTENTS (continued)

# CONTENTS (continued)

Figure Page

# CONTENTS (continued)

Figure Page

# CONTENTS (continued)

Figure Page



Table

# CONTENTS (continued)

# 1 INTRODUCTION

## 1.1 PURPOSE

This Informational Report provides information for using Extensible Markup Language (XML) Telemetric and Command Exchange (XTCE) 1.2 (reference [1]). This document expands upon the annotation and intended usage official XTCE 1.2 specification. XTCE is an XML Schema for describing telemetry and telecommand database items for spacecraft monitoring and control.

XTCE is used to facilitate the exchange of spacecraft telemetry and command databases between different organizations and systems during any mission phase. It is a nonproprietary format that avoids the need for customized import/export tools.

This document is a revision of an earlier version for XTCE 1.1. Further, it is a companion to the high-level introduction contained in the report CCSDS 660.2-G-2 (reference [2]).

## 1.2 SCOPE

The scope of this document is limited to the exchange of satellite telemetry and commanding databases during development through operation phases of a mission. It addresses general XTCE concepts and terminology. Finally, it lists all the major elements and attributes, and many minor ones, to explain all the various aspects of the XTCE Schema and how they interact to make a complete database description.

## 1.3 DOCUMENT STRUCTURE

This Report is organized as follows:

a) Section 1 specifies the purpose and scope.

b) Section 2 provides a general description of XTCE.

c) Section 3 specifies core XTCE concepts.

d) Section 4 describes XTCE elements and attributes.

e) Section 5 describes advanced concepts and complete examples.

f) Section 6 describes a telemetry packet.

g) Section 7 describes a command packet.

h) Section 8 discusses programming and XTCE.

i) Section 9 presents a detailed example.

j) Annex A contains a full XML example.

k) Annex B contains XTCE 1.1 schema keys that have been fixed.

l) Annex C contains a list of acronyms used in the document.

## 1.4 DEFINITIONS

### 1.4.1 TERMS

XTCE is an XML Schema using the World Wide Web Consortium (W3C) Schema language, which is a standard for constructing the rules for a particular XML. From the user's standpoint, XTCE looks like a series of elements and attributes with specific names oriented towards spacecraft telemetry and telecommand. The XTCE Schema itself is a collection of both schema types and derived schema types using the XML Schema language. The following terminology is used in this document:

**XML Schema language**: A W3C standard for building specific rules for XML.

**XTCE Schema**: The rules for any XTCE XML file.

**XTCE file or XTCE instance document**: An XML file that validates against the XTCE Schema using an XML parser.

**XML parser**: A program that reads in XML Schemas and then validates specific XML files against it.

**XML Schema types**: The base types used to form the XML Schema language.

**Derived XML Schema types**: New schema types created for a specific XML Schema, such as XTCE.

**XTCE Schema types**: The created types that form the XTCE Schema.

**Element or tag**: In XML, items formed using angle brackets; the terms *element* and *tag* are interchangeable, for example, <Element>Content</Element>.

NOTE – In this document, the location of elements and attributes in XTCE Schema may be described using a path-like designation such as:

/topElement/middleElement/endElement

This helps orient its location within the schema for the reader.

**Attribute**: In XML, a qualifier within a specific element, such as:

<Element attribute="myAttributeContent">Content</Element>

In textual descriptions, attributes will often be preceded by an '@' such as:

'@attribute' or 'Element/@attribute'

**Annotation**: Descriptive information within XTCE Schema itself.

NOTES

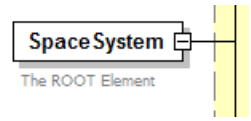
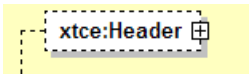
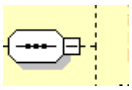
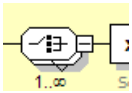
1 The XTCE annotation is considered canon; however, it is also terse. This document expands it but does not knowingly contradict it.

2 In some cases a special 'appinfo' annotation is included whenever it is necessary to perform an additional validity check that is not describable in the W3C schema language.

### 1.4.2 CONVENTIONS

This document uses diagrams generated from the commercial application XMLSpy from Altova Ltd, which shows the XTCE schema visually instead of in code format. Table 1-1 provides a brief guide to these diagrams:

**Table 1-1: Guide to XTCE Diagrams**

| XMLSpy Image | Description | Graphical Symbol Meaning |
|---|---|---|
|   | Solid Box | Required XML element in document |
|   | Box with Dashed lines | Optional XML element in document |
|  | Sequence Box | Ordered set of elements (qualifier for number often present) |
|   | Choice Box | Choice from child elements presented (qualifiers present) |
|   | Attributes Box | Attributes within an element name, such as <element attribute="myAttributeContent"> |

## 1.5 REFERENCES

The following publications are referenced in this document. At the time of publication, the editions indicated were valid. All publications are subject to revision, and users of this document are encouraged to investigate the possibility of applying the most recent editions of the publications indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS publications.

[1] "XML Telemetric and Command Exchange™ (XTCE™)." OMG—Object Management Group®. https://www.omg.org/xtce/index.htm.

[2] *XML Telemetric and Command Exchange (XTCE)*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 660.2-G-2. Washington, D.C.: CCSDS, February 2021.

[3] "All Issues." OMG Dashboards. https://issues.omg.org/issues/.

[4] "Domain Level Task Force." OMG—Object Management Group®. https://www.omg.org/space/.

[5] "XML Tutorial." W3Schools Online Web Tutorials. https://www.w3schools.com/xml/default.asp.

[6] "XML Schema." W3Schools Online Web Tutorials. https://www.w3schools.com/xml/xml_schema.asp.

# 2 OVERVIEW

## 2.1 GENERAL

This document presents a nonproprietary format (XTCE) to describe a telemetry and telecommand mission operations database. A mission operations database is typically the ground format description file ingested by a tool that uses the information to configure itself so that it may decommutate mission telemetry values and build correct mission commands. Additional information typically associated with these 'mission ops database' files are alarms (limits), units, calibrations, algorithms, and descriptive information.

## 2.2 WHAT IS XTCE?

XTCE is an XML Schema (XSD) for describing telemetry and commands needed by space missions to decommutate telemetry and build commands on the ground. XTCE contains rules for element names, element order, attributes, and their content. These rules are used to construct XTCE XML files.

An XTCE document describes a space system. The space system is divided into two sections: one file for telemetry metadata, and another for command metadata (where 'meta' means 'information about'). An XTCE space system may be hierarchical, with each space system containing sub-space systems.

Telemetry items are described using the Parameter element and the associated elements called ParameterTypes. ParameterTypes describe attributes of individual telemetry items such as link encoding, alarms, and calibration information.

Telemetry parameter descriptions are formed into descriptive blocks called containers. The containers describe packaging of the mission telemetry and use inheritance to complete each package description. They are used for marking identified areas and their expected values within packets or minor frames.

Commands share many of the same basic elements with telemetry. As with telemetry, there are command parameters and ParameterTypes. In addition, commands have arguments, which have an ArgumentType. Each mission will decide which command items are needed.

One command may extend another to produce a new command description by inheriting functionality. Command descriptions also use inheritance.

Once XTCE files are constructed, they can be used to exchange telemetry and command descriptions with other users. For new missions, XTCE could be used as a native format for the telemetry and command processing system. XTCE is a broad specification; most users will use a subset of the capabilities.

Many of the XTCE Schema types are reused to create the various elements. This gives the files a repetitive look.

## 2.3 XTCE VERSIONS

### 2.3.1 GENERAL

There are several versions of XTCE that have been published over the years. The first version was XTCE 1.0 published in 2006. Shortly thereafter, version XTCE 1.1 was published in 2006 after adjustments following a CCSDS review. Finally, XTCE 1.2 was published in 2019 after years of accumulated knowledge by end users and deliberations on their findings by the standards community.

This document generally pertains to all of them, but specifically version 1.2.

### 2.3.2 CHANGES FROM XTCE 1.1 TO XTCE 1.2

XTCE 1.2 has many important changes to it but is largely backwards compatible syntactically with XTCE 1.1. Even so, some syntax has changed slightly.

A change of note relates to how arrays are defined in XTCE 1.1 that many users felt was awkard, or just insufficient: in that version, only the number of dimensions is defined in ArrayParameterType, and the size of each dimesion is specified when it is used within a container.

However, in XTCE 1.2 this has been revised. Now the array dimensions and their sizes are specified directly in the ArrayParameterType.

There are more changes: one that is invisible to the end user, but which the programmer will notice occurs with the underlying schema types. In XTCE 1.2. these have been cleaned up to produce a better programmatic mapping when using tools like JAXB (Java XML Binding) to generate a class representation from the schema.

A list of major changes are as follows:

- The SpaceSystem namespace is updated.
- Various enumerated strings values have been normalized.
- Byte order can be specified in several ways.
- Exponents are integers.
- Specifying string length is updated.
- Float encoding sizes are modified.
- Epoch time/date data type updated.
- Argument area supports both an argumentRef and parameterRef.
- Arrays are updated so that their dimension sizes can be specified locally.

- Various annotations updated and expanded.
- Numerous schema types improved or modified.
- The schema keys have been fixed or expanded.
- There is a new alarm for numerics.
- There is an inside/outside flag for some alarms.
- Xinclude is supported.
- Time association has moved to the container entry area.
- Ancillary data and short description appear in even more locations.
- The command side schema types have been optimised somewhat.
- Parameter instance references in arguments has been expanded to include arguments instance references.
- Numerious small bug fixes and typos have been addressed.

# 3 XTCE CORE CONCEPTS AND COMMON ELEMENTS

## 3.1 OVERVIEW

This section introduces several core XTCE concepts. Many of these concepts reappear throughout the schema. They are introduced here to reduce repetition.

## 3.2 OVERALL STRUCTURE

XTCE consists of the following top-level XML structure:

```
<xtce:SpaceSystem> <!-- the root element in XTCE -->
    <xtce:TelemetryMetaData/> <!-- telemetry description section -->
    <xtce:CommandMetaData/> <!-- command description section -->
    <xtce:SpaceSystem/> <!-- optional child sub-SpaceSystems (recursive) -->
</xtce:SpaceSystem>
```

Each of these top-level elements has many child elements and attributes. For example, TelemetryMetaData has the following child element (in abbreviated form):

```
<xtce:TelemetryMetaData> <!-- telemetry description section -->
    <xtce:ParameterTypeSet/> <!-- user types, calibration curves, limits, link info, etc. -->
    <xtce:Parameter/> <!-- user telemetry items: name, links to ParameterType -->
    <xtce:ContainerSet/> <!-- user packaging for telemetry items, packets or minor frames -->
</xtce:TelemetryMetaData>
```

The following is a CommandMetaData child example.

```
<xtce:CommandMetaData> <!-- command description section -->
        <xtce:ParameterTypeSet/> <!-- parameter type area for commands -->
        <xtce:ArgumentTypeSet/> <!-- argument types, similar to above -->
        <xtce:MetaCommandSet/> <!-- cmd description area, arguments, packets etc. -->
        <xtce:CommandContainerSet/> <!-- common items referenced above -->
</xtce:CommandMetaData>
```

Each of these elements has child elements. Some are required; most are optional. In addition, most elements have attributes, and many of the attributes have default values.

## 3.3 NAME REFERENCES

### 3.3.1 GENERAL

XTCE uses a string pointer called NameReference to reference items in other parts of an XTCE file. The reference consists of the name of the item of interest and an optional path to its location in the XTCE file. The path portion is formed by SpaceSystem names and has the same general format as directory names in Unix/Linux systems. In XTCE the name of the item of interest is held in the attribute ‘@name’, which occurs in many elements. The final

path item is the item of interest in a NameReference. For example, using a NameReference, a parameter defined in ParameterSet refers to a ParameterType in ParameterTypeSet. The NameReference in this case consists of the user-supplied name in the attribute @name of the ParameterType.

The rules governing NameReferences are as follows and differ in some aspects to file paths:

a) If a NameReference is only the name of the item being referenced, it is an *unqualified* NameReference. Unqualified NameReferences refer first to an item in the local SpaceSystem. If the name does not exist, then all SpaceSystems in the SpaceSystem tree above it will be searched to find a match.

b) If a NameReference consists of the name of the item and a path, it is a *qualified* NameReference; a qualified NameReference refers to an item specifically pointed to by the path. The path is formed using SpaceSystem names and some optional reserved special characters. The path form may be relative or absolute.

c) Using unqualified or relative qualified NameReferences is the recommended practice.

For qualified NameReferences, the path portion is formed using a Unix-like directory construct separated by '/' and allowing '.' and '..' special characters. The items between the separators are formed by using SpaceSystem names (from the @name of SpaceSystem). Per SpaceSystem, many NameReferences share a common namespace that is enforced by the 'keys' at the top level of the XTCE Schema. The keys are used to check for name uniqueness in certain major elements within a SpaceSystem. (See annex B for proper 'keys' enforced by the XTCE1.1 standard.)

### 3.3.2 SHARED NameReference NAMESPACES

The XTCE 1.2 schema defines keys to enforce uniqueness in certain area of the schema. In a sense these are namespace. Generally speaking, there are certain elements in XTCE that have a *name* attribute that is shared between CommandMetaData and TelemetryMetaData. The following share namespaces:

– TelemetryMetaData/ParameterSet and CommandMetaData/ParameterSet;

– TelemetryMetaData/ParameterTypeSet and CommandMetaData/ParameterTypeSet;

– TelemetryMetaData/ContainerSet and CommandMetaData/CommandContainerSet;

– TelemetryMetaData/ContainerSet, CommandMetaData/CommandContainerSet, and MetaCommand/CommandContainer.

### 3.3.3 SYNTAX FOR NameReferences

#### 3.3.3.1 General

NameReferences in XTCE are strings that adhere to the following syntax:

– Directory style ‘NameType’ is used, where ‘NameType’ is a restricted string with the pattern of ‘[^./:\[\] ]+’.

  – All name references use a Unix-like name referencing mechanism across the SpaceSystem tree where the ‘/’, ‘.’, and ‘..’ are used to navigate through the hierarchy.

  – The Name of the item of interest is restricted and follows the directory path.

  – SpaceSystem names are governed by NameTypes.

#### 3.3.3.2 ParameterInstanceReference Syntax Rules

ParameterInstanceReferences add the following two special syntax variations to NameReferences: field in an AggregrateType, and a cell in an Array:

{Optional Path}Name{Optional Field Reference or Optional Array Cell Reference}

– The Optional Field References is Aggregate Name followed by dot (‘.’) and then the aggregate field name.

– The Array Cell Reference is typical array square bracket (‘[ ]’) notation.

For ArrayInstanceReferences, the size of the dimension of an array parameter is set in the container. The array parameter should be used in several containers, with each container having the correct size for the dimensions.

## 3.4 COMMON XTCE ELEMENTS

### 3.4.1 OVERVIEW

This subsection will be referenced in other sections in this document as these XTCE elements and attributes reappear throughout the XTCE file. Again, this is done to reduce repetition in other parts of this report.

### 3.4.2 ELEMENTS OF THE NameDescriptionType

#### 3.4.2.1 General

The elements and attributes associated with the NameDescriptionType are identified in figure 3-1. The NameDescription schema type is the root type of many XTCE Schema types.

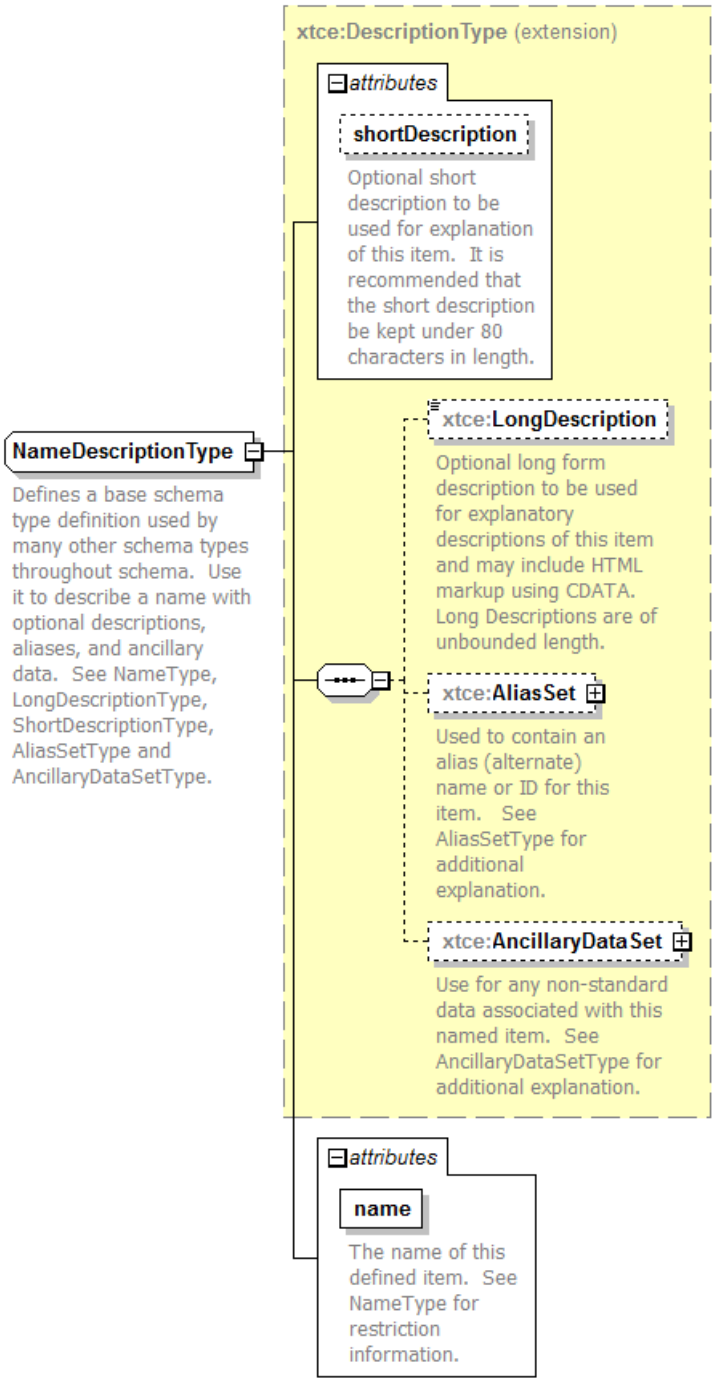



**Figure 3-1: NameDescription**

#### 3.4.2.2 name Attribute

The name attribute is mandatory or exception in the OptionalNamDescriptionType variation, where it is optional. If there is a name attribute, then that item will be referenced using a NameReference. For certain elements (such as calibrators) the name is optional.

```
<xtce:Parameter name="BatVolt1" parameterTypeRef="VoltageType"/>
```

#### 3.4.2.3 shortDescription Attribute

The shortDescription attribute is a summary description up to 80 characters long. This is not enforced.

```
<xtce:Parameter name="BatVolt1" parameterTypeRef="VoltageType" shortDescription="Battery
Voltage"/>
```

#### 3.4.2.4 LongDescription Element

The LongDescription element is used to provide for a more complete description of the item. HTML encapsulated in CDATA is allowed within the LongDescription element.

```
<xtce:Parameter name="BatVolt1" parameterTypeRef="VoltageType" shortDescription="Battery
Voltage">
    <xtce:LongDescription>Battery Voltage from EPS-Subsystem,
    EPSA_BAT_S7</xtce:LongDescription>
</xtce:Parameter>
```

#### 3.4.2.5 AliasSet Element

Organizations may have more than one name for a given item: an official name, and another that is in common use, that is, an alias. Use of the optional AliasSet element allows definition of aliases.

```
<xtce:Parameter name="BatVolt1" parameterTypeRef="VoltageType" shortDescription="Battery
Voltage">
    <xtce:LongDescription>Battery Voltage from EPS-Subsystem, subsystem 7</xtce:LongDescription>
    <xtce:AliasSet>
        <xtce:Alias nameSpace="Display" alias="EABVS7_1"/>
    </xtce:AliasSet>
</xtce:Parameter>
```

The example above is representative of the shorthand naming conventions often used in mission operations. Here, an alias is created with an unreadable naming convention.

NOTE – Aliases are not visible to NameReferences.

#### 3.4.2.6 AncillaryDataSet Element

AncillaryData is used to define tag/value associations for user-defined and interpreted items.

```
<xtce:Parameter name="BatVolt1" parameterTypeRef="VoltageType" shortDescription="Battery
Voltage">
    <xtce:LongDescription>Battery Voltage from EPS-Subsystem, subsystem 7</xtce:LongDescription>
    <xtce:AliasSet>
        <xtce:Alias nameSpace=" Display" alias=" EABVS7_1"/>
    </xtce:AliasSet>
    <xtce:AncillaryDataSet>
        <xtce:AncillaryData name="LRVSTK">true, max: 3</xtce:AncillaryData>
    </xtce:AncillaryDataSet>
</xtce:Parameter>
```

In this example, a system-dependent flag is set to a value that makes sense only for that implementation. AncillaryData can have a MIME type associated with it.

AncillaryData is useful for defining specialty flags or items that are not well represented in XTCE. Since they are not in the schema, care should be taken in defining their intended meaning.

### 3.4.3 COMPARISON ELEMENTS OF COMPARISONTYPE

#### 3.4.3.1 General

XTCE's comparison (expression) elements include Comparison, ComparisonList, Boolean Expression, and Custom Algorithm. These elements are used in alarms (limits) calibrators and RestrictionCriteria. They are also used in other locations that require the evaluation of an expression formed by ParameterInstanceRefs and a value.

#### 3.4.3.2 Comparison Element

##### 3.4.3.2.1 General

The Comparison element compares a single parameter instance against a supplied value using a comparison operator. Figure 3-2 shows the Comparison element.

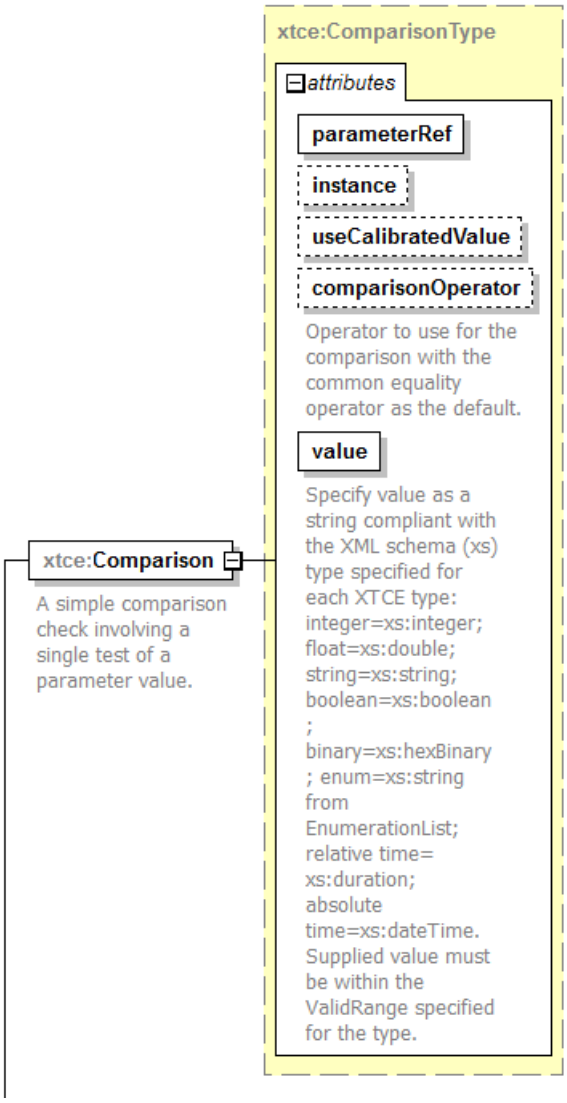



**Figure 3-2: Comparison Element**

#### 3.4.3.2.2 parameterRef Attribute

The parameterRef attribute refers to the parameter's description in ParameterSet. If the comparison is a container area, it refers to the container entries first.

#### 3.4.3.2.3 instance Attribute

The instance attribute may be specified. The instance defaults to zero or the last value. The most recent instance is referred to as the Last Reported Value (LRV).

#### 3.4.3.2.4 useCalibratedValue Attribute

The useCalibratedValue attribute determines if a calibrated value is used in the comparison. The useCalibratedValue defaults to ‘true’.

#### 3.4.3.2.5 comparisonOperator Attribute

The comparisonOperator attribute is a list of standard operators: ==, <, <=, >, >=, and !=. The default comparisonOperator is ==.

– The following example uses the defaults:

```
<xtce:Comparison parameterRef="Simple" value="100"/>
```

– The following is a more complex version that specifies all values:

```
<xtce:Comparison parameterRef="Complicated" value="99.9919"
comparisonOperator="&gt;=" instance="1" useCalibratedValue="false"/>
```

#### 3.4.3.2.6 value Attribute

The value attribute is a string. The comparison may be against a ParameterType that represents a different data type (such as an integer). Depending on the scenario, the string may need to be converted to a different data type in order to process the comparison:

a) For telemetry parameters:

   1) If the useCalibratedValue is ‘false’, then the value should be converted to the ParameterType’s DataEncoding data type: string, integer, float, or Boolean.

   2) If the flag is ‘true’, then the ParameterType’s data type should be used.

b) For command parameters:

   1) If the useCalibratedValue is ‘false’, then the data type represented by the ParameterType is used.

   2) Otherwise, the data type represented by the DataEncoding is used.

Listing the value’s legal formats will help make exchanging with other formats easier. The following are recommendations:

a) Numerical integer values use xsd:integer, xtce:HexadecimalType, xtce:OctalType, and xtce:BinaryType rules.

   – For BinaryType, ‘0x’ and ‘0o’ and ‘0b’ are placed before the value.

   – For FloatParameterType, xsd:float or xsd:double is used, depending on the precision.

b) For Boolean, xsd:boolean is used, and for enumerations one of the valid labels for the ParameterType is specified.

   – In the case of RelativeTimeParameterType and AbsoluteTimeParameterType values, xsd:duration and xsd:dateTime are used. Specifying the time zone is recommended.

### 3.4.3.3 ComparisonList Element

ComparisonList is simply a list of Comparison elements. All Comparisons in the ComparisonList must be 'true' in order for ComparisonList to be 'true'.

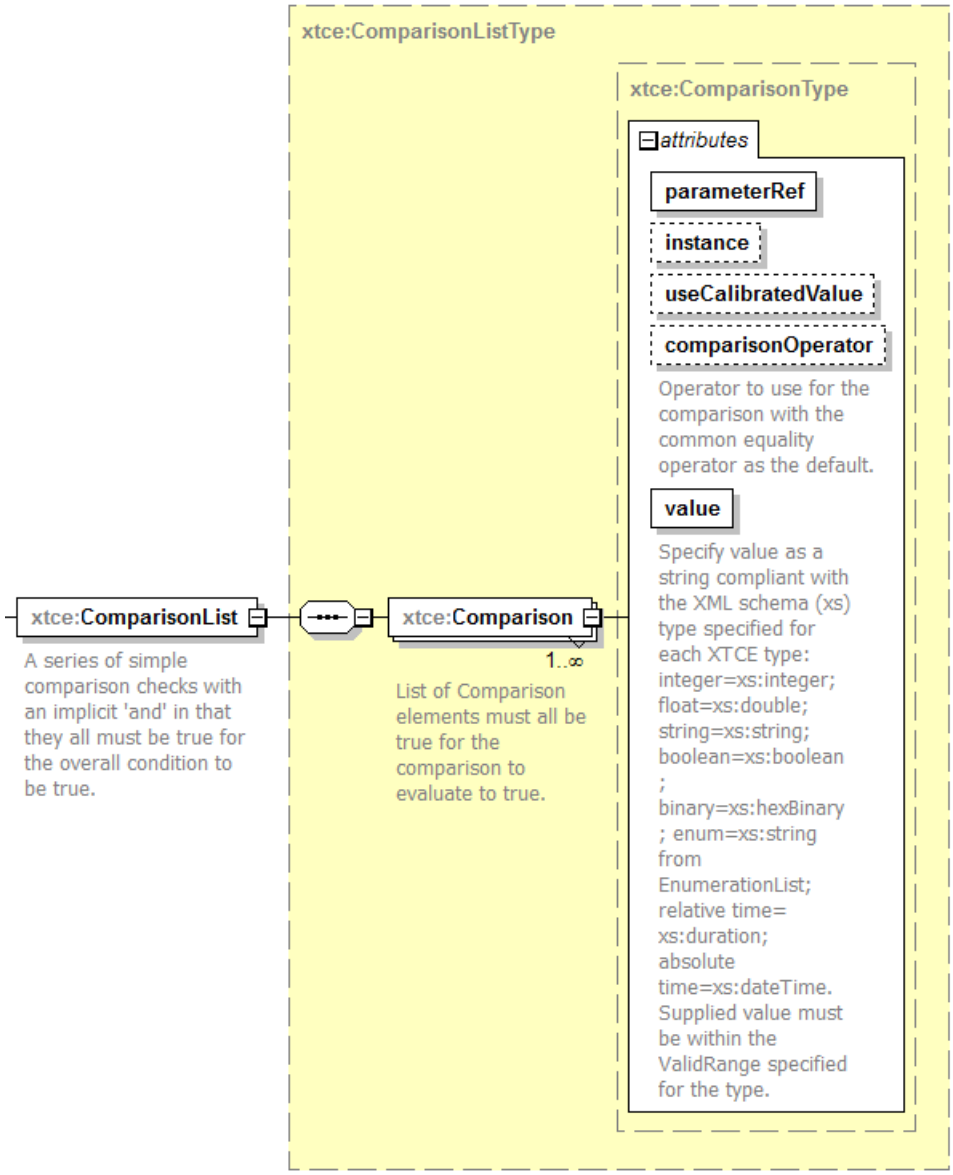



**Figure 3-3: ComparisonList**

#### 3.4.3.4 Boolean Expression Element

##### 3.4.3.4.1 General

BooleanExpression is used to construct complex ‘and/or’ Boolean expressions.

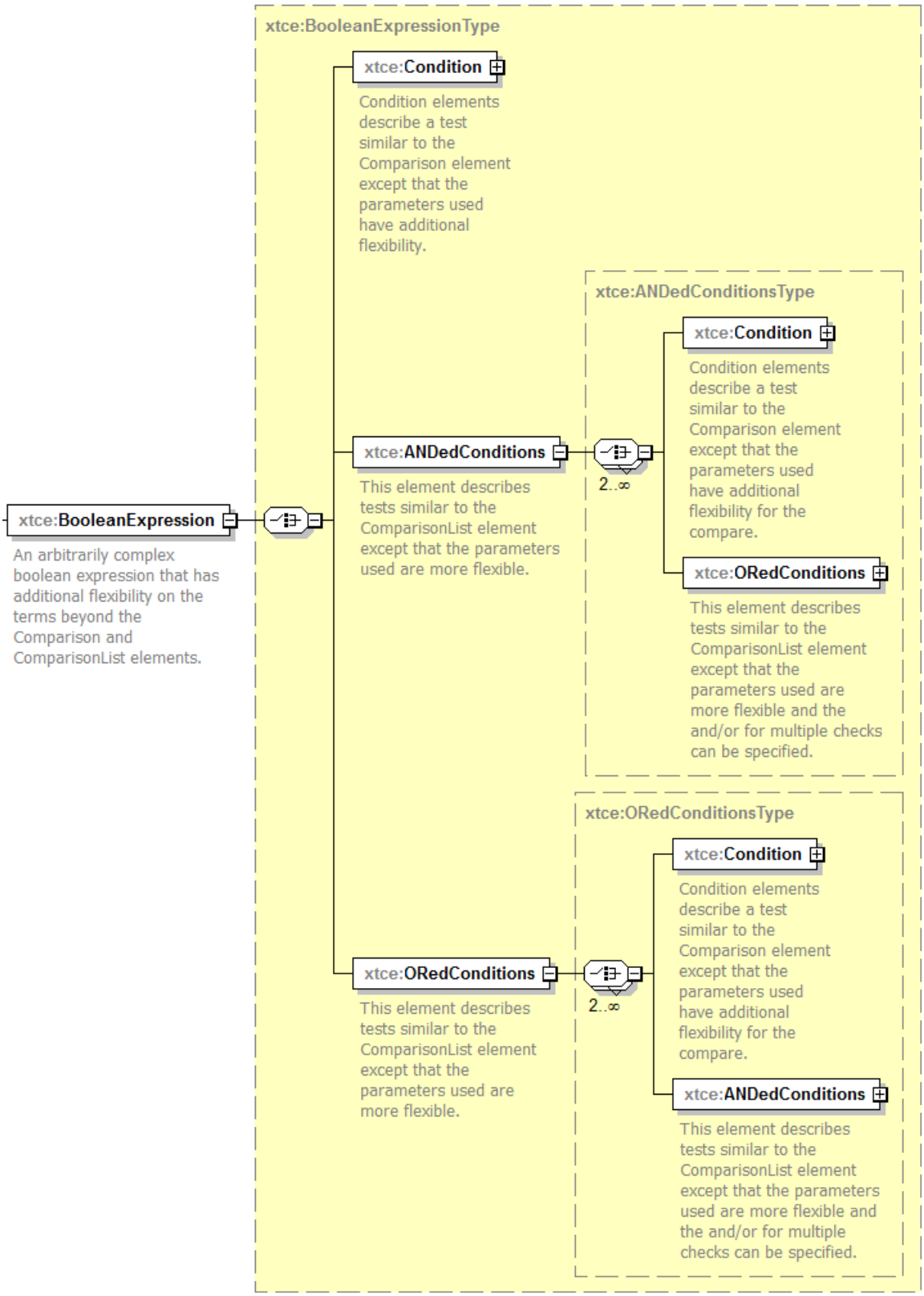



**Figure 3-4: BooleanExpression**

The Condition element here is similar to the Comparison element. Values or other parameters may be used on both sides of the comparison operator. Example of a BooleanExpression:

(ParameterAndValue == 100) AND (ParameterAndParameter >= TheOtherParameter) AND ((P2 <= P3) OR (P4 != 99))

```
<xtce:BooleanExpression>
    <xtce:ANDedConditions>
        <xtce:Condition>
            <xtce:ParameterInstanceRef parameterRef="ParameterAndValue"/>
            <xtce:ComparisonOperator>==</xtce:ComparisonOperator>
            <xtce:Value>100</xtce:Value>
        </xtce:Condition>
        <xtce:Condition>
            <xtce:ParameterInstanceRef parameterRef="ParameterAndParameter"/>
            <xtce:ComparisonOperator>&gt;=</xtce:ComparisonOperator>
            <xtce:ParameterInstanceRef parameterRef="TheOtherParameter"/>
        </xtce:Condition>
        <xtce:ORedConditions>
            <xtce:Condition>
                <xtce:ParameterInstanceRef parameterRef="P2"/>
                <xtce:ComparisonOperator>&lt;=</xtce:ComparisonOperator>
                <xtce:ParameterInstanceRef parameterRef="P3"/>
            </xtce:Condition>
            <xtce:Condition>
                <xtce:ParameterInstanceRef parameterRef="P4"/>
                <xtce:ComparisonOperator>!=</xtce:ComparisonOperator>
                <xtce:Value>99</xtce:Value>
            </xtce:Condition>
        </xtce:ORedConditions>
    </xtce:ANDedConditions>
</xtce:BooleanExpression>
```

#### 3.4.3.4.2 Condition Element

Condition is a child element of BooleanExpression. Condition is similar to Comparison, but differs in that it allows one to provide two parameter instances or a value. Condition's ComparisonOperator element must be explicitly set since there is no default value for it.

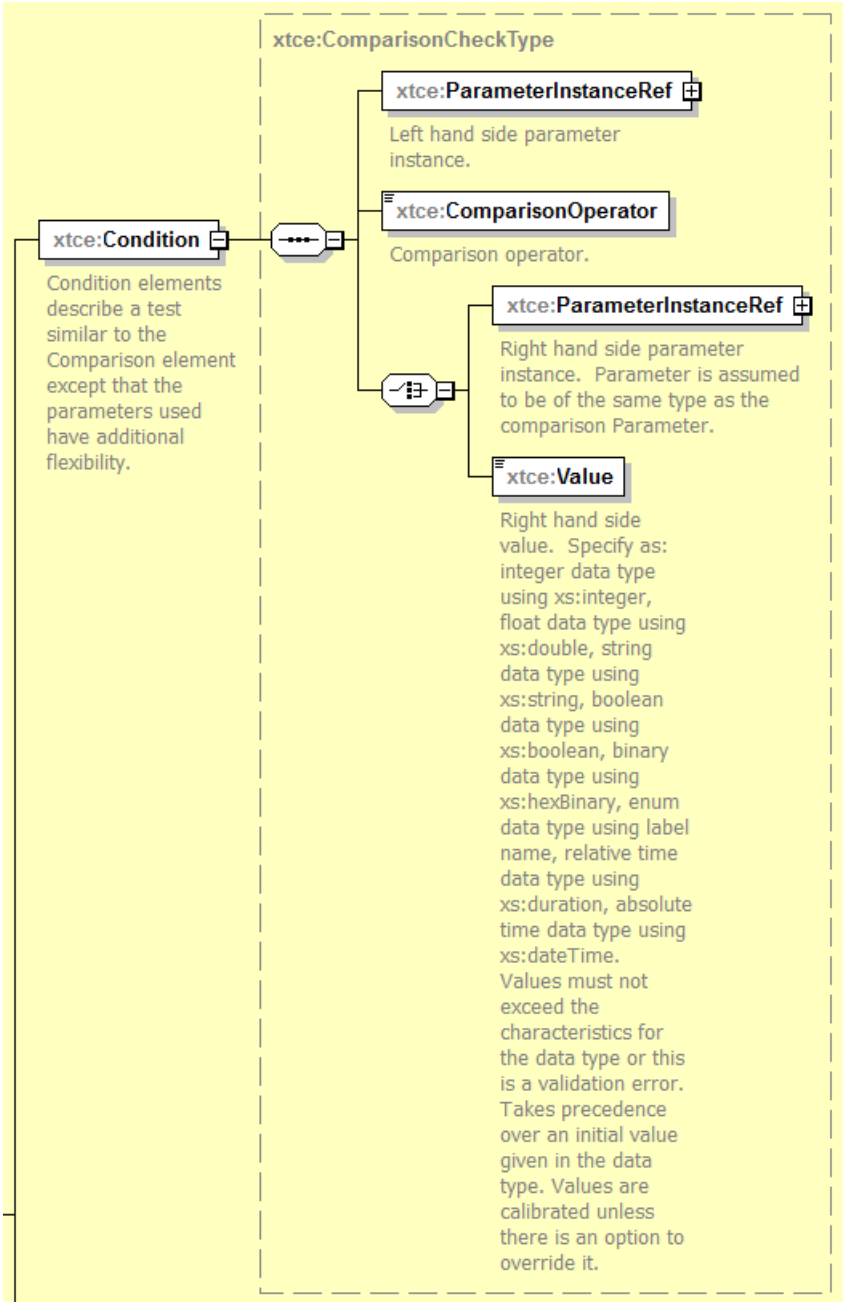



**Figure 3-5: Condition**

### 3.4.3.5 CustomAlgorithm Element

CustomAlgorithm allows for the specification of an algorithm that can be processed to produce a result. This element is used in a number of locations throughout XTCE. It is often specified in a construct along with Comparison, ComparisonList, or BooleanExpression. It is used to represent processing that cannot be done with those elements.

This item starts as a NameDescription and includes a list of parameters and constants. There is an optional sub-element that allows the algorithm code to be transferred as text. CustomAlgorithm is an InputAlgorithmType and related to AlgorithmSet.

NOTE – It appears annotation was lost from XTCE 1.1; however, that annotation still applies to this element in XTCE 1.2.

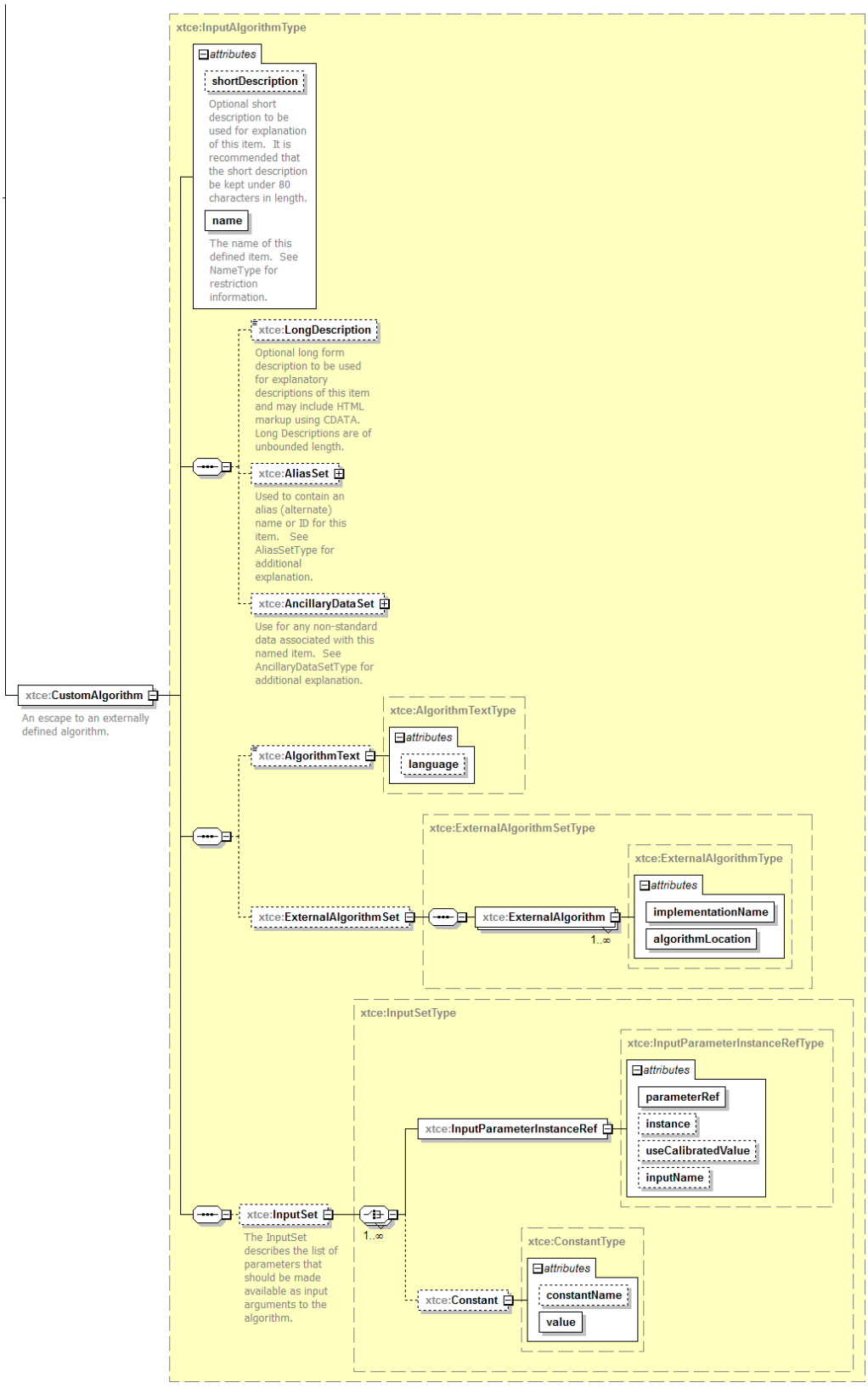



**Figure 3-6: CustomAlgorithm Element**

The following is a simple example of the type of information that could be held in CustomAlgorithm. Various options allow for additional flexibility.

```
<xtce:ContextMatch>
    <xtce:CustomAlgorithm name="SimpleRemoteAlgorithm">
        <xtce:ExternalAlgorithmSet>
            <xtce:ExternalAlgorithm algorithmLocation=http://MyMission/Algorithms/CalcOrbit
             implementationName="Java"/>
        </xtce:ExternalAlgorithmSet>
    </xtce:CustomAlgorithm>
</xtce:ContextMatch>
```

### 3.4.3.6 MatchCriteriaType Element

MatchCriteria is an XTCE Schema type that groups expression elements. This schema type is used in many locations in XTCE.

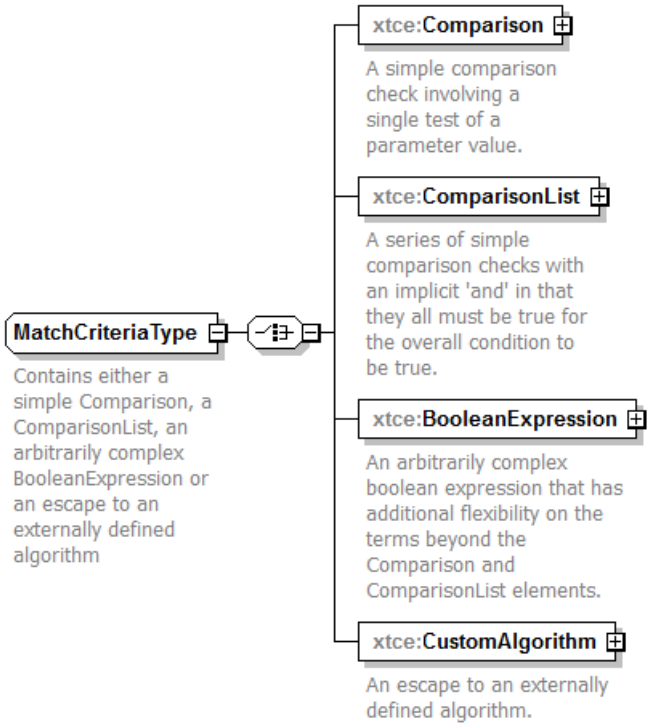



**Figure 3-7: MatchCriteria**

### 3.4.3.7 ContextMatch Element

Contexts are user defined and describe when alarms or calibrations are active. A context may be used by missions to enable or disable certain functionality based on the evaluation of an expression. Specific contexts for missions are used by ContextAlarms and ContextCalibrators. The context is defined in a ContextMatch element.

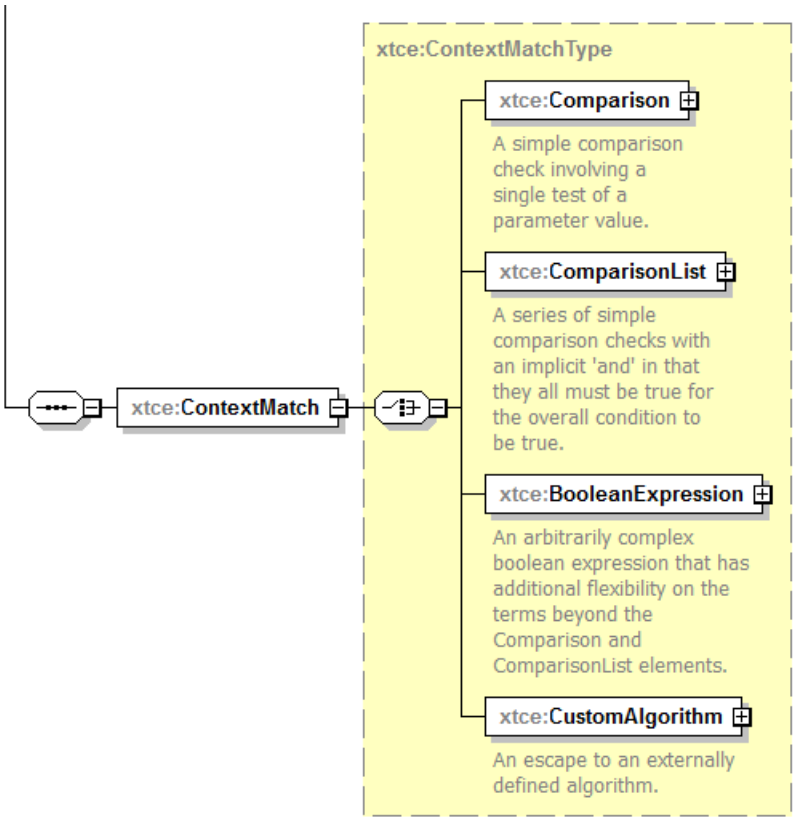



**Figure 3-8: ContextMatch**

Contexts could be defined for mission phases. One approach would be to create a session variable of enumerated mission phases and then use these as comparisons in the contexts of interest.

### 3.4.4 PARAMETERINSTANCEREFERENCES

A ParameterInstanceRef is a NameReference to the named parameter's value during command and telemetry processing. It is also a reference to its definition in the XTCE document.

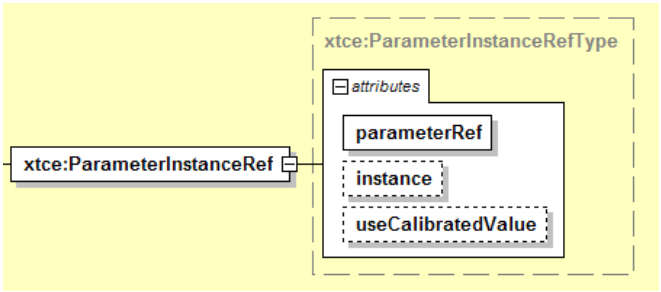



**Figure 3-9: ParameterInstanceRef**

The value may be defined in a conceptual value table or in a container. The container takes precedence if the ParameterInstanceRef is defined in that area.

– The parameterRef attribute refers to its description in the XTCE document.

– The instance is the version of the decommutated value to use. An instance of zero means the last decommutated value of the named Parameter (Last Recorded Value). Negative instances reference recorded values in the past (back in time). The default value of instance is zero.

– The useCalibratedValue attribute specifies whether the raw or calibrated value is to be used. The default value of useCalibratedValue is 'true'.

Positive instances can be specified (representing recorded values forward in time), but this is not recommended.

### 3.4.5 SIZE-RELATED ELEMENTS

#### 3.4.5.1 General

Several elements, such as the SizeInBits element, are used throughout the schema when a size needs to be specified. The elements related to size are in the XTCE Schema type called IntegerValueType.

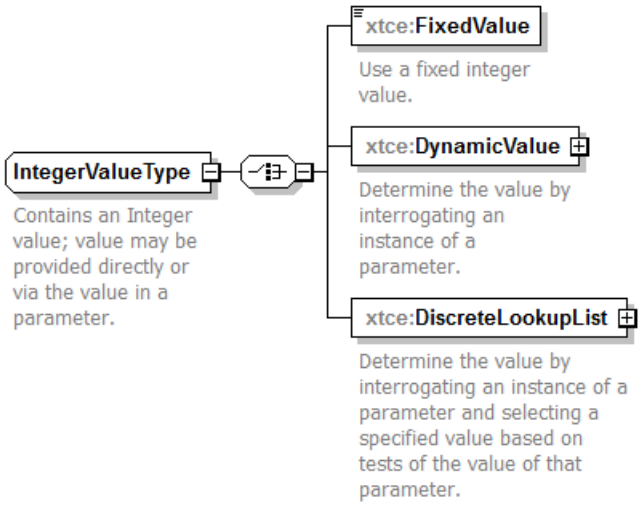



**Figure 3-10: IntegerValueType**

NOTE – There is a similar type used on the CommandMetaData side of XTCE called ArgumentIntegerValueType. The principle difference is that it supports both ParameterInstanceRef and ArgumentInstanceRef; this is shown below.

### 3.4.5.2 FixedValue Element

The FixedValue element specifies the exact size as an integer.

```
<xtce:SizeInBits>
    <xtce:FixedValue>11</xtce:FixedValue>
</xtce:SizeInBits>
```

### 3.4.5.3 DynamicValue Element

The DynamicValue element uses a ParameterInstanceRef to look up the size in the named parameter value table. The value may be adjusted using the slope and intercept of a line. Its instance and calibrated designation may also be specified (defaults are provided).

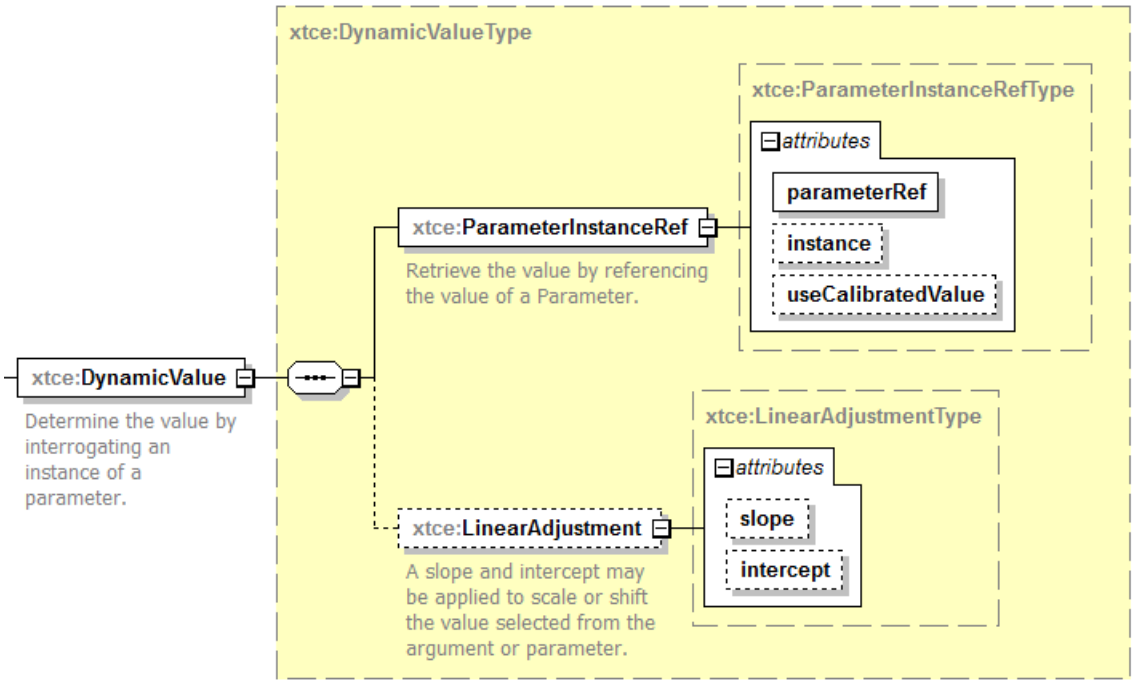



**Figure 3-11: DynamicValue**

The following example looks up instance zero of 'SizeFromThisParameter', then multiplies it by 8, and then adds 25.

```
<xtce:DynamicValue>
    <xtce:ParameterInstanceRef parameterRef="SizeFromThisParameter"/>
    <xtce:LinearAdjustment intercept="25" slope="8"/>
</xtce:DynamicValue>
```

The following shows DynamicValue as it is modified for CommandMetaData.

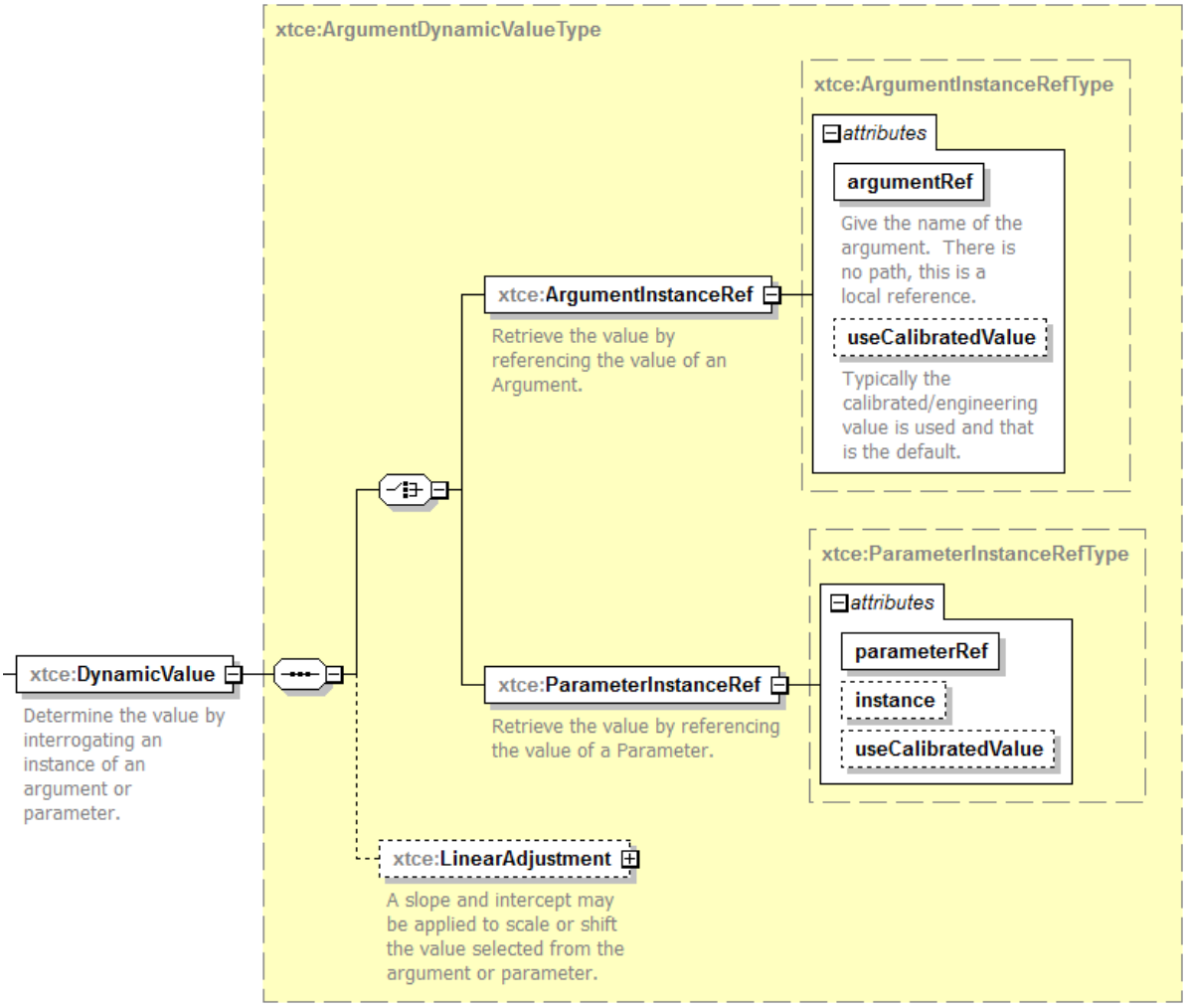



**Figure 3-12: ArgumentDynamicValue**

As can be seen in figure 3-12 above, the DynamicValue in CommandMetaData allows for either an ArgumentInstanceRef or a ParameterInstanceRef.

### 3.4.5.4 DiscreteLookupList Element

DiscreteLookupList is a list of the element DiscreteLookup, which contains a condition and associated value. The value from the first true condition is returned. The DiscreteLookup is constructed from a MatchCriteria (see 3.4.3.6), although now it is wrapped in its own schema types, as well as the DiscreteLookupListType itself.

NOTE – There is also a CommandMetaData side specific one with similar modifications as described in DynamicValue above.

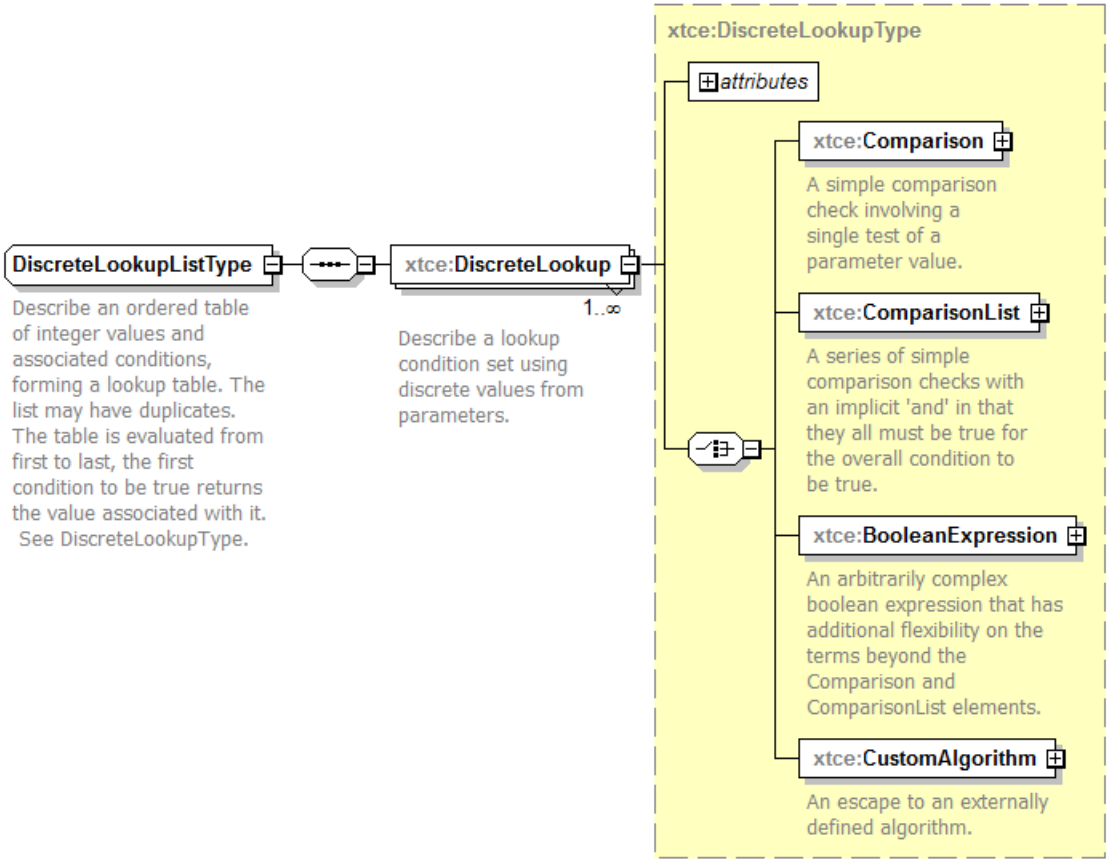



**Figure 3-13: DiscreteLookup**

A large variety of constructions are possible with this element. The example below simulates checking the value of a single parameter, which results in different returned sizes. Here, the value of '10', '25', or '711' will be returned, depending on which condition is true.

```
<xtce:DiscreteLookupList>
    <xtce:DiscreteLookup value="10">
        <xtce:Comparison parameterRef="P1" value="1"/>
    </xtce:DiscreteLookup>
    <xtce:DiscreteLookup value="25">
        <xtce:Comparison parameterRef="P1" value="2"/>
    </xtce:DiscreteLookup>
    <xtce:DiscreteLookup value="711">
        <xtce:Comparison parameterRef="P1" value="3"/>
    </xtce:DiscreteLookup>
</xtce:DiscreteLookupList>
```

Here is the same construction for the CommandMetaData side of XTCE; it calls both the ParameterInstanceRef and ArgumentInstanceRef.

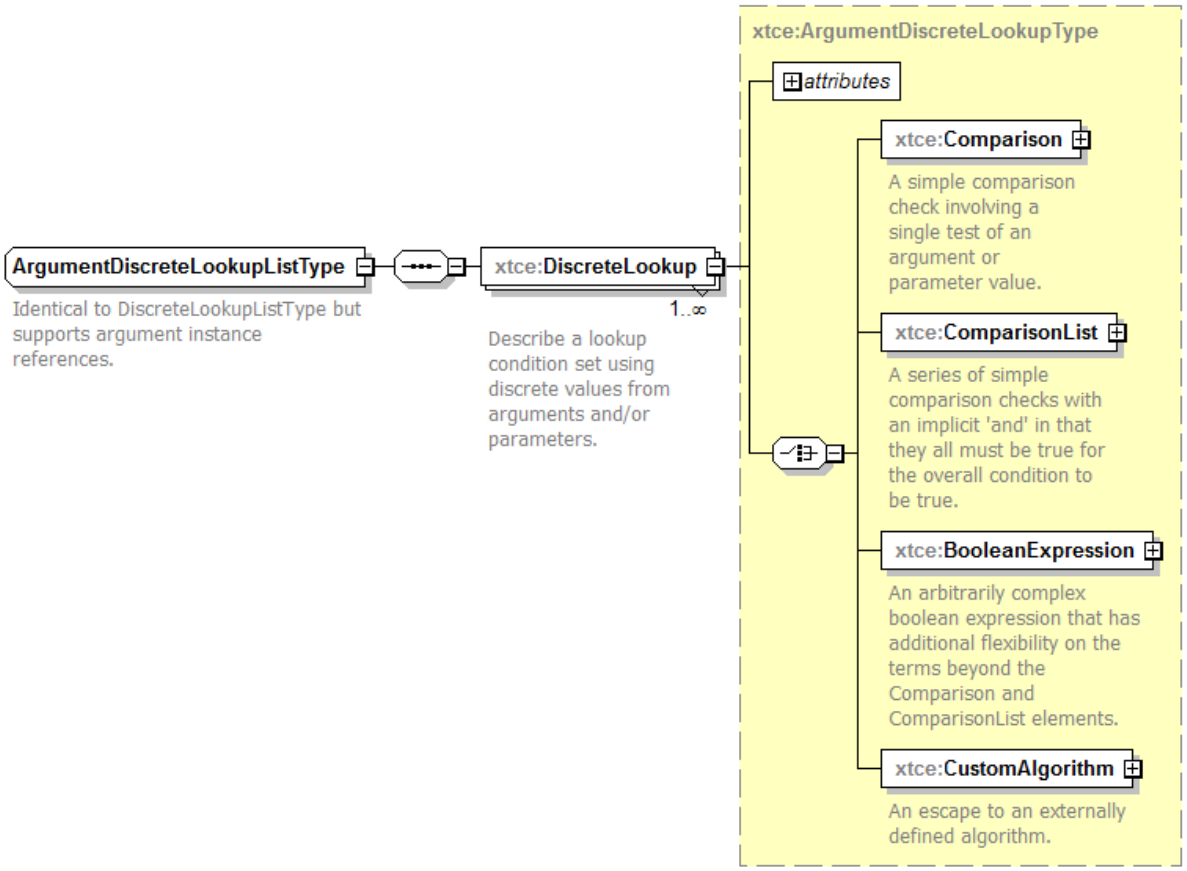



**Figure 3-14: ArgumentDiscreteLookupList**

## 3.5 MODIFIED COMMANDMETADATA SCHEMA TYPES

As has been eluded to above, many CommandMetaData side schema types have been modified slightly in XTCE 1.2 to allow for either a ParameterInstanceRef or ArgumentInstanceRef. The overall effect is the command side has become more independent from XTCE 1.1. The following schema types are affected, and in the annotation, there is usually something of this nature: 'Identifical to X element but supports argument instance references'. All or nearly all have the word 'Argument' prefixed to their previous name. They are listed here as follows:

- ArgumentArgumentRefEntryType;
- ArgumentArrayArgumentRefEntryType;
- ArgumentArrayParameterRefEntryType;
- ArgumentContainerSegmentRefEntryType;
- ArgumentFixedValueEntryType;
- ArgumentIndirectParameterRefEntryType;

- ArgumentParameterRefEntryType;
- ArgumentParameterSegmentRefEntryType;
- ArgumentStreamSegmentRefEntryType;
- ArgumentLocationInContainerInBitsType;
- ArgumentSequenceEntryType;
- ArgumentComparisonType;
- ArgumentComparisonCheckType;
- ArgumentComparisonListType;
- ArgumentDiscreteLookupType;
- ArgumentDiscreteLookupListType;
- ArgumentDynamicValueType;
- ArgumentInputAlgorithmType;
- ArgumentInputSetType;
- ArgumentInstanceRefType;
- ArgumentBooleanExpressionType;
- ArgumentANDedConditionsType;
- ArgumentORedConditionsType;
- ArgumentMatchCriteriaType;
- ArgumentBaseDataType;
- ArgumentBaseTimeDataType;
- ArgumentBooleanDataType;
- ArgumentDimensionType;
- ArgumentDimensionListType;
- ArgumentEnumeratedDataType;
- ArgumentFloatDataType;
- ArgumentIntegerDataType;
- ArgumentStringDataType;
- ArgumentBinaryDataEncodingType;

- ArgumentVariableStringType;
- ArgumentStringDataEncodingType;
- ArgumentIntegerValueType;
- ArgumentRepeatType.

## 3.6 NEW AND MODIFIED SCHEMA TYPES

In XTCE 1.2, there has been major creation of new and rework of many existing schema types overall, not just the items listed above related to ArgumentInstanceRef support. The overall purpose was to clean up the mapping in popular software technologies such as JAXB so that, in essence, every schema type becomes a top-level class. Consequently, in transitioning from XTCE 1.1, developers will probably confront the largest changes, while end users will see limited change to the overall syntax.

# 4 XTCE ELEMENTS AND ATTRIBUTES

## 4.1 OVERVIEW

This section describes the remaining XTCE element and attributes. It explains the function of each and in many cases provides an example. It refers to the sections above as needed.

## 4.2 THE SPACESYSTEM ELEMENT

### 4.2.1 GENERAL

The SpaceSystem element forms the root of an XTCE instance document (file). It contains elements for describing telemetry and command descriptions. It also describes attributes and elements associated with documentation, document management, and services.

A SpaceSystem may have child SpaceSystems, the resulting structure forming a hierarchical tree definition. The meaning of the SpaceSystem is left to the user.

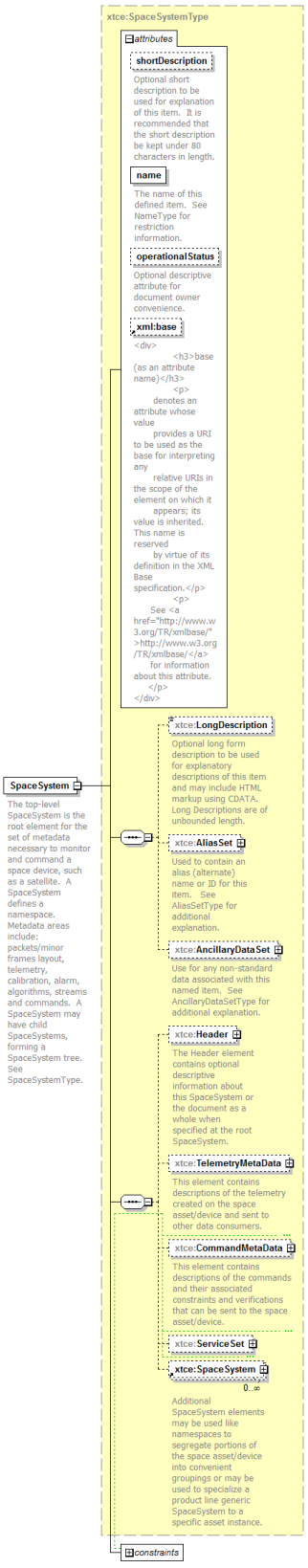



**Figure 4-1: SpaceSystem Root Element**

### 4.2.2 ENCODING

All valid XML documents must have a version/encoding element at the top of this form:

<?xml version="1.0" encoding="UTF-8"?>

The encoding is specified before the document root element (i.e., before the first SpaceSystem element).

### 4.2.3 SCHEMA XMLNS:XTCE

The top level schema element set the namespace to ‘xtce’ and other features.

‘http://www.omg.org/spec/XTCE/20180204’

### 4.2.4 IMPORT XML.XSD

In XTCE1.2, the next element after schema is an import element for the XML.XSD that is used for supporting xinclude.

### 4.2.5 XML:BASE ATTRIBUTE

Also new in XTCE 1.2, XML:BASE ATTRIBUTE is also used for xinclude support.

NOTE – When creating instance documents, consider that XML tools will automatically load the document in question from the URL. If this is not acceptable to an organization, the official XTCE Schema should be placed in a well-known location in its file system.

a) Although the exact name of the schema may be important to an organization, it is not important in terms of validation.

b) Many users may find it easier to put the schema location in a relative path such as:

‘SpaceSystem.xsd’

c) Another popular XTCE Schema location designation has been:

‘https://www.omg.org/spec/XTCE/20180204/SpaceSystem.xsd’

### 4.2.6 NAMEDESCRIPTION ATTRIBUTES AND ELEMENTS

Subsection 3.4.2 contains a description of the elements and attributes associated with this element: name, shortDescription, LongDescription, AliasSet, and AncillaryDataSet.

### 4.2.7 HEADER

#### 4.2.7.1 General

The SpaceSystem Header element contains several documentation-oriented areas important to missions.

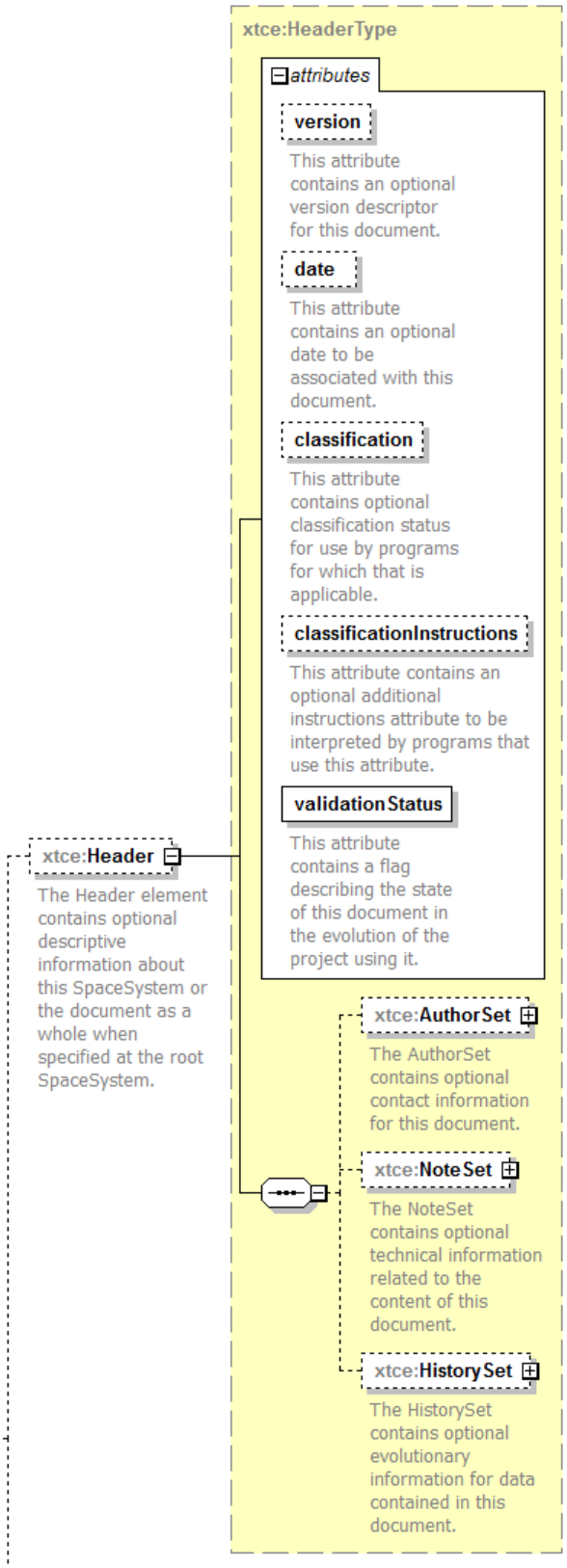



**Figure 4-2: Header**

#### 4.2.7.2 version Attribute

The version attribute indicates the version of the XTCE document. It is left to the end user to define the format.

#### 4.2.7.3 date Attribute

The date attribute indicates the date the XTCE document was created. The date attribute's format is left up to the end user to define.

#### 4.2.7.4 classification Attribute

The classification attribute indicates the XTCE document classification (normal, contingency, special operations, etc.). The content of the classification attribute is left to the end user.

#### 4.2.7.5 classificationInstructions Attribute

Based on the classification of the XTCE document special instructions might be required. The content of the classificationInstructions attribute is left up to the end user.

#### 4.2.7.6 validationStatus Attribute

The validationStatus attribute indicates the XTCE document validation status (successful, failed, etc.). The content of the validationStatus attribute is left up to the end user.

#### 4.2.7.7 AuthorSet Element

Authors of the document are placed within the AuthorSet element. The content of the Author element itself is left up to the end user.

#### 4.2.7.8 NoteSet Element

The NoteSet element allows capturing of notes associated with the file. The NoteSet contents are set at the discretion of the end user.

#### 4.2.7.9 HistorySet Element

The HistorySet element allows the capture of document creation history. It is set at the discretion of the end user.

#### 4.2.7.10 Child SpaceSystem—SpaceSystem Hierarchy

A SpaceSystem may have one or more child SpaceSystems forming a hierarchy of SpaceSystem elements. Any number of levels may be created to form a tree representation for an entire project as it meets the needs of the user.

A SpaceSystem tree may map nicely to various views of the project, but it adds complexity to the XTCE document processing (especially NameReferences). It is recommended to use just one SpaceSystem since it simplifies the NameReferences.

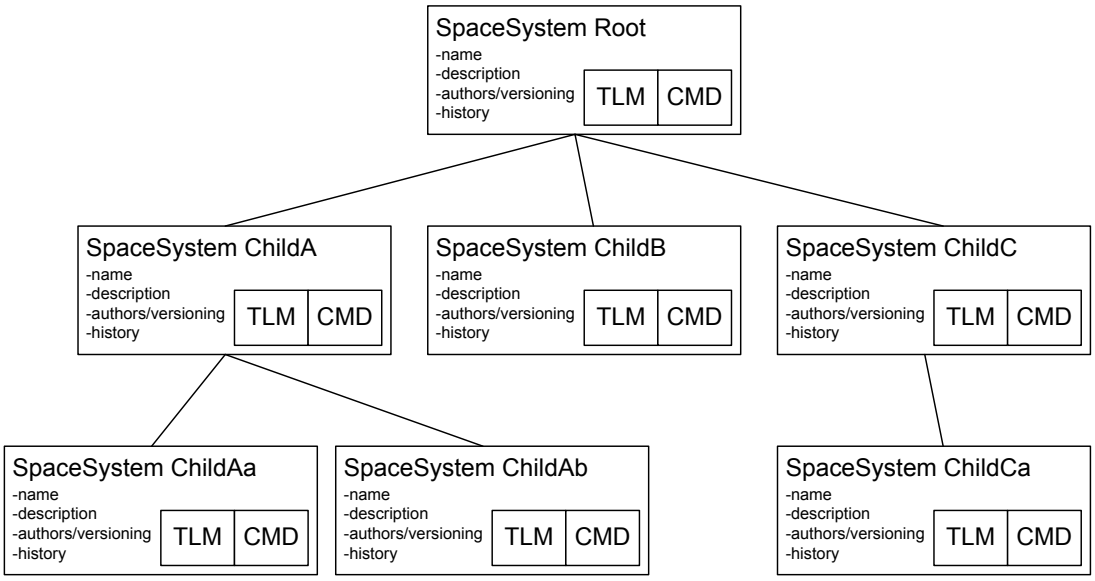



**Figure 4-3: SpaceSystem Tree Diagram**

```
<xtce:SpaceSystem name="Root">
    <xtce:SpaceSystem name="ChildA">
        <xtce:SpaceSystem name="ChildAa"/>
        <xtce:SpaceSystem name="ChildAb"/>
    </xtce:SpaceSystem>
    <xtce:SpaceSystem name="ChildB"/>
    <xtce:SpaceSystem name="ChildC">
        <xtce:SpaceSystem name="ChildCa"/>
    </xtce:SpaceSystem>
</xtce:SpaceSystem>
```

NOTE – For the purposes of brevity, in the above example, the various child elements and attributes are not being shown.

## 4.3 TELEMETRYMETADATA—TELEMETRY

### 4.3.1 GENERAL

Telemetry description information is captured in the TelemetryMetaData area of SpaceSystem.

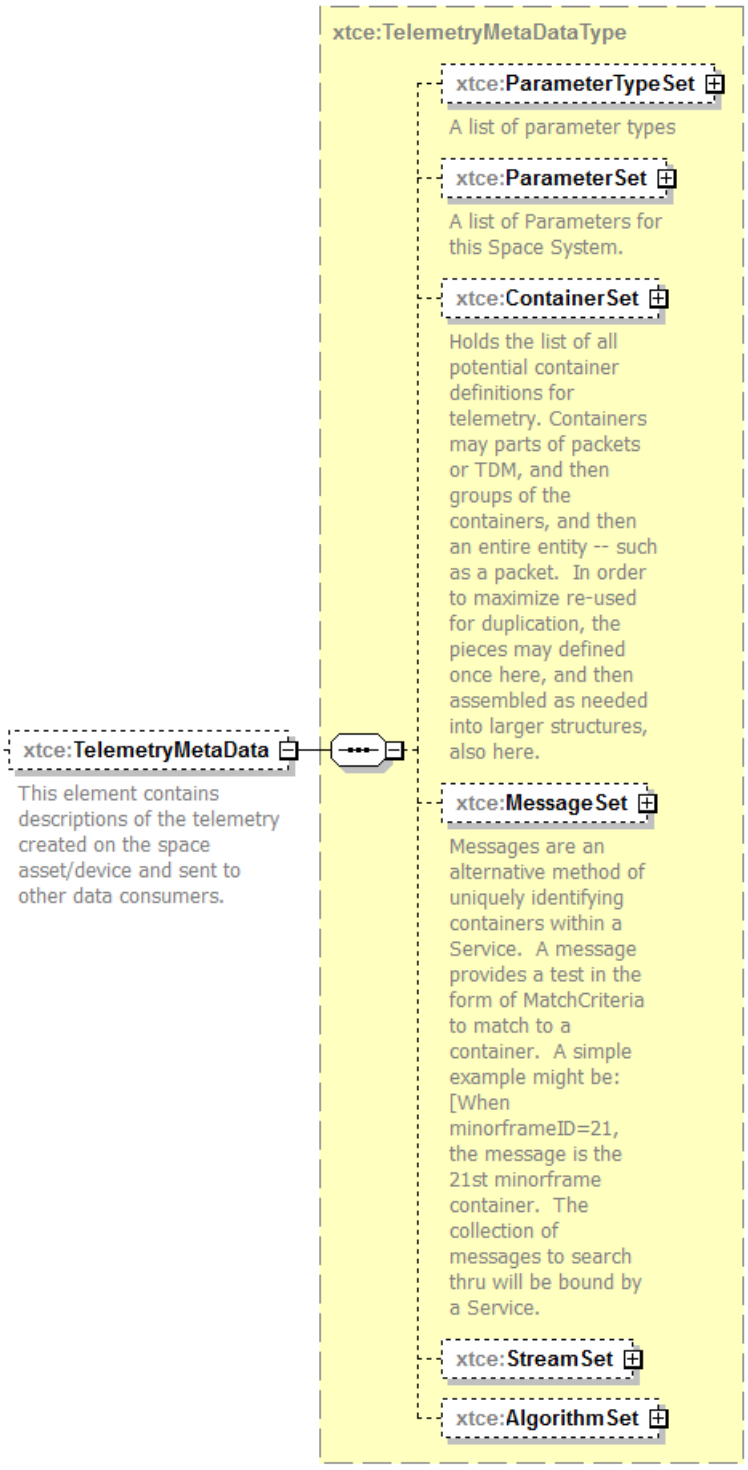



**Figure 4-4: TelemetryMetaData**

TelemetryMetaData contains elements for ParameterTypes, Parameters, Containers, Messages, Streams, and Algorithms. With these elements, one can create descriptions for telemetry and how it is packaged (e.g., a CCSDS packet).

Telemetry is described in XTCE using the following forms:

- Uncalibrated Parameters;
- Calibrated Parameters;
- System-Supplied Parameters;
- Derived Parameters.

'Parameters' refers to individual telemetry values (e.g., a mnemonic value) sent from the remote system (spacecraft) to the ground. These telemetred values have a data type that describes how that value is encoded. This is called ParameterType in XTCE, and each Parameter has one of them. XTCE's ParameterTypes goes somewhat further and describes both the source data type (the format of the information from the spacecraft) and (likely) destination data type after ground conversion or calibration. (In the case of calibrated parameters, this is a polynomial or linear function.)

The destination data type is defined by the particular ParamterType, for example, IntegerParameterType. And the source data type is captured within it as one of the DataEncodings (for example IntegerDataEncoding). These two together then form the complete XTCE 'data type' definition for a Parameter, and there are many additional attributes and elements within this construct: calibrators, alarms, valid range, and various other properties.

Containers are used to group parameters into common blocks that can be used to describe abstract structures (formats) like the CCSDS Packet format, ESA PUS, or minor frames.

### 4.3.2 PARAMETERTYPESET—PARAMETERTYPES

#### 4.3.2.1 General

ParameterTypes are defined in ParameterTypeSet. Each Parameter has a ParameterType through a NameReference. ParameterTypes contain the bulk of the descriptive information related to individual telemetry items.

Many of the ParameterTypes are similar to each other since they are constructed using the same schema types and follow a similar pattern. Most ParameterTypes consist of the area's documentation, encoding, and alarm. Within the encoding area is the calibration area. The overall pattern is introduced in this section. Follow-on sections show the patterns for each individual ParameterType.

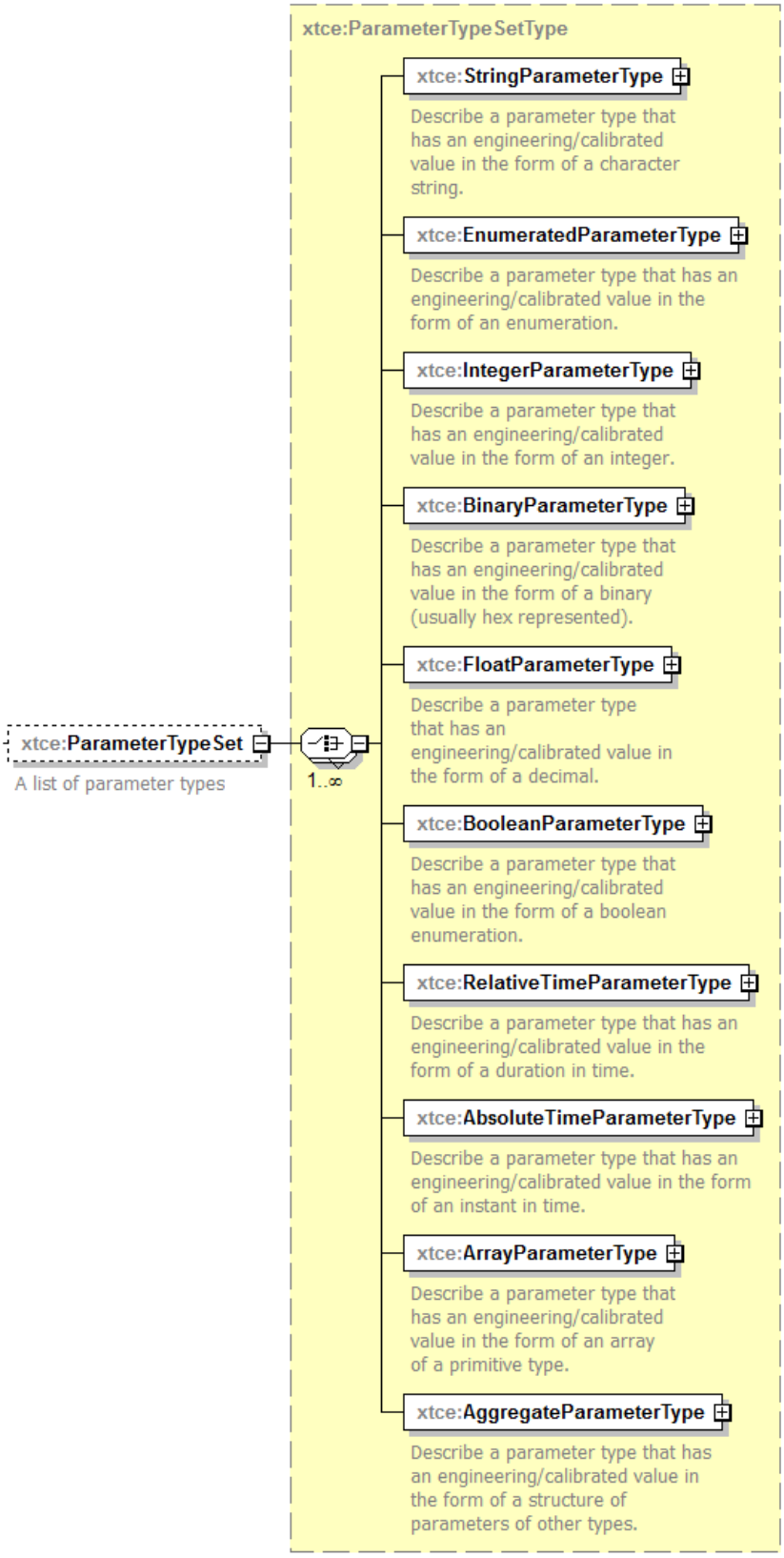



**Figure 4-5: ParameterTypes**

#### 4.3.2.2 ParameterType Pattern

##### 4.3.2.2.1 General

The basic ParameterType pattern is shown below using the FloatParameterType. It has become larger in XTCE 1.2 because of the substantial increase in annotation. The resulting diagrams are split into four parts to better fit the page.

The first part, shown below, contains the name, descriptive items, the alias and ancillary elements, and several ParameterType specific attributes.

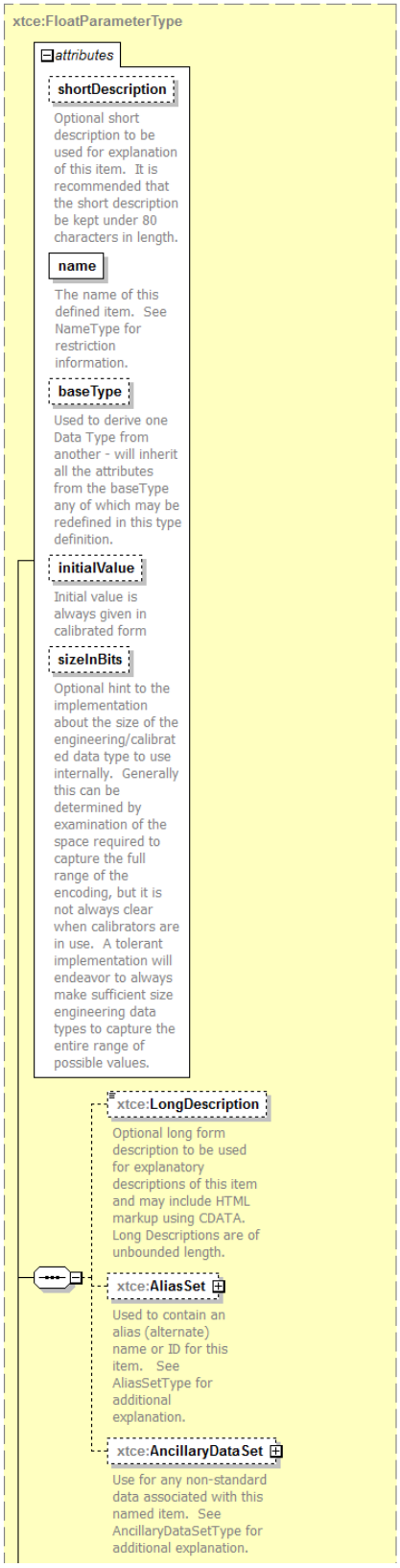



**Figure 4-6: FloatParameterType—Part 1**

The second part of the syntax includes all the DataEncoding choices.

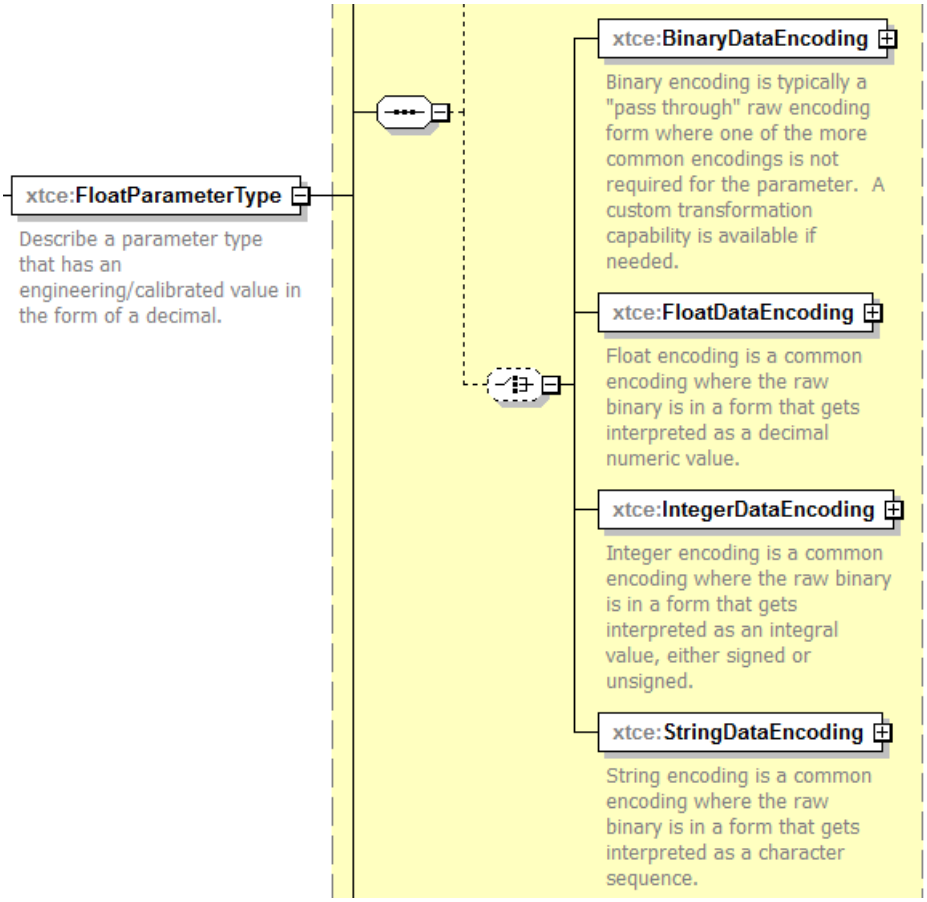



**Figure 4-7: FloatParameterType—Part 2**

The third part shows the ToString and ValidRange elements. These are only in the numeric ParameterTypes (Float and Integer).

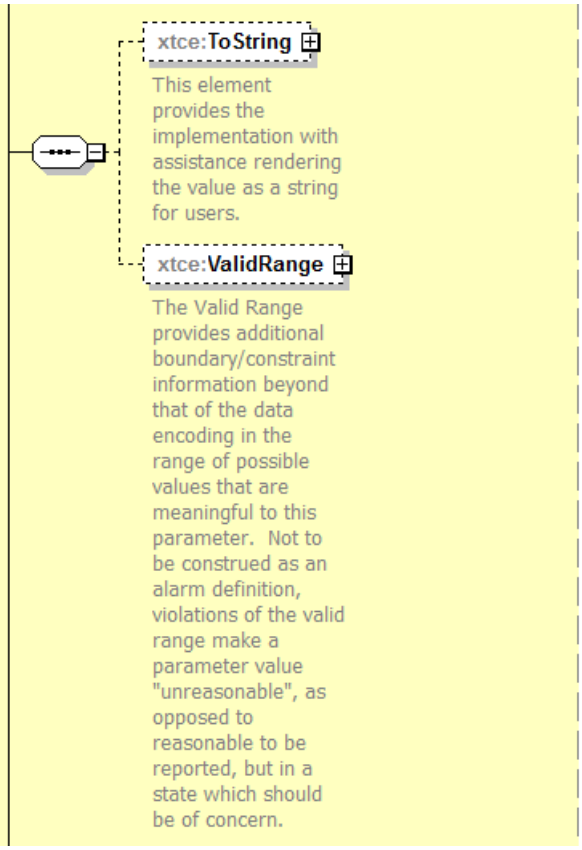



**Figure 4-8: FloatParameterType—Part 3**

The final part of Parameter, the alarms, are only in the scalar ParameterTypes, with the exception of AbsoluteTimeParameterType.

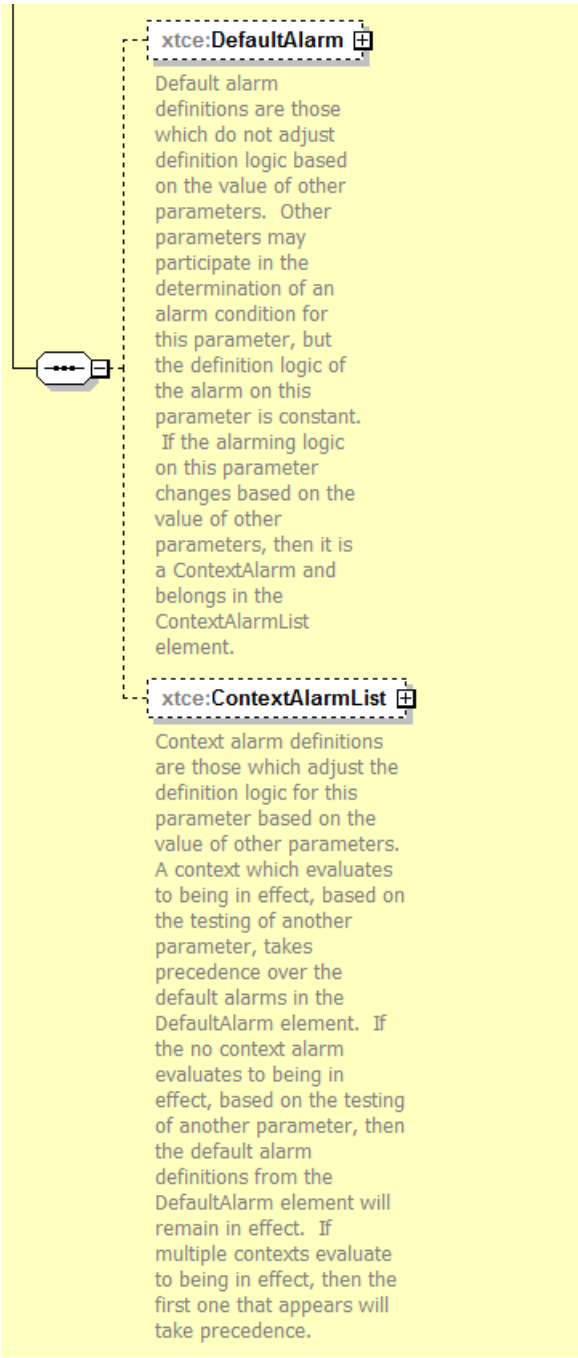



**Figure 4-9: FloatParameterType—Part 4**

NOTE – In BinaryParameterType, the ContextAlarmList is called BinaryContextAlarmList. This appears to be a typo in XTCE 1.2.

The ParameterTypes defines the relationship between the remote (source) data type of a data item (telemetry) and the destination data type that will hold it (receiving ground system data type). Alarms and calibrators are specified in this area if they are applicable.

NOTES

1 ValidRange check is only applicable to IntegerParameterTypes and FloatParameterTypes.

2 Each ParameterType has four possible DataEncodings (Integer, Float, String, Binary).

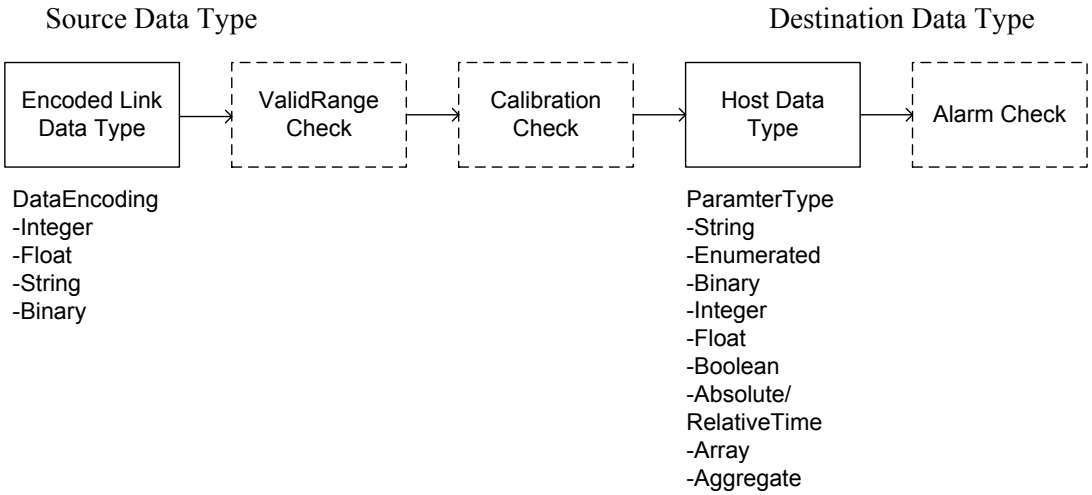



**Figure 4-10: Relationship of ParameterType Elements**

#### 4.3.2.2.2 nameDescription Attributes and Elements

Subsection 3.4.2 contains a description of the elements and attributes associated with name, shortDescription, LongDescription, AliasSet, and AncillaryDataSet.

#### 4.3.2.2.3 baseType Attribute

##### 4.3.2.2.3.1 General

Some ParameterTypes may optionally inherit from other ParameterTypes to form a new ParameterType description. The String, Enumerated, Binary, Integer, Float, and Boolean ParameterTypes have a @baseType and therefore extend other ParameterTypes.

The following broad rules apply:

a) ParameterTypes may only inherit from other ParameterTypes that have a baseType attribute (e.g., ArgumentTypes may only extend other ArgumentTypes).

b) Inheritance may only occur between the same ParameterTypes (e.g., IntegerParameterTypes may only inherit from another IntegerParameterType).

c) Only certain elements and attributes may be overridden or inherited by the child. In some cases these are specific to each ParameterType.

d) baseType NameReferences that form loops are illegal.

e) Empty or unspecified content does not override explicit content.

f) The result of the inheritance must be a valid XTCE ParameterType construction.

#### 4.3.2.2.3.2 Common ParameterType Inheritance Rules

Common ParameterType inheritance is used to narrow a general parameter description that is applicable to many parameters.

The underlying ParameterType/DataEncoding relationship cannot be changed. For example, it may be desirable to add alarms specific to a device that is represented in more than one location on a SpaceSystem.

Table 4-1 is a list of elements and attributes per type that the baseType inheritance mechanism can apply.

**Table 4-1: Common ParameterType Inheritance Rules**

| Element or Attribute | Inheritance Rule |
|---|---|
| @name | Not inherited by child. |
| @shortDescription | Not inherited by child. |
| @baseType | N/A. |
| @initialValue | The child overrides parent's shortDescription if supplied; otherwise, the child gets the parent's content if it is present. |
| LongDescription | Not inherited by child. |
| AliasSet | Not inherited by child. |
| AncillaryDataSet | Child's content prefixed to parent's content if present. |
| UnitSet | The child's UnitSet will override the parent's if it has content (is nonempty); otherwise, it gets the parent's UnitSet. An empty UnitSet cannot override one that has content. |
| DataEncodings | The DataEncodings and most of their child elements cannot be overridden or changed by the child. The calibrator elements can be set or overridden by the child. An empty (or unspecified) DataEncoding cannot override one with content. If the parent has no DataEncoding, the child cannot specify a DataEncoding. The DataEncoding specified in the child should match the parent. |
| DataEncoding/DefaultCalibrator | The child may override the parent's content or inherit the parent's content. |
| DataEncoding/ContextCalibrator | The child may add calibrators that are prefixed to a specified parent. Otherwise, the child inherits any specified by the parent. |
| BinaryDataEncoding/TransformAlgorithm | FromBinaryTransformationAlgorithm and ToBinaryTransformationAlgorthm can be overridden by the child if specified in the parent. Otherwise the child inherits the parent's content. |
| DefaultAlarm | The child's DefaultAlarm overrides the parent's if supplied; otherwise, the child gets the parent's content, if present. |
| ContextAlarmList | The child's content is prefixed to the parent's content, if present. |

#### 4.3.2.2.3.3 Specific ParameterType Inheritance Rules

##### 4.3.2.2.3.3.1 General

Certain items are specific to each ParameterType. Those items are listed in table 4-2.

**Table 4-2: Specific ParameterType Inheritance Rules**

| Element or Attribute | Inheritance Rule |
|---|---|
| StringParameterType/@restrictionPattern | The child overrides the parent's if supplied; otherwise the child gets the parent's content if present. |
| StringParameterType/@characterWidth | The child may not override parent content. |
| StringParameterType/SizeRangeInCharacters | The child overrides the parent's if supplied; otherwise the child gets the parent's content if present. |
| EnumeratedParameterType/EnumerationList | The child may add enumerations that are prefixed to the parent's. |
| IntegerParameterType/@sizeInBits | The child cannot override the parent, including default values. The child gets the parent's value even if it specifies a different value. |
| IntegerParameterType/@signed | The child cannot override the parent. The child gets the parent's value even if it specifies a different value. |
| IntegerParameterType/@validRangeAppliesToCalibrated | The child cannot override the parent. The child gets the parent's value even if it specifies a different value. |
| IntegerParameterType/toString | The child overrides the parent's if supplied; otherwise,the child gets the parent's content, if present. |
| IntegerParameterType/ValidRange | The child overrides the parent's if supplied; otherwise, the child gets the parent's content, if present. An empty ValidRange cannot override one with content. |
| FloatParameterType/@sizeInBits | The child cannot override the parent. The child gets the parent's value even if it specifies a different value. |
| FloatParameterType/@validRangeAppliesToCalibrated | The child cannot override the parent. The child gets the parent's value even if it specifies a different value. |
| FloatParameterType/toString | The child overrides the parent's if supplied; otherwise, the child gets the parent's content, if present. |
| FloatParameterType/ValidRange | The child overrides the parent's if supplied; otherwise, the child gets the parent's content, if present. An empty ValidRange cannot override one with content. |
| BooleanParameterType/@oneStringValue | The child cannot override the parent. The child gets the parent's value even if it specifies a different value. |
| BooleanParameterType/@zeroStringValue | The child cannot override the parent. The child gets the parent's value even if it specifies a different value. |

#### 4.3.2.2.3.3.2 StringParameterType Example

The following is the original base ParameterType. It defines a general ‘status message’ that is simply a string of a fixed length.

```
<xtce:StringParameterType name="StatusMsgType">
    <xtce:UnitSet/>
        <xtce:StringDataEncoding>
            <xtce:SizeInBits>
                <xtce:Fixed>
                    <xtce:FixedValue>160</xtce:FixedValue>
                </xtce:Fixed>
            </xtce:SizeInBits>
        </xtce:StringDataEncoding>
    </xtce:StringParameterType>
```

A particular ‘PSU-B’ subsystem ParameterType extends the base ParameterType. Only strings that start with ‘PSU-B:’ are allowed. The extending type inherits the string encoding from the parent and adds a restriction pattern and alarm.

```
<xtce:StringParameterType name="PSU-B_MsgType" restrictionPattern="^PSU-B:"
baseType="StatusMsgType">
    <xtce:UnitSet/>
        <xtce:StringDataEncoding/>
    <xtce:DefaultAlarm>
        <xtce:StringAlarmList>
            <xtce:StringAlarm matchPattern="PSU-B: LL0V OUT" alarmLevel="warning"/>
        </xtce:StringAlarmList>
    </xtce:DefaultAlarm>
</xtce:StringParameterType>
```

The following shows a construction if all the element and attributes were combined into a single ParameterType.

```
<xtce:StringParameterType name="ResultsOfInheritance_PSU-B_Msg" restrictionPattern="^PSU-B:">
    <xtce:UnitSet/>
    <xtce:StringDataEncoding>
        <xtce:SizeInBits>
            <xtce:Fixed>
                <xtce:FixedValue>160</xtce:FixedValue>
            </xtce:Fixed>
        </xtce:SizeInBits>
    </xtce:StringDataEncoding>
    <xtce:DefaultAlarm>
        <xtce:StringAlarmList>
            <xtce:StringAlarm matchPattern="PSU-B: LL0V OUT" alarmLevel="warning"/>
        </xtce:StringAlarmList>
    </xtce:DefaultAlarm>
</xtce:StringParameterType>
```

#### 4.3.2.2.4 UnitSet

##### 4.3.2.2.4.1 General

Units are defined in the UnitSet area within each ParameterType. UnitSet is an option element now in XTCE 1.2. It may be empty.

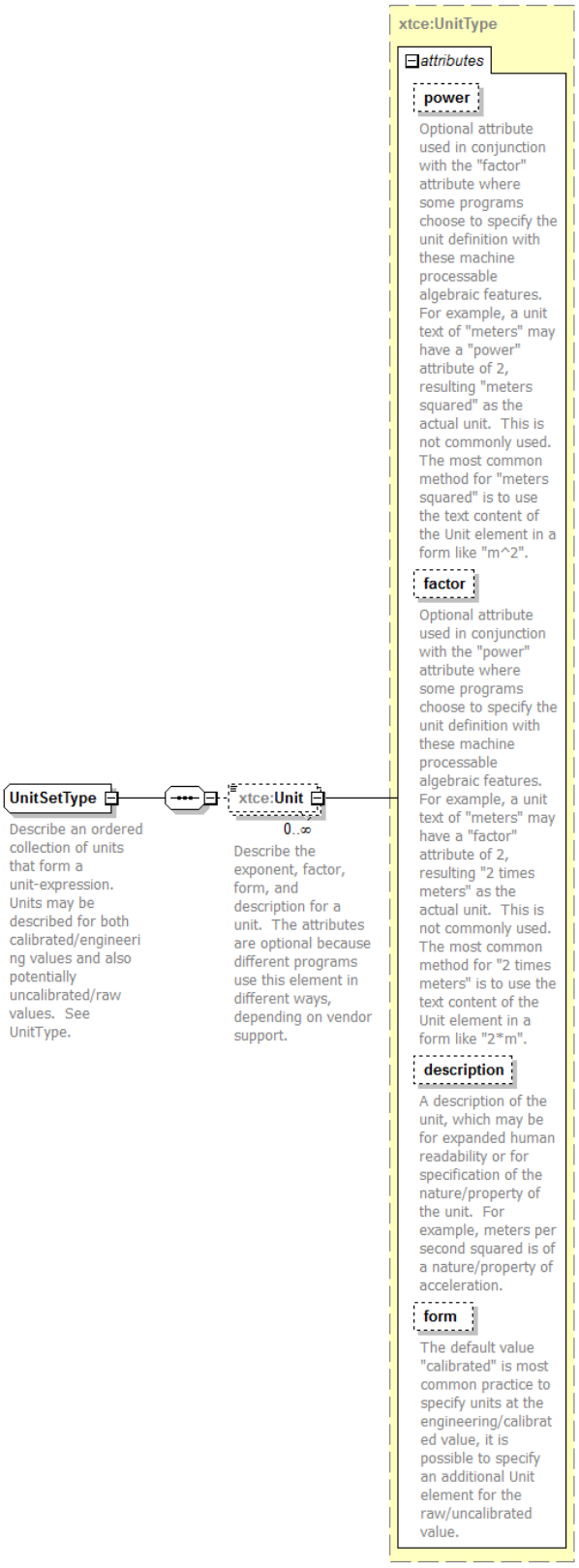



**Figure 4-11: UnitSet**

Unit elements in UnitSet are used together to build more complex formulas, for example, to define ‘meter/sec’ (meters per second). In this example, two Unit elements would be defined: one with a content of ‘meter’, a power of ‘1’ (default), and a factor of ‘1’ (default); the other with a content of ‘sec’, a power of ‘-1’, and a factor ‘1’. The result is a formula of ‘meter * sec^-1’ or ‘meter/sec’.

Unit terms are multiplied together to form a more complex formula.

```
<xtce:UnitSet>
    <!--this is meters/sec or meters * seconds^-1-->
    <xtce:Unit>meters</xtce:Unit>
    <xtce:Unit power="-1">second</xtce:Unit>
</xtce:UnitSet>
```

#### 4.3.2.2.4.2 Form Attribute

The form attributes, new in XTCE 1.2, can be used to set the unit as being related to the calibrated value or raw value.

The default is calibrated, and this is typical. However, because UnitSet can allow multiple Unit elements, it is possible to set the raw unit by specifying Unit two times. This would be in conjunction with the common practice of just using the Unit element itself to hold the unit text, and ignoring the other attributes.

RECOMMENDATION – Many (or even most) implementations only support Unit as holding the text representation of the entire unit value. The power and factor attributes are ignored, and multiple Units are not formed into an expression. Or if there is more than one Unit element as indicated above, it holds multiple Unit values, one for calibrated and one for raw.

```
<xtce:UnitSet>
    <!--this is the common practice way for meters/sec or meters * seconds^-1-->
    <xtce:Unit>m/s</xtce:Unit>
</xtce:UnitSet>
```

### 4.3.2.2.5 DataEncoding Elements

#### 4.3.2.2.5.1 General

The DataEncoding elements in most of the ParameterTypes describe source value format from the spacecraft. The four supported encodings are StringDataEncoding, IntegerDataEncoding, FloatDataEncoding, and BinaryDataEncoding, which will be described in the sections below. However, each of these shares a common schema type lineage, and this will be described first.

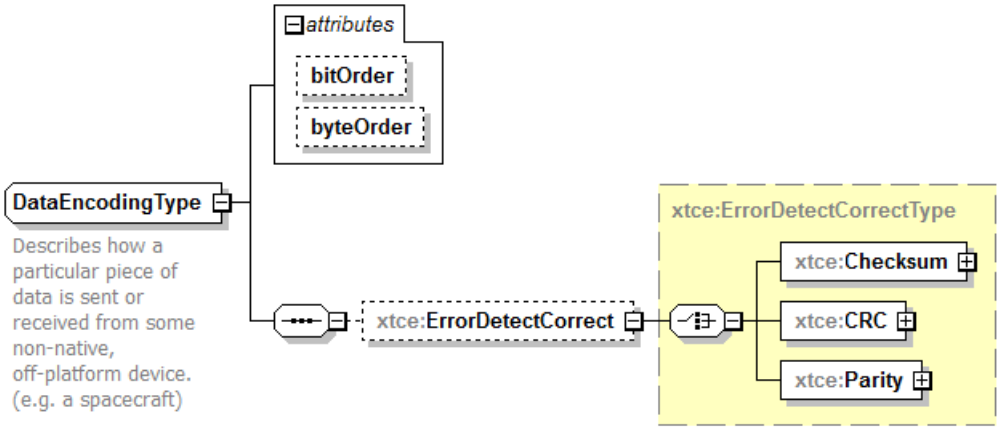



**Figure 4-12: Parent DataEncodingType**

All DataEncodings share the same parent schema type (DataEncodingType) and so follow a pattern and share certain elements and attributes. These will be described first.

#### 4.3.2.2.5.2 bitOrder Attribute

The bitOrder attribute is used to specify whether the Most Significant Bit (MSB) or Least Significant Bit (LSB) is the first bit in the stream. The default is mostSignificantBitFirst.

#### 4.3.2.2.5.3 ErrorDetectCorrect Element

The ErrorDetectCorrect element describes the parity or Cyclic Redundancy Check (CRC) checks that may be used during the transmission of information on the link. New for XTCE1.2, a Checksum element has been added.

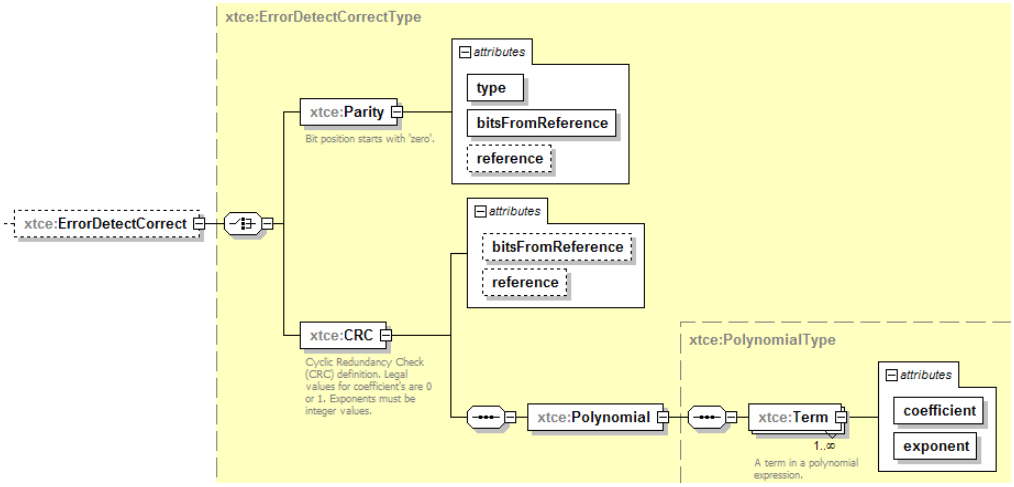



**Figure 4-13: ErrorDetectCorrect Element**

The Parity element is used to describe the offset (bitsFromReference), where the offset should be applied (start or end), and the type (even or odd).

– Reference=start:
  - offset=0: the first bit in the stream of the item;
  - offset=1: the 2nd bit in the stream of the item;
  - offset=length of item -2: the last bit in the stream of the item;
  - length of item -1: parity location;
– Reference=end:
  - offset=0: the last bit in the stream of the item;
  - offset=1: the 2nd to last bit in the stream of the item;
  - offset=length of item -1: the first bit in the stream of the item;
  - length of item -1: parity location.

The parity calculates from the location (reference, bitsFromReference) to the last bit of the item, not including the parity bit itself.

The CRC describes the offset (bitsFromReference) and the reference (start or end), then the polynomial itself (see 4.3.2.2.6.3, which reuses the same schema types). Other CRC details may need to be stored in AncillaryData. BitsFromReference and reference are similar to Parity.

The Checksum…

#### 4.3.2.2.5.4 ByteOrder Attribute

The ByteOrderList element has been reworked in XTCE 1.2 to be a text-based attribute. The ByteOrder allows the specification of byte order transmission of an item. If the element is unspecified, then the Most Significant Byte (MSB) is transmitted first in the stream. Otherwise, a list can be constructed to describe the transmission byte orderings.

To construct a new ordering, it must be recognized that the first Byte element in the list corresponds to the first item in the stream. Its byteSignificance attribute is then used to weight the other Bytes' order relative to each other in the list. For example, if the first **b**yte has the lowest byteSignificance in terms of its value, it will be the Least Significant Byte (LSB) in the list. If it has the highest value, it would be the MSB.

```
<xtce:IntegerDataEncoding bitOrder="leastSignificantBitFirst" sizeInBits="24">
    <xtce:ByteOrderList>
        <xtce:Byte byteSignificance="2"/>
        <xtce:Byte byteSignificance="1"/>
```

```
            <xtce:Byte byteSignificance="0"/>
        </xtce:ByteOrderList>
    </xtce:IntegerDataEncoding>
```

The LSB should be given the value of '0' and the MSB a value of 'ByteOrderList length -1'. For example, in a four Byte ByteOrderList, the MSB would have a value of '3', and the LSB would have a value of '0'.

In the above example, a 24-bit item is downlinked. The 3rd byte (2—MSB) is to be received first, then the 2nd (1—middle) byte, and finally the 1st byte (0—LSB). Within each byte in this construction, the bits are ordered from low-bit first to high-bit.

This element can be used in conjunction with the bitOrder attribute, which is present in every DataEncoding. The bit ordering applies to the entire construction.

#### 4.3.2.2.5.5 StringDataEncoding Element

##### 4.3.2.2.5.5.1 General

The rest of StringDataEncoding is described in this section. Other elements have been hidden if they are not specific to StringDataEncoding and have been discussed above.

StringDataEncoding describes strings data types as they appear in telemetry so that they can be decommutted successfully on the ground.

StringDataEncoding supports several kind of string formats, including UTF-8, UTF-16, and US-ASCII, along with several others listed below.

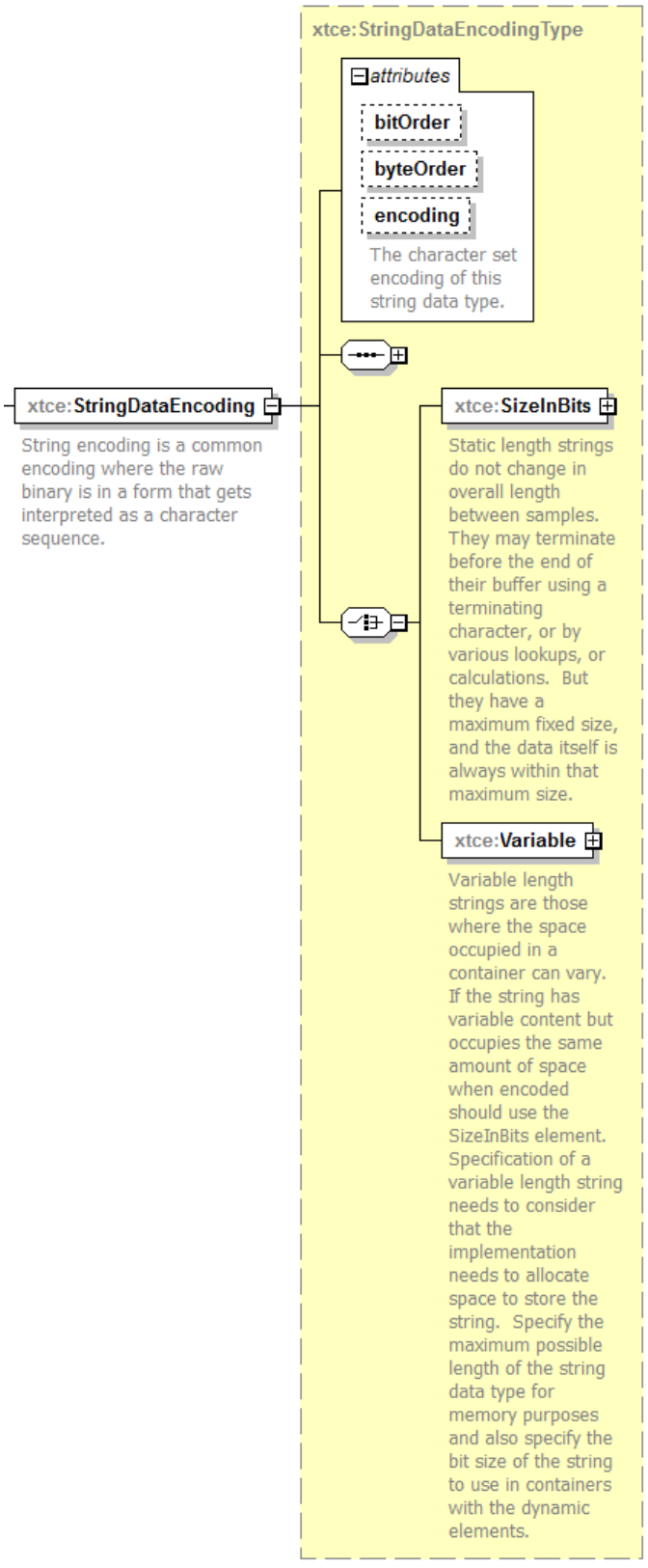



**Figure 4-14: StringDataEncodingType**

StringDataEncoding supports describing string telemetered values in two overall ways. These are discussed below.

#### 4.3.2.2.5.5.2 encoding Attribute

The encoding format of the string can be one of these:

- US-ASCII;
- ISO-8859-1;
- Windows-1252;
- UTF-8;
- UTF-16;
- UTF-16LE;
- UTF-16BE;
- UTF-32;
- UTF-32LE;
- UTF-32BE.

NOTE – Where the byte order is applicable, it may conflict with the byteOrder attribute which should elicit at least a warning in any implementation.

#### 4.3.2.2.5.5.3 SizeInBits

The first form of a string is one that is fixed in length overall. It may be literally of a specific length at all times, or maybe fit within an fixed length field but be terminated specifically to length for each instance value.

In either case, the SizeInBits element is used for this type of string.

#### 4.3.2.2.5.5.4 SizeInBits/Fixed/FixedValue Element

The SizeInBits/Fixed element is set to a fixed bit length portion of the description. If no other elements are set, then this is the specific size of the string.

NOTE – The SizeInBits should take into the account the character encoding information.

#### 4.3.2.2.5.5.5 SizeInBits/TerminationChar Element

If the TerminationChar is specified under SizeInBits, then the string fits within the 'box' of FixedValue, but may be shorter due to the presence of the terminatingChar.

```
<xtce:StringDataEncoding>
    <xtce:SizeInBits>
        <xtce:TerminationChar>0000</xtce:TerminationChar>
    </xtce:SizeInBits>
</xtce:StringDataEncoding>
```

#### 4.3.2.2.5.5.6 SizeInBits/LeadingSize Element

LeadingSize is another way some strings are sized. It is similar to the string length specification in Pascal strings. The string length is specified before the characters. This element is the length in bits of that size tag. The default is 16 bits.

```
<xtce:StringDataEncoding>
    <xtce:SizeInBits>
        <xtce:LeadingSize sizeInBitsOfSizeTag="17"/>
    </xtce:SizeInBits>
</xtce:StringDataEncoding>
```

For the purposes of illustration, the leading size is 17 bits, which is an unlikely value in real life since the default is 16. The size should always be interpreted as an unsigned integer.

If this element is specified under SizeInBits, then the string fits within the 'box' of FixedValue, but may be shorter due to the presence of the terminatingChar.

RECOMMENDATION – It is an error if SizeInBits is set and both TerminationChar and LeadingSize are set.

#### 4.3.2.2.5.5.7 Variable

If Variable is set, the string is truly variable length in nature. In this case, the maximum size allowed of the value must be set. The syntax is shown below.

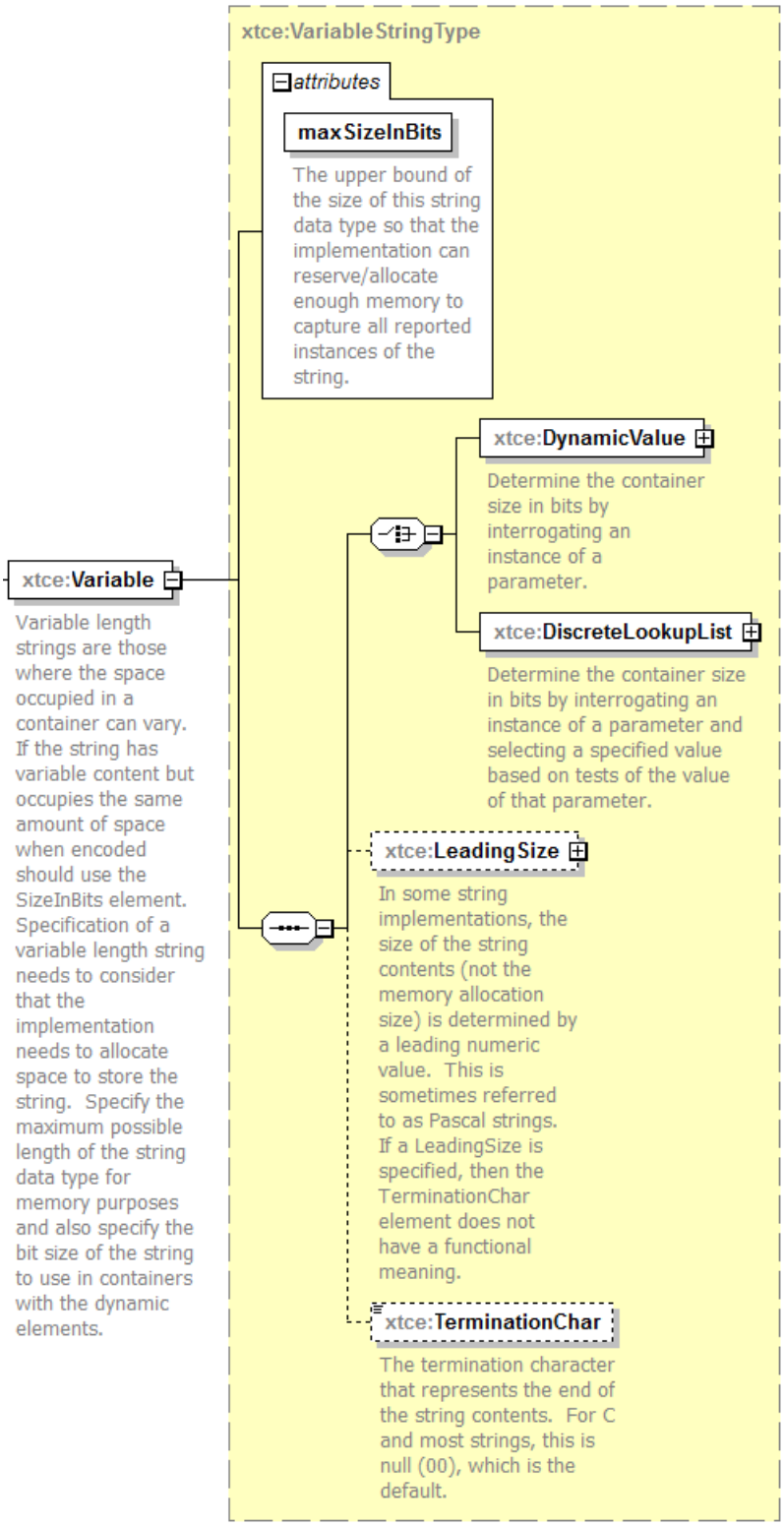



**Figure 4-15: Variable String**

As can be seen in the above diagram, there are several ways to specify the length of the variable length string: by looking up another value (DynamicValue or DiscreteLookupList), by a leading size, or by a terminationChar.

RECOMMENDATION – It an error for more than one of these to be set, even though the syntax allows it.

#### 4.3.2.2.5.6 IntegerDataEncoding Element

##### 4.3.2.2.5.6.1 General

Telemetered values that are encoded as integers are described with IntegerDataEncoding. In XTCE 1.2, this schema type has been split off from FloatDataEncoding, which in XTCE 1.1 shared a common ancestor.

The IntegerDataEncoding also allows for calibrator descriptions. These should be interpreted as expressions (not shown; see 4.3.2.2.5.6.5) that convert telemetered value to some ground value in some format, typically a Float. (In that scenario, the FloatParameterType would have an IntegerDataEncoding and a calibrator definition.)

But it is not impossible in XTCE for the calibrator to be applied from the IntegerDataEncoding to IntegerParameterType, or even various other ParameterTypes.

Although legal, syntactically, some definitions are suspect due to their lack of real work use cases, and so common practice in this area has been captured in the table below (see 4.3.2.5) for DataEncoding and ParameterType combinations that are typically supported within the industry.

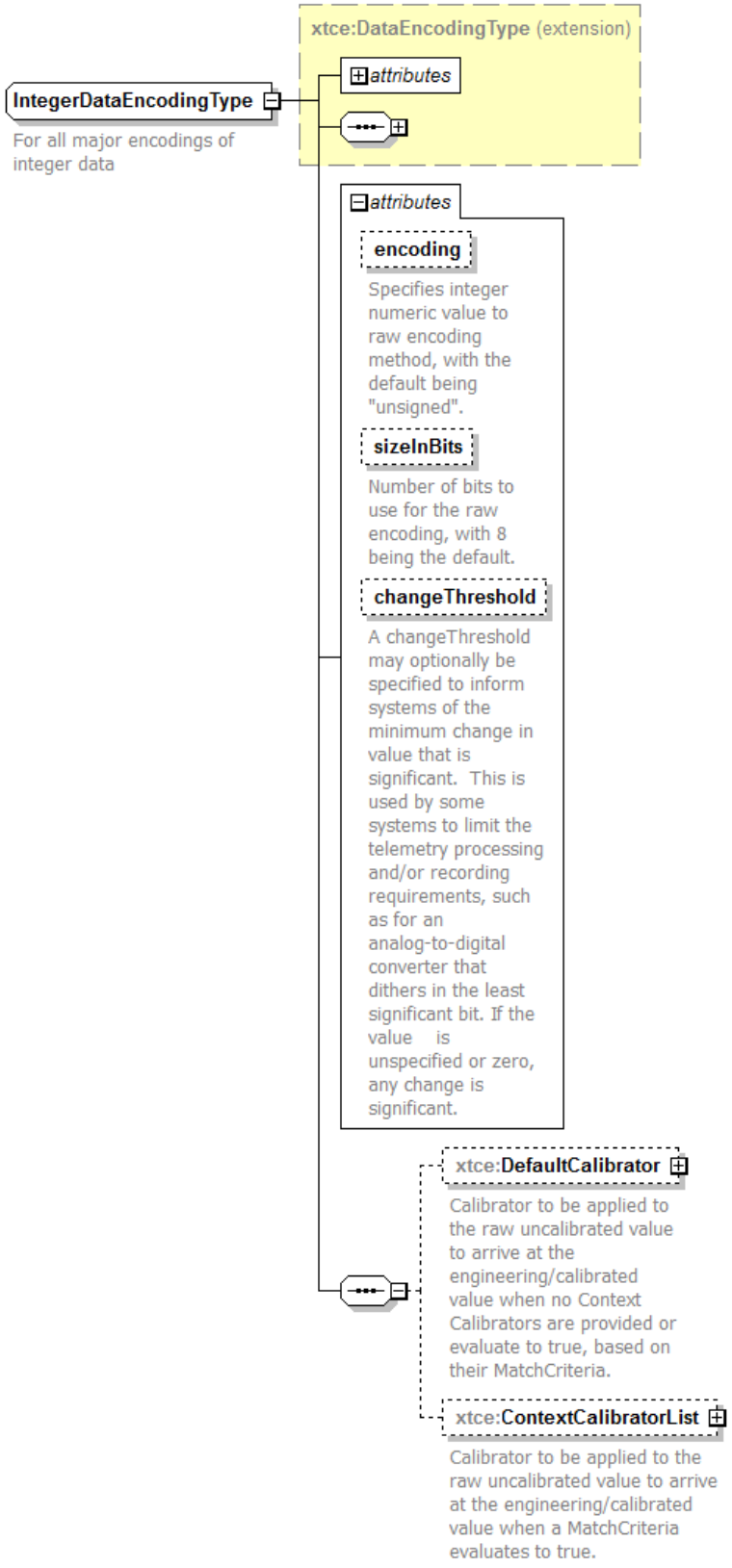



**Figure 4-16: IntegerDataEncoding**

As can be seen above, the shared common elements are not shown, and the items specific to IntegerDataEncoding are shown. These are described below.

#### 4.3.2.2.5.6.2 encoding Attribute

Choose one of the following for the encoding attribute:

- unsigned (default);
- twosComplement;
- onesComplement;
- Binary Coded Decimal (BCD);
- packedBCD.

It is typical in most use cases that the value of this encoding attribute will be either unsigned or twosComplement.

#### 4.3.2.2.5.6.3 sizeInBits Attribute

The size in bits of the item can be any positive number. The default value is 8 bits.

#### 4.3.2.2.5.6.4 changeThreshold

A flag to a real processing system on whether to collect or store this value if it does not change enough.

#### 4.3.2.2.5.6.5 DefaultCalibrator and ContextCalibratorList

Subsection 4.3.2.2.6 shows the details for the float and integer calibrators specific to IntegerDataEncoding and FloatDataEncoding.

Both elements share the same calibrator construction, using the CalibratorType for the DefaultCalibrator and the CalibratorListType for ContextCalibrator.

### 4.3.2.2.5.7 FloatDataEncoding Element

#### 4.3.2.2.5.7.1 General

FloatDataEncoding can describe various kinds of telemetered float format. In XTCE 1.2, it now supports these formats:

- EEE754_1985;
- IEEE754;

- MILSTD_1750A;
- DEC;
- IBM;
- TI.

The sizeInBits attribute sets the overall size; values are as follows: 16, 32, 40, 48, 64, 80, 128.

RECOMMENDATION – Not all formats support all the sizes. It is up to the implementation to determine if the sizeInBits and encoding combination are legal. If they are not legal, the document is in error. If the combination is not supported, that is an implementation issue.

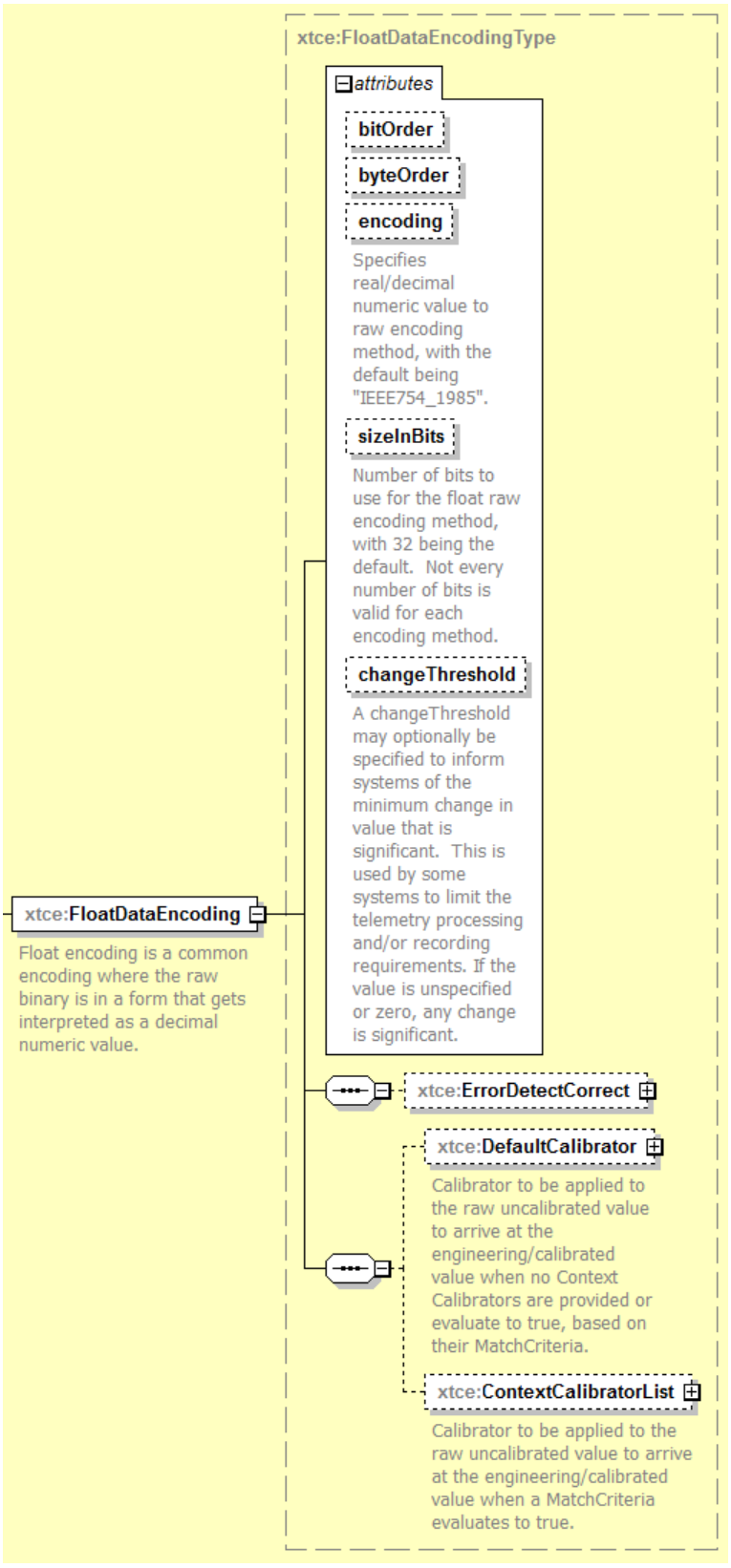



**Figure 4-17: FloatDataEncoding Element**

#### 4.3.2.2.5.7.2 encoding Attribute

Both IEEE-754 and MIL-1750A are supported. If another floating-point format is needed, BinaryEncoding is used, and the appropriate descriptive items in AncillaryData in the ParameterType are specified.

#### 4.3.2.2.5.7.3 sizeInBits Attribute

The sizeInBits attribute supports sizes 32, 64, or 128 bits. Setting the value to 'non-32' is a workaround for the MIL-1750A 48 bits.

#### 4.3.2.2.5.7.4 DefaultCalibrator and ContextCalibratorList Elements

There are two areas to define calibrators under FloatDataEncoding and IntegerDataEncoding: DefaultCalibrator and ContextCalibrator.

- The DefaultCalibrator is an optional element and may be used to define a calibrator that will apply by default. (See 4.3.2.2.5.6.5, 4.3.2.2.5.7.4, and 4.3.2.2.6.)
- ContextCalibrator is dependent on ContextMatch (a MatchCriteriaType) in order for it to be used. (See 3.4.3.6 for more explanation.)
- The number of ContextCalibrators defined in the ContextCalibratorList is unlimited.

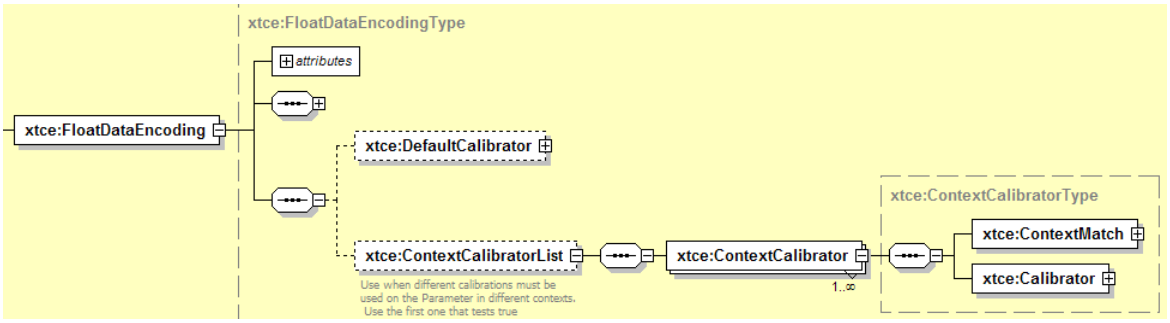



**Figure 4-18: DefaultCalibrator**

Subsection 4.3.2.2.6 contains a detailed discussion of the Numeric Calibrators specific to IntegerDataEncoding and FloatDataEncoding.

#### 4.3.2.2.5.8 BinaryDataEncoding Element

#### 4.3.2.2.5.8.1 General

BinaryDataEncoding is used to describe telemetry that is either 'data type less' in nature (in which case, it would probably be a child of BinaryParametertype) or is one of the other data

types but cannot be described using the other data encodings (for example, a float format that is unavaible in FloatDataEncoding).

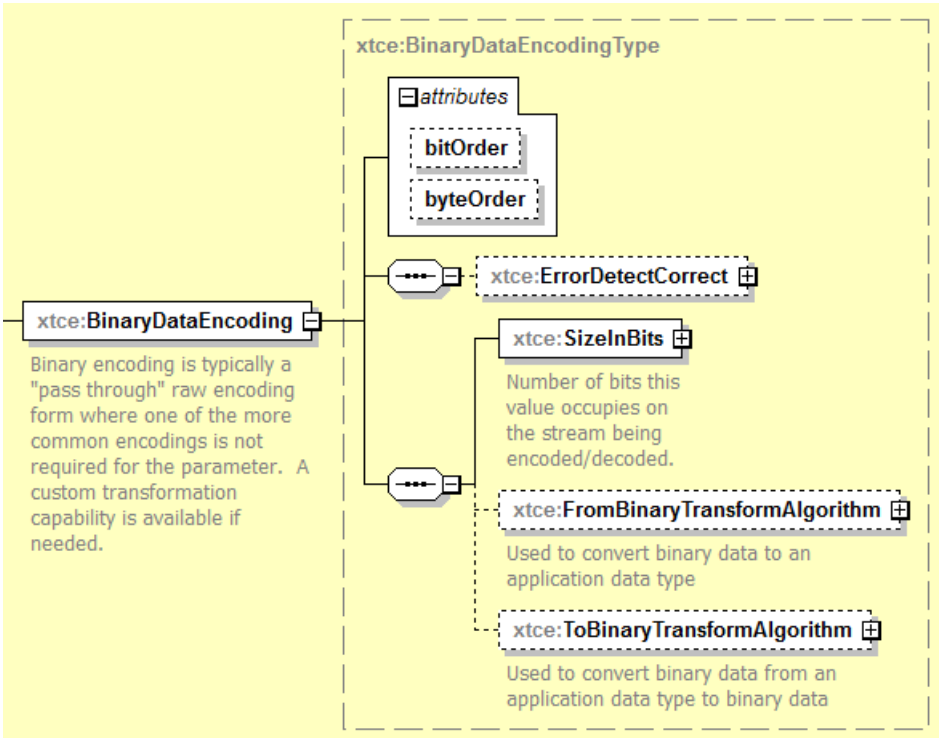



**Figure 4-19: BinaryDataEncoding**

#### 4.3.2.2.5.8.2 From/ToBinaryTransformAlgorithm

BinaryTransformAlgorithms are custom algorithms used to describe the conversion from a binary format to the destination format. (See 3.4.3.5 for a description of CustomAlgorithm.)

#### 4.3.2.2.5.9 Specifying No DataEncoding—Non-Telemetry Data Types

All the DataEncodings are optional. Specifying a ParameterType without a DataEncoding indicates that it is a ‘local’, ‘ground’, ‘constant’, or even ‘derived’ variable data type whose value will not be supplied through telemetry directly.

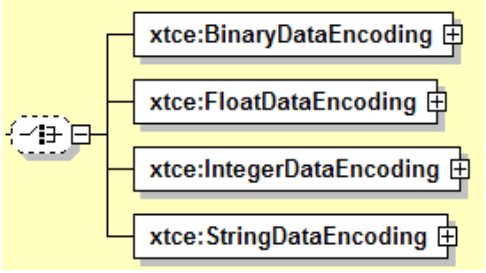



**Figure 4-20: Optional Choice for Encodings**

These variables then define values that are not explicitly part of the telemetry stream. For example, a mission phase may be represented as EnumerationParameterType (and Parameter).

The child element of Parameter/ParameterProperties/@dataSource should set this value to 'local'.

Value created locally, so called 'pseudo-telemetry' would not have a DataEncoding. (The attribute dataSource should be set to 'derived'.)

To define a system variable, a Parameter would reference a ParameterType that has no data encoding element and set the Parameter/ParameterProperties/dataSource to one of the choices that is not 'telemetered'.

Such variables may be used in various conditions-related calibrators and alarms as ParameterInstanceRefs. The following is an example of defining some kind of local or system variable data type:

```
<xtce:TelemetryMetaData>
    <xtce:ParameterTypeSet>
    <xtce:StringParameterType name="SystemNameType">
        <xtce:UnitSet/>
    </xtce:StringParameterType>
    </xtce:ParameterTypeSet>
</xtce:TelemetryMetaData>
```

In this example, the string system variable type SystemNameType has been defined as one whose value will be supplied by the system.

RECOMMENDATION – These are paired with Parameter, and so one of the appropriate Parameter/ParameterProperties/dataSource should be set: local, ground, derived, or constant. If the DataEncoding is not set and the dataSource is set to telemetered (either explicitly or implied), the implementation should at least issue a warning.

#### 4.3.2.2.5.10 SizeInBits element

Subsection 3.4.5 contains the description of the SizeInBits element.

### 4.3.2.2.6 Numeric Calibrators—FloatDataEncoding and IntegerDataEncoding

#### 4.3.2.2.6.1 General

DefaultCalibrator and ContextCalibrator may support spline, polynomial, or more general mathematical expression calibrators. These are used for FloatParameterType and IntegerParameterType.

NOTE – The DefaultCalibrator and ContextCalibrator associated with FloatDataEncoding and IntegerDataEncoding are NameDescription types. They can be optionally named, have aliases, or have AncillaryData.

#### 4.3.2.2.6.2 Spline Calibrators

##### 4.3.2.2.6.2.1 General

Linear calibrators may be represented with the SplineCalibrators element if the order attribute is set to '1' (the default). Other order values have additional meaning (see 4.3.2.2.6.2.4).

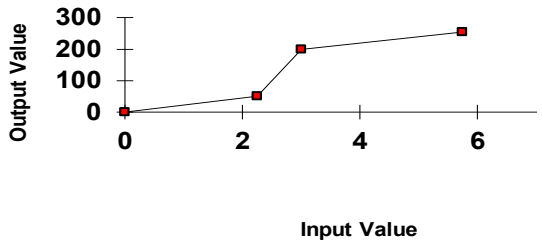



**Figure 4-21: Spline Calibrator Graph**

The linear (Spline) calibrator maps input value to an output value.

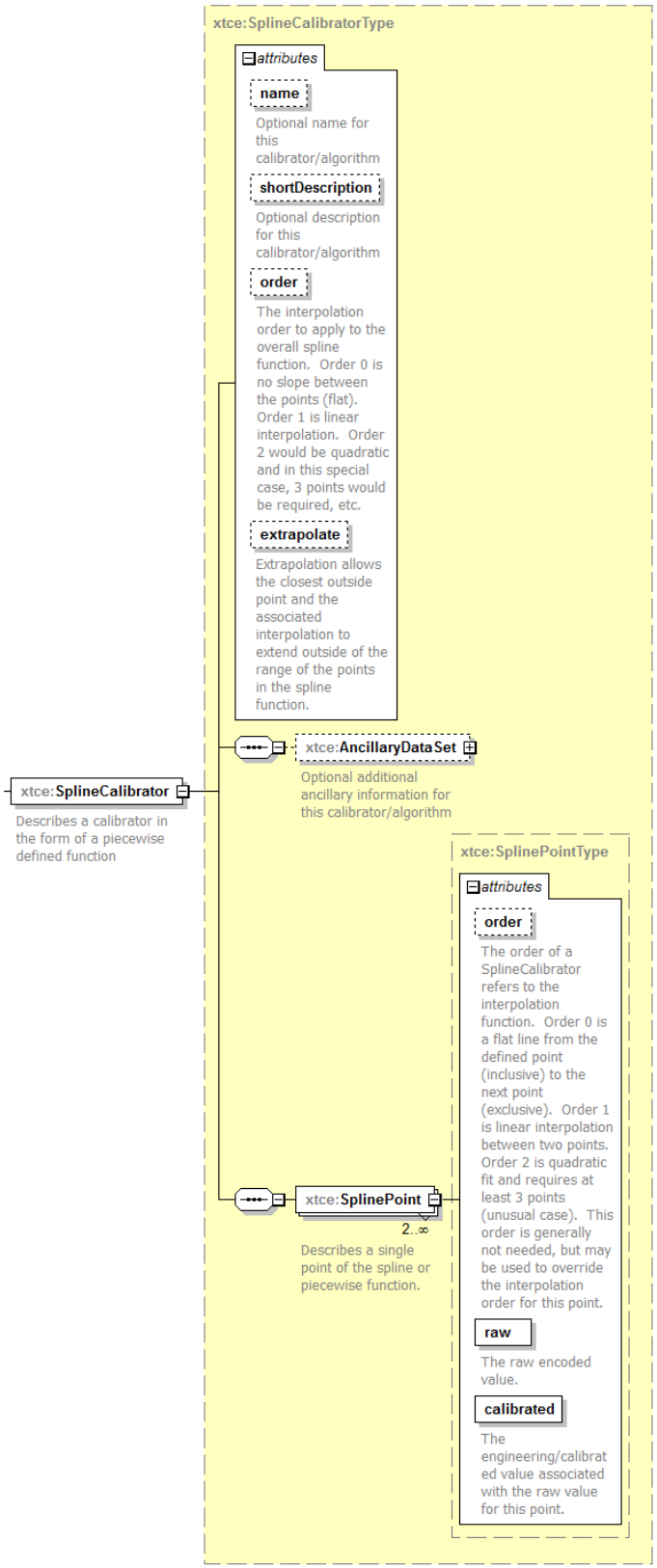



**Figure 4-22: SplineCalibrator**

In figure 4-22, the raw attribute refers to the input value, and the calibrated attribute refers to the output value.

#### 4.3.2.2.6.2.2 name

Some systems have names for the calibrators. This is a convenient place to store that information. At this time, the name cannot be referenced by other locations in the document.

#### 4.3.2.2.6.2.3 shortDescription

Descriptive information related to this calibrator.

#### 4.3.2.2.6.2.4 order Attribute

The following are the order attribute values and corresponding meanings:

- zero: a step-wise function;
- one: a linear calibrator;
- two: a quadratic spline;
- three: a cubic spline;
- four or more: (and so forth).

#### 4.3.2.2.6.2.5 extrapolate Attribute

If extrapolate is set to ‘true’, the values after the end points of the specified spline will be extrapolated.

In XTCE 1.2, AncillaryData appears in far more places, such as here in the calibrator section.

#### 4.3.2.2.6.2.6 SplinePoint

##### 4.3.2.2.6.2.6.1 General

In the example below, each point represents a mapping from an input value (raw) to an output value (calibrated):

```
<xtce:SplineCalibrator>
    <xtce:SplinePoint raw="1" calibrated="10"/>
    <xtce:SplinePoint raw="2" calibrated="100"/>
    <xtce:SplinePoint raw="3" calibrated="500"/>
</xtce:SplineCalibrator>
```

The following shows the use of some legal double-type values:

```
<xtce:SplineCalibrator>
    <xtce:SplinePoint raw="1" calibrated="10.4222E0"/>
    <xtce:SplinePoint raw="2" calibrated="100.78E-2"/>
    <xtce:SplinePoint raw="3" calibrated="INF"/>
</xtce:SplineCalibrator>
```

#### 4.3.2.2.6.2.6.2 SplinePoint—order Attribute

The order for each point can also be set here.

### 4.3.2.2.6.3 PolynomialCalibrator

#### 4.3.2.2.6.3.1 General

Polynomial calibration is supported by the PolynomialCalibrator element.

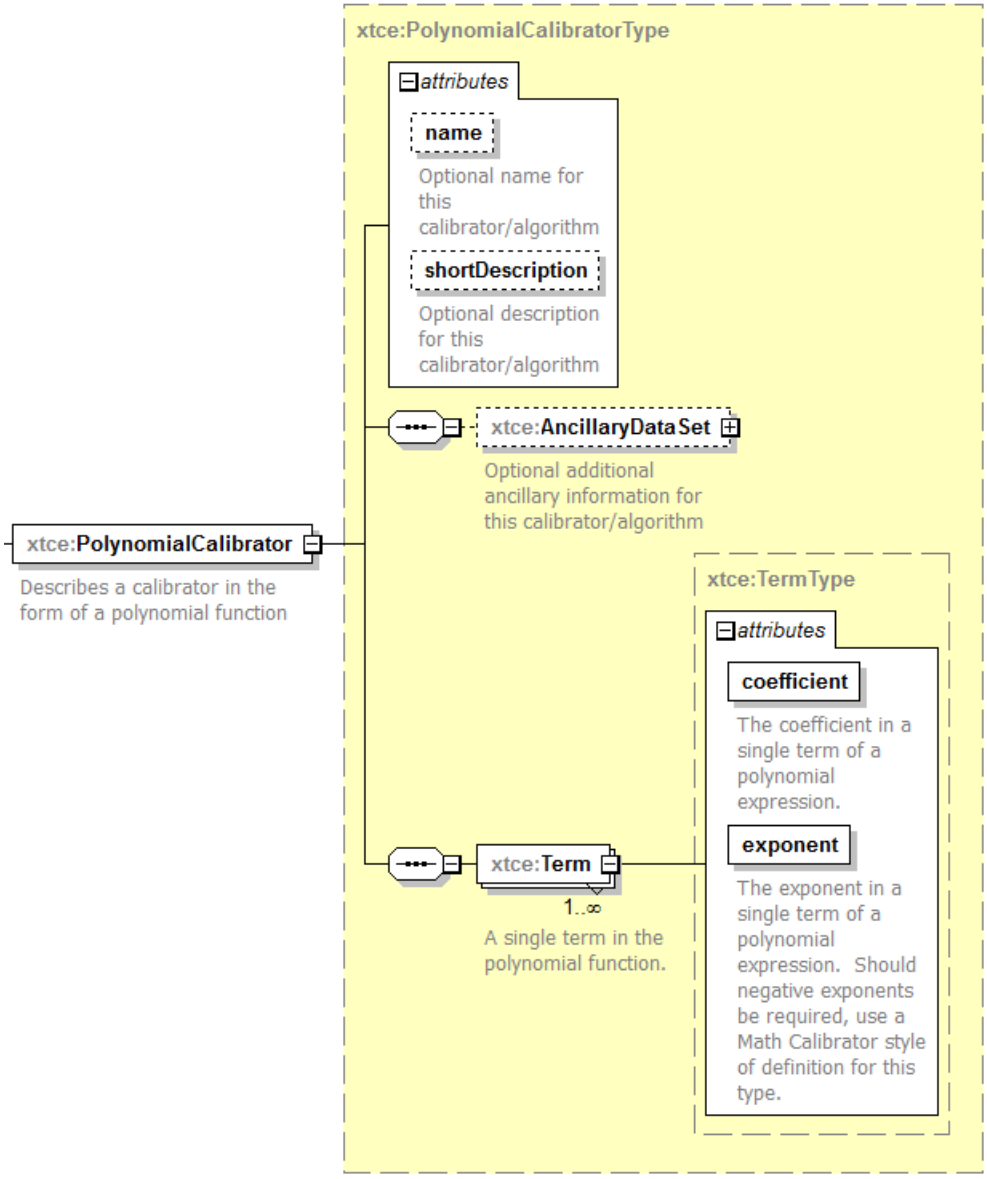



**Figure 4-23: PolynomialCalibrator**

The polynomial description consists of a series of Term elements with the attributes coefficient and exponent.

#### 4.3.2.2.6.3.2 Name, shortDescription, and AncillaryDataSet

Described above, these have been added in XTCE 1.2. to help more fully describe the calibrator.

#### 4.3.2.2.6.3.3 Term Element

##### 4.3.2.2.6.3.3.1 General

Term is used to describe a single polynomial term for a polynomial equation.

##### 4.3.2.2.6.3.3.2 coefficient Attribute

The coefficient attribute is used to specify the coefficient of the term of a polynomial equation.

##### 4.3.2.2.6.3.3.3 exponent Attribute

The exponent of the Term element is a non-negative integer value. The following is an example equation:

$$y = 0.5 + 1.5x + -0.045x^2 + 1.25x^3 + 0.0025x^4$$

The equation above is represented in table format in table 4-3.

**Table 4-3: Exponent and Coefficient Table**

| Exponent | Coefficient |
|---|---|
| 0 | 0.5 |
| 1 | 1.5 |
| 2 | -0.045 |
| 3 | 1.25 |
| 4 | 0.0025 |

The following is the XTCE representation:

```
<xtce:PolynomialCalibrator>
    <xtce:Term exponent="0" coefficient="0.5"/>
    <xtce:Term exponent="1" coefficient="1.5"/>
    <xtce:Term exponent="2" coefficient="-0.045"/>
    <xtce:Term exponent="3" coefficient="1.25"/>
    <xtce:Term exponent="4" coefficient="2.5E-3"/>
</xtce:PolynomialCalibrator>
```

#### 4.3.2.2.6.4 MathOperationCalibrator

MathOperationCalibrator describes a calibration using a variety of possible mathematical operators and operands in the MathOperationType. (See 4.3.7.3 for a discussion of this type.)

#### 4.3.2.3 Alarm Descriptions

##### 4.3.2.3.1 General

The following subsections are the various alarm descriptions available for different ParameterTypes. They are specified in the DefaultAlarm and ContextAlarmList areas of their respective elements. While some of these are unique to their specific ParameterType, others are shared among one or more. They are called out here to avoid repeating the information in each ParameterType description below, and referenced there as is appropriate.

##### 4.3.2.3.2 DefaultAlarm and ContextAlarmList

Most ParameterTypes have a DefaultAlarm and ContextAlarmList.

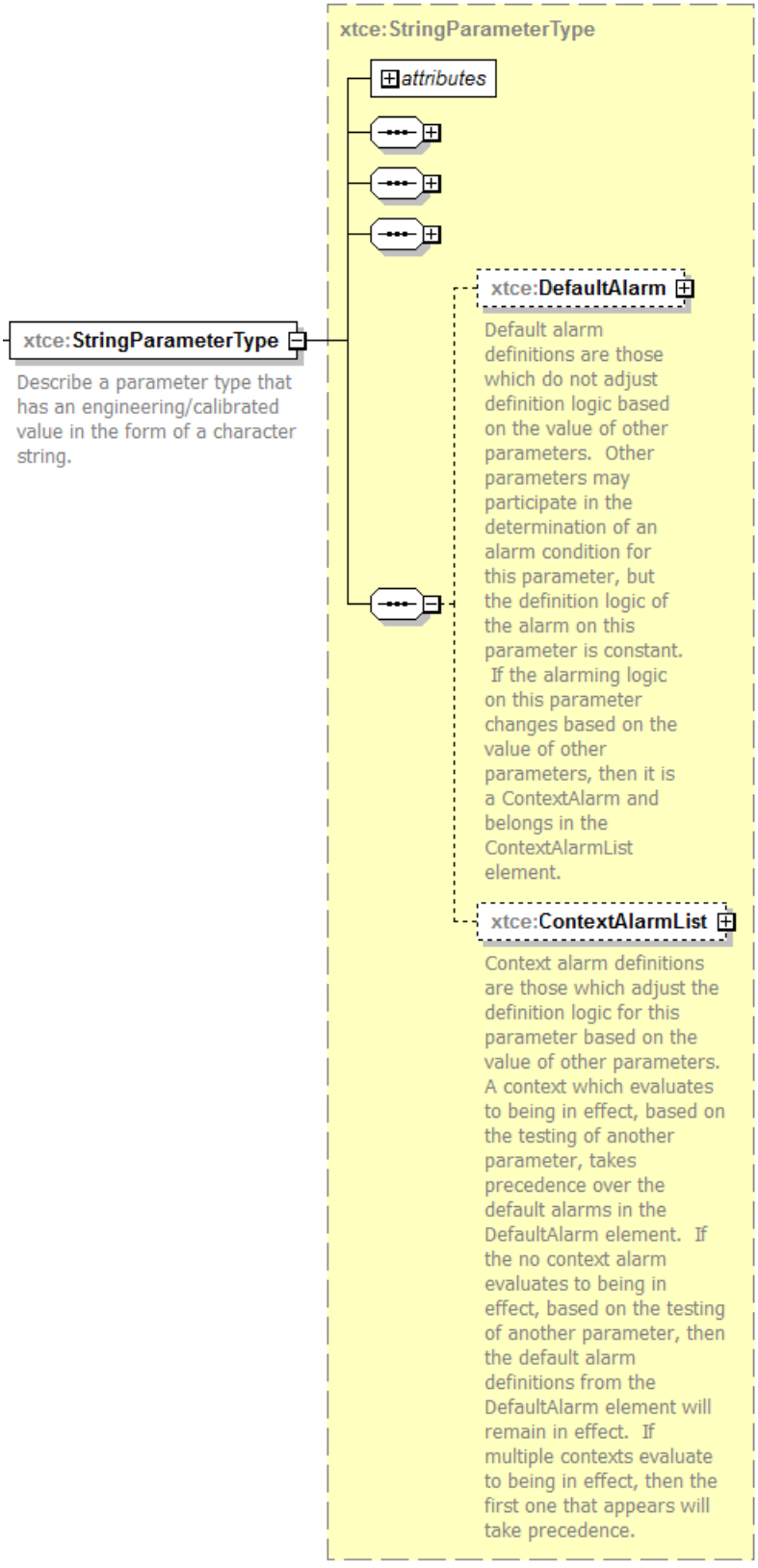



**Figure 4-24: DefaultAlarm and ContextAlarmList**

- The DefaultAlarm applies when no defined ContextAlarm is active.
- ContextAlarms are defined for certain contexts and take precedence over the DefaultAlarm. The first context to evaluate to 'true' is used to indicate which alarm to process.
- A ContextAlarm has a ContextMatch, which is a MatchCriteria (see 3.4.3.6 for further description of the MatchCriteria).
- Any number of ContextAlarms may be defined for ParameterTypes that support them in the ContextAlarmList.

#### 4.3.2.3.3 DefaultAlarm

All the scalar ParameterTypes have alarms except for AbsoluteTimeParameterType, which does not have an alarm. The base of all these default alarms is the AlarmType, which is described in the next section first.

After that, there are several alarms specific to their ParameterType, and these are described in turn.

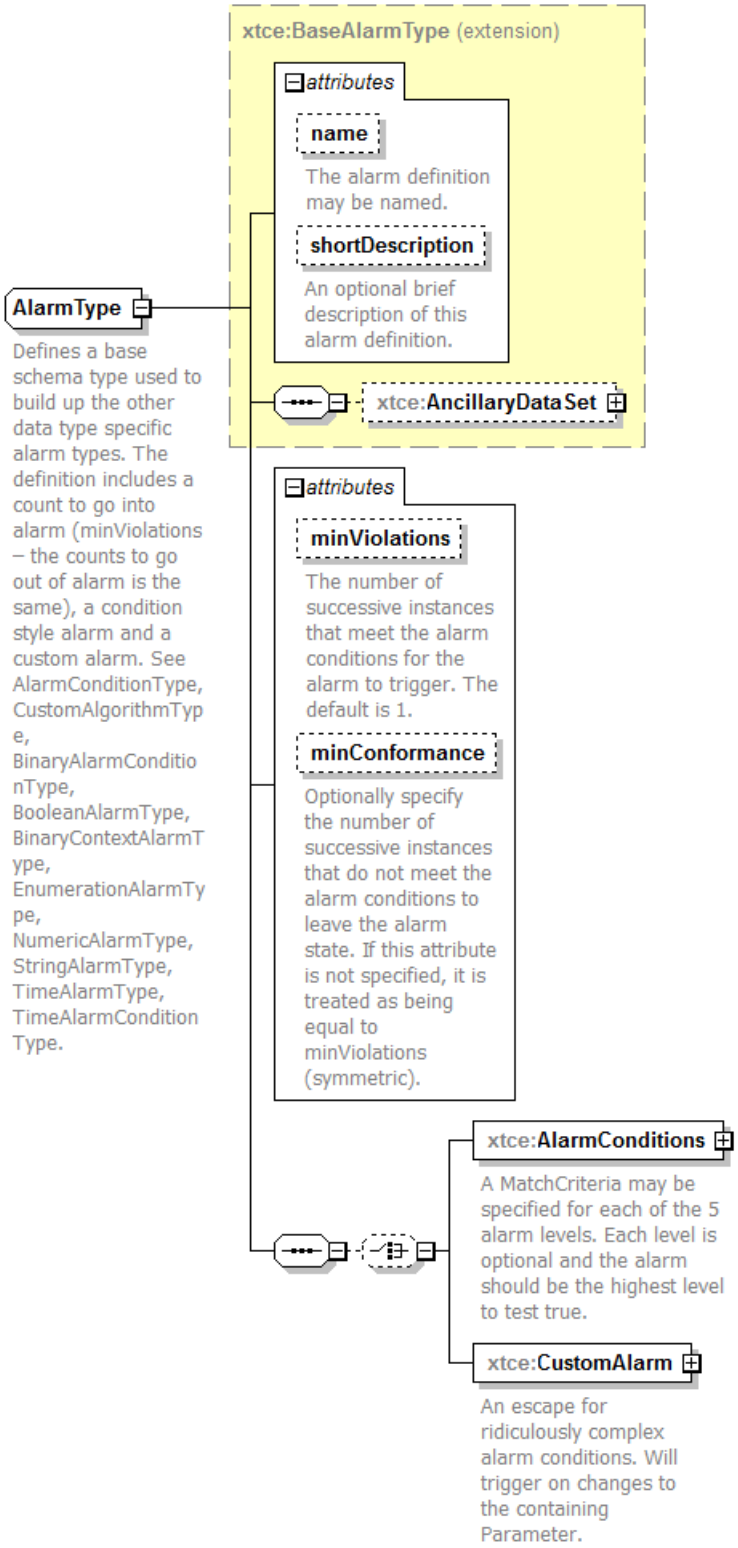



**Figure 4-25: AlarmType**

Similar to calibrators, the alarms have been expanded in XTCE 1.2 to allow them to be named and to provide descriptive or ancillary information.

#### 4.3.2.3.3.1 minViolations

The minViolations attribute is simply the number of successive violations that must occur before the Parameter is in the alarm state.

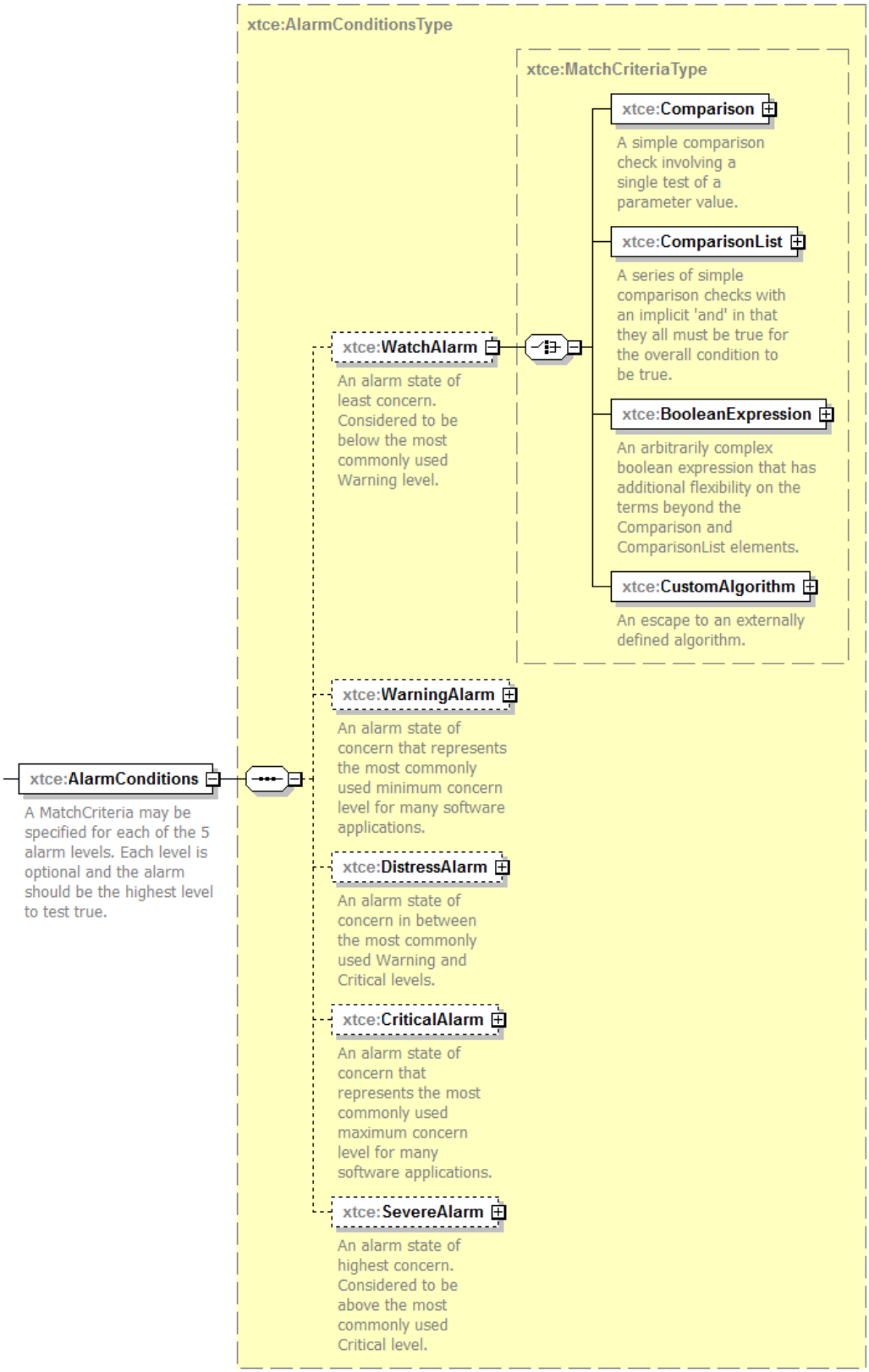
#### 4.3.2.3.3.2 minConformance

The minConformance attribute is the opposite of minViolations. It is the number of successive non-violations that remove the alarm state.

#### 4.3.2.3.3.3 AlarmConditions

The AlarmConditions element is a base kind of alarm that defines various kind of conditions that will result in an alarm state if the expression evaluates to true.

Each level is based on MatchCriteria.


xtce:AlarmConditionsType
xtce:MatchCriteriaType
xtce:Comparison
A simple comparison check involving a single test of a parameter value.
xtce:ComparisonList
A series of simple comparison checks with an implicit 'and' in that they all must be true for the overall condition to be true.
xtce:BooleanExpression
An arbitrarily complex boolean expression that has additional flexibility on the terms beyond the Comparison and ComparisonList elements.
xtce:CustomAlgorithm
An escape to an externally defined algorithm.
xtce:WatchAlarm
An alarm state of least concern. Considered to be below the most commonly used Warning level.
xtce:WarningAlarm
An alarm state of concern that represents the most commonly used minimum concern level for many software applications.
xtce:DistressAlarm
An alarm state of concern in between the most commonly used Warning and Critical levels.
xtce:CriticalAlarm
An alarm state of concern that represents the most commonly used maximum concern level for many software applications.
xtce:SevereAlarm
An alarm state of highest concern. Considered to be above the most commonly used Critical level.
xtce:AlarmConditions
A MatchCriteria may be specified for each of the 5 alarm levels. Each level is optional and the alarm should be the highest level to test true.

#### 4.3.2.3.3.4 CustomAlarm

The CustomAlarm element is an algorithm-based alarm. Because it also extends BaseAlarm, it has the name and descriptive elements from that.

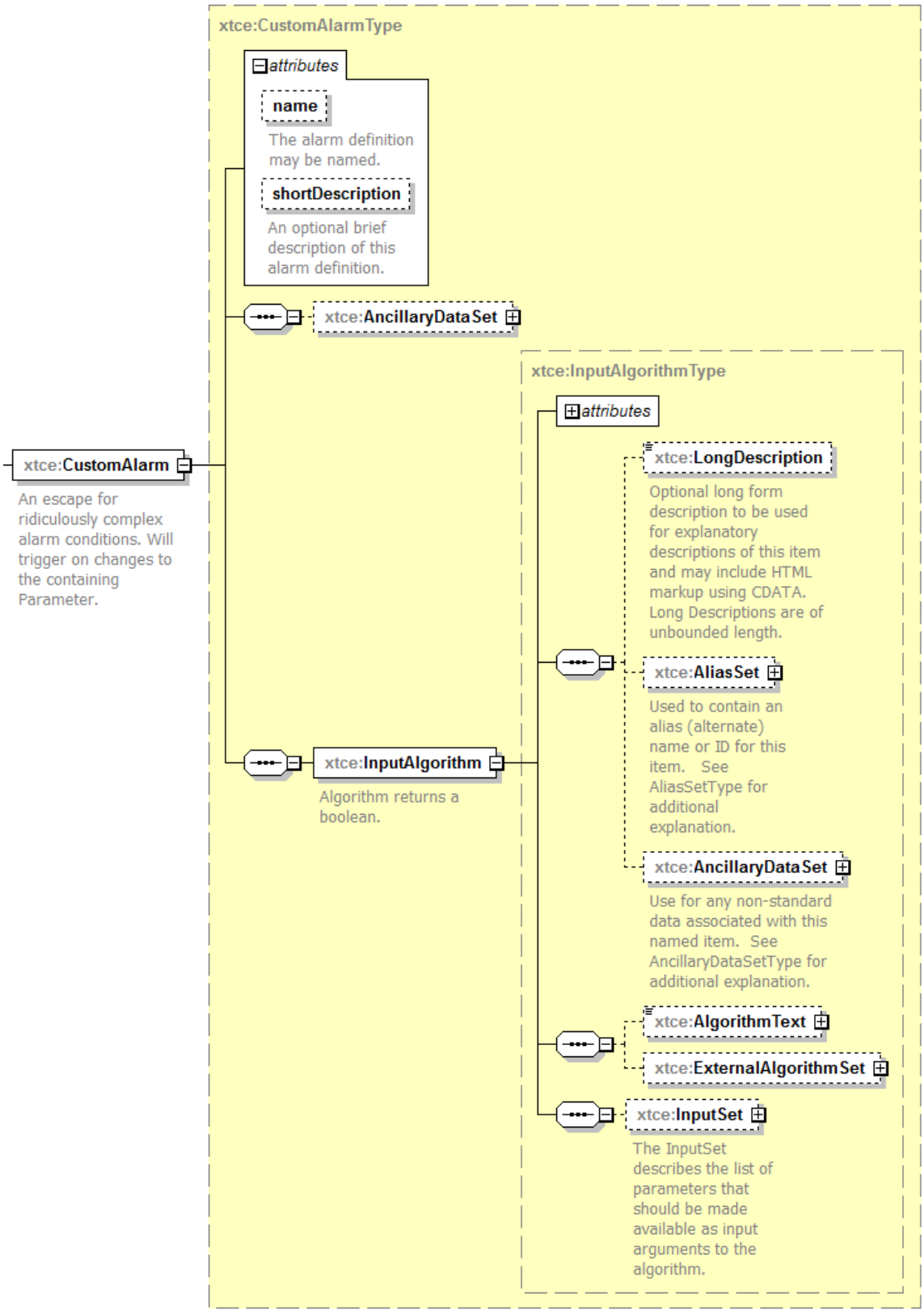



**Figure 4-26: CustomAlarm**

And because it has the name and descriptive attribtues from InputAlgorithm, it appears to have another set from that.

Recommenation: use the InputAlgorithm for the name of the algorithm and the top alarm name, which is optional for the alarm name itself (these may be the same), but ignore the name in CustomAlarm itself.

#### 4.3.2.3.4 ContextAlarmList

ContextAlarms are alarm definitions that have an expression that must be evaluated to determine if the alarm is active. They are specialized by scalar ParameterType (less AbsoluteTimeParameterType, which has no alarms), but the overall pattern is the same among them.

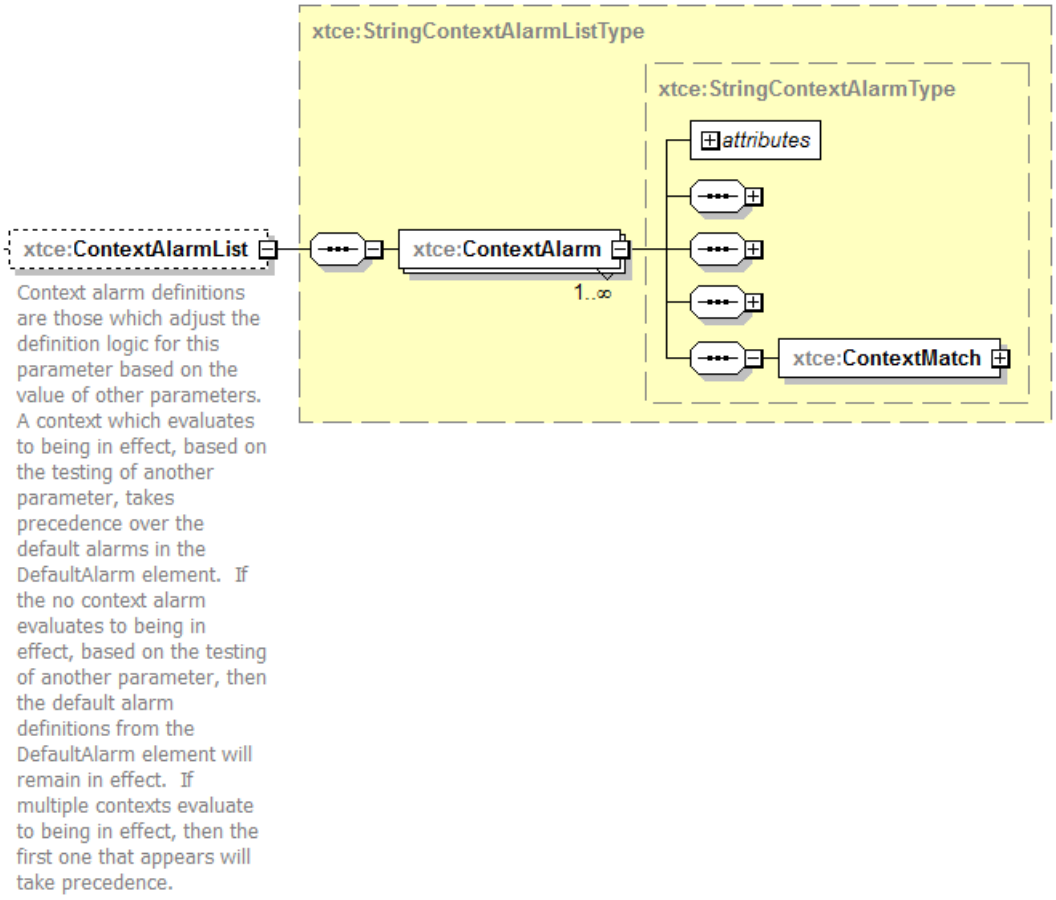



**Figure 4-27: ContextAlarmList**

There can be any number of these since it is a list, and they all must be evaluated to determine if they are active.

Finally if none of them are active or in the alarm state, the default alarm should be considered next.

In a way, it is something like a chain of if-then-else statements, where each ContextAlarm is an if-then, and the final else is the DefaultCalibrator.

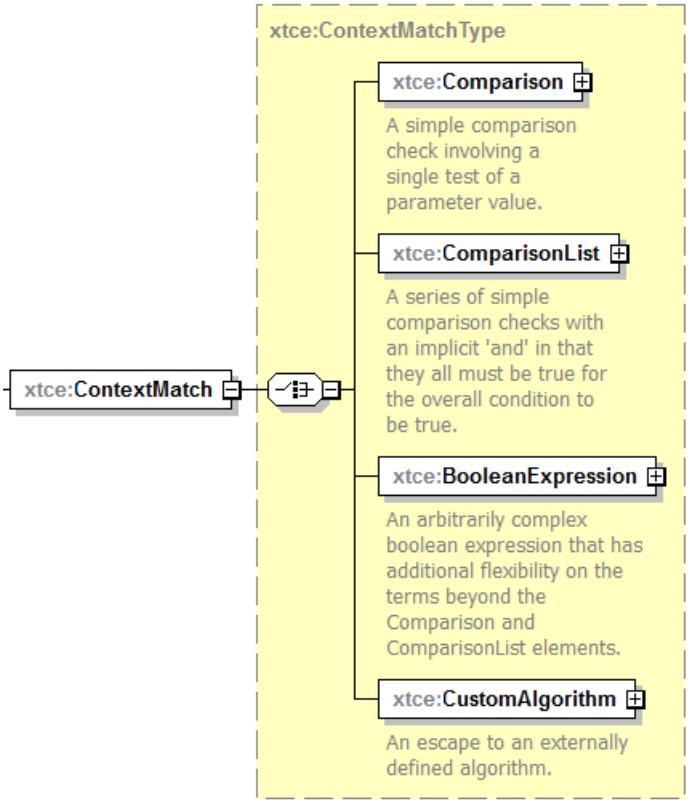



**Figure 4-28: ContextMatch**

As can be seen above, the ContextMatch element of the ContextAlarm is a kind of MatchCriteria (see 3.4.3.6).

#### 4.3.2.3.5 EnumerationAlarmList—EnumerationAlarm

##### 4.3.2.3.5.1 General

The EnumeratedParameterType has its own kind of alarm. EnumerationAlarmLists consists of a list of one or more EnumerationAlarm elements that contain the attributes alarmLevel and enumerationValue.

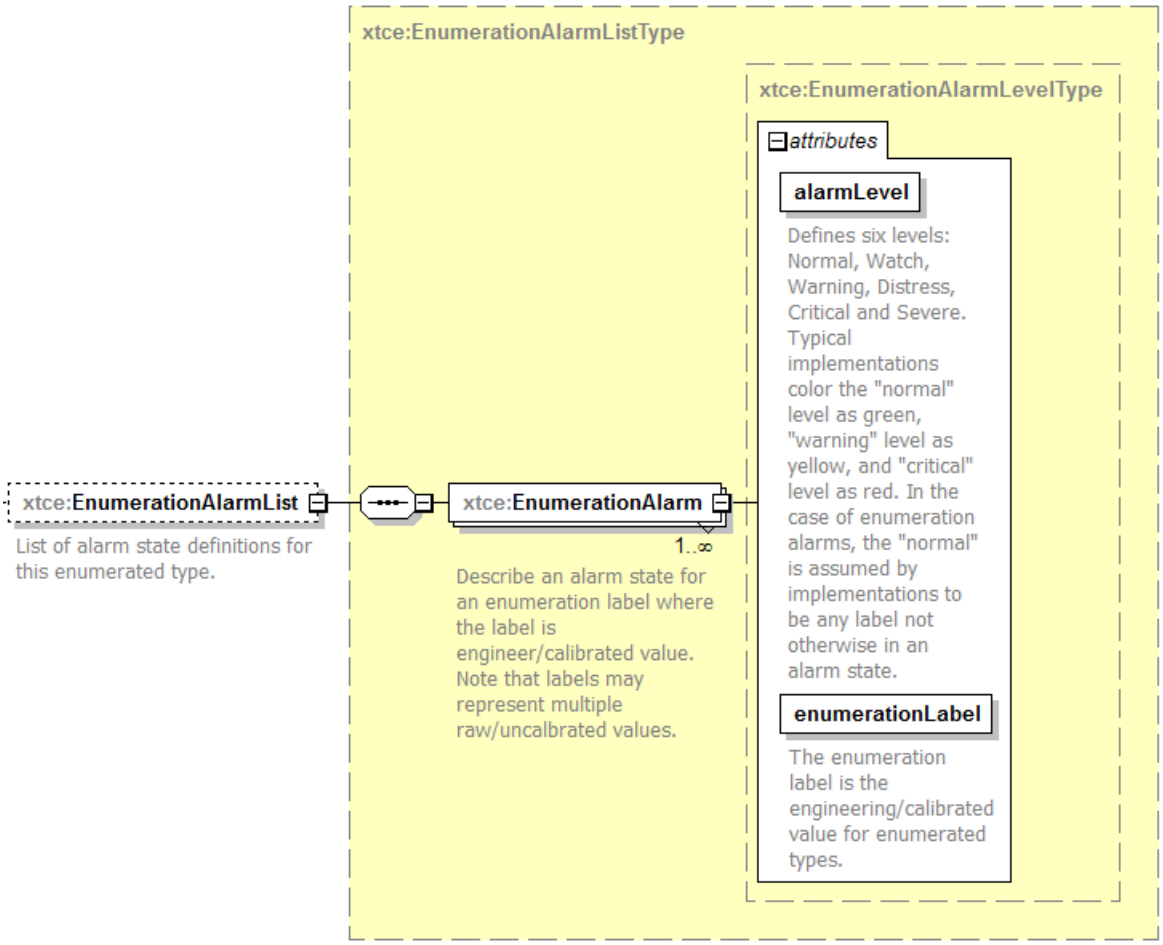



**Figure 4-29: EnumerationAlarm**

#### 4.3.2.3.5.2 alarmLevel Attribute

Alarm levels are one of: ‘normal’, ‘watch’, ‘warning’, ‘distress’, ‘critical’, or ‘severe’.

#### 4.3.2.3.5.3 enumerationValue Attributes

A specific enumeration label is associated with a value. The following is an example of a specified label.

```
<xtce:EnumerationAlarmList>
    <xtce:EnumerationAlarm alarmLevel="warning" enumerationValue="NUM_RANGE_ERR"/>
</xtce:EnumerationAlarmList>
```

#### 4.3.2.3.6 AlarmConditions

The AlarmConditions element appears in many of the ParameterTypes and are used to specify alarm condition levels. Each alarm level is a MatchCriteria. (See 3.4.3.6 for a discussion of the MatchCriteriaType.)

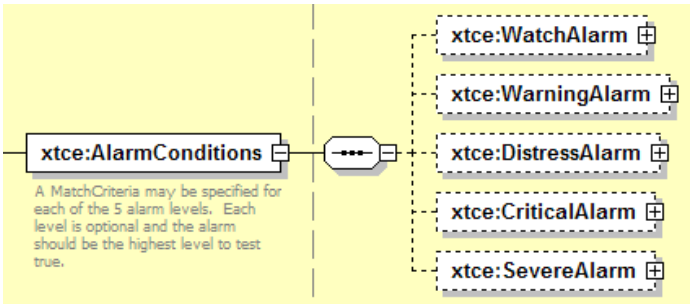



**Figure 4-30: AlarmConditions**

```
<xtce:AlarmConditions>
    <xtce:WarningAlarm>
        <xtce:Comparison parameterRef="BAT1VOLT1" value="12.4"
        comparisonOperator="&gt;"/>
    </xtce:WarningAlarm>
</xtce:AlarmConditions>
```

In this example the calibrated value of parameter instance zero (last recorded value) is compared to 12.4. If it is greater, the alarm level is set to ‘Warning’.

The various levels are optional, but the alarm ‘returns’ the most severe level to test ‘true’.

#### 4.3.2.3.7 Numeric Alarms

##### 4.3.2.3.7.1 General

Numeric alarms only appear in FloatParameterType and IntegerParameterType. The general term ‘Numeric Alarms’ refers to the StaticAlarmRanges, ChangeAlarmRanges, and now in XTCE 1.2, the MultiRangeAlarm elements, which appear in the NumericAlarmType.

##### 4.3.2.3.7.2 StaticAlarmRanges—Fixed Ranges

StaticAlarmRanges describe fixed alarm numeric ranges and appear as follows for the five supported range elements:

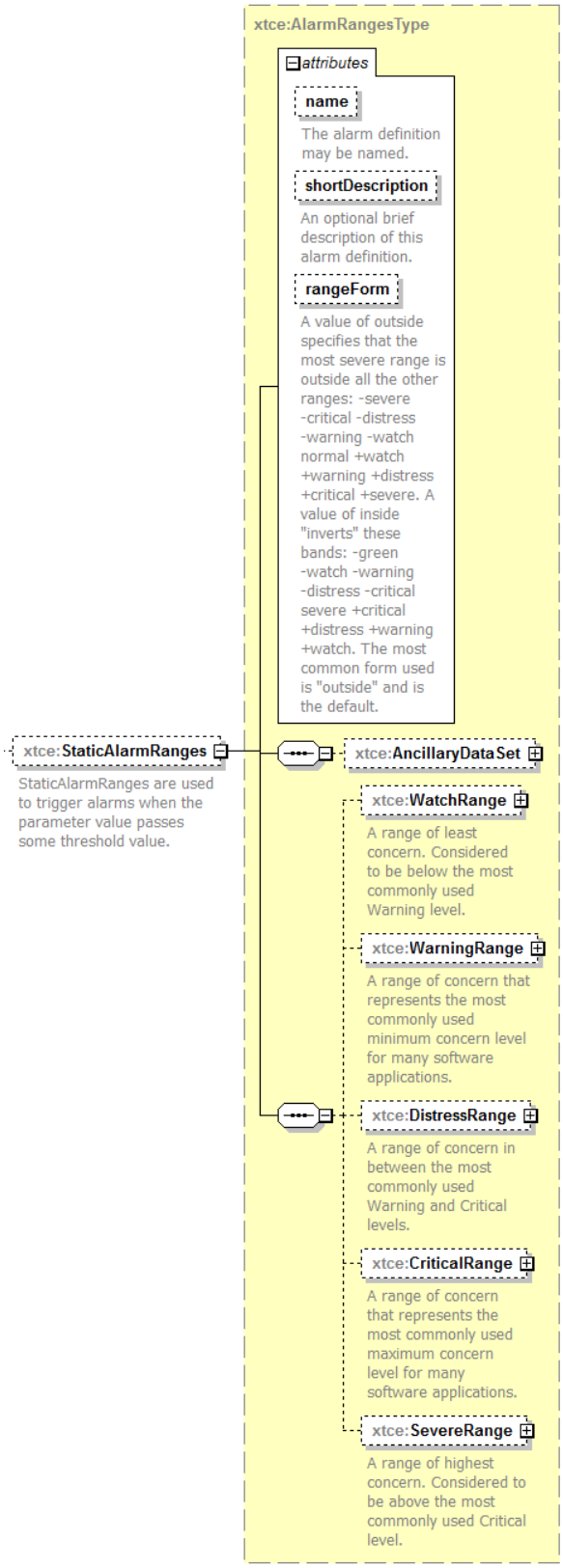



**Figure 4-31: StaticAlarmRanges**

XTCE 1.2 now supports outside and inside alarms.

NOTE – In the previous version of the this report, the term ‘inside alarms’ was used to denote the style of alarm or limit in which the least significant range, the inner range, is ‘green’ or normal. However, the industry is not consistent in this terminology, and some call these ‘outside’ alarms. XTCE 1.2 now describes alarms with the inner most range being green as ‘outside alarms’, adopting the terminology of the weighting of the ‘red’ or most severe range. Given that, it now describes ‘inside alarms’ as being green or normal at the outermost range.

#### 4.3.2.3.7.2.1 rangeForm – Outside Alarms

When the attribute rangeForm is ‘outside’ (the default), the alarms have the normal (green here) range inside the least severe range specified and the most severe (red here) range on the outside. This is illustrated graphically as follows:

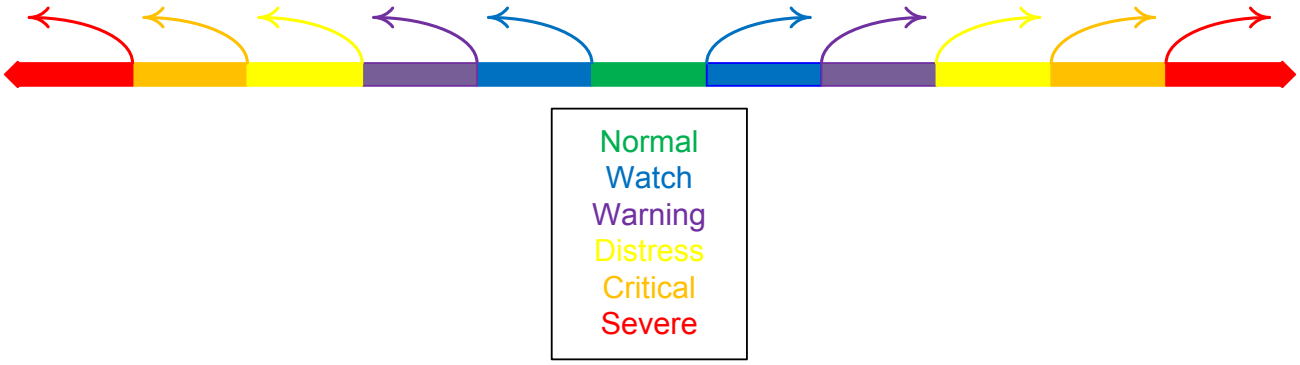



**Figure 4-32: Visual Depiction of Outside Alarm Range Bands**

NOTE – The above colors are only illustrative, although many systems use a three color system: green, yellow, and red, from least to most severe.

RECOMMENDATION – On systems with a tri-color limit scheme (typically green, yellow, and red, as noted), the green range would be Normal; the next highest range, which is typically yellow, would be Warning; and the red range, which is the most severe range, would be Severe in XTCE’s nomeclature.

```
<xtce:StaticAlarmRanges>
    <xtce:WarningRange minInclusive="-5" maxInclusive="5"/>
    <xtce:CriticalRange minInclusive="-20" maxInclusive="20"/>
</xtce:StaticAlarmRanges>
```

In the above example, the ranges are as follows:

– the normal range is $-5 < x < 5$;

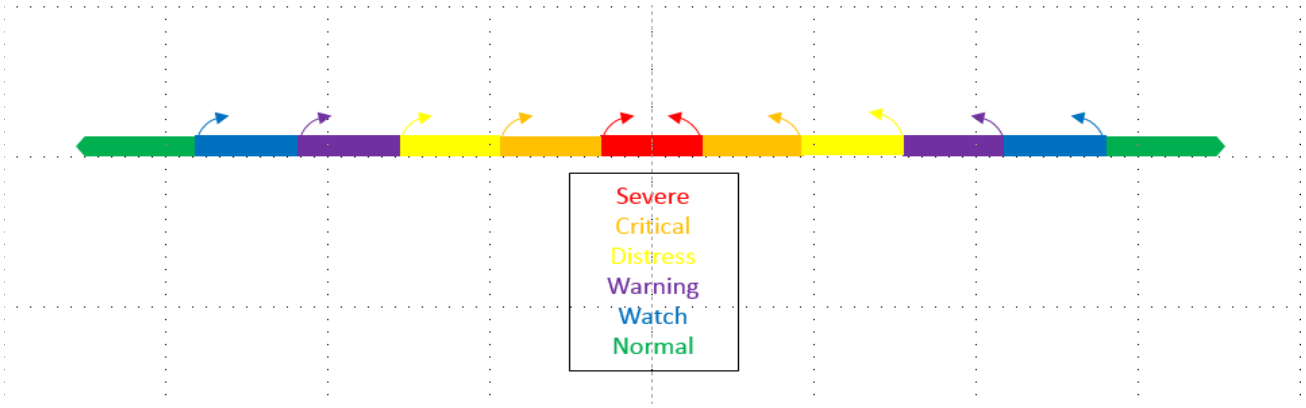
– the warning range is -20 < *x* <= -5 and 5 <= *x* < 20;

– the critical range is *x* <= -20 and *x* >= 20.

In XTCE, the 'min' specification always implies that it 'points' towards -Infinity; the 'max' band always points towards +Infinity.

#### 4.3.2.3.7.2.2 rangeForm – Inside Alarms

If the rangeForm is set to 'inside', the ranges are inverted, and the least severe (unstated) range is on the outside to infinity. The inner bands form the increasingly severe ranges with the innermost band being the most severe.




```
<xtce:StaticAlarmRanges>
    <xtce:WarningRange minInclusive="-25" maxInclusive="25"/>
    <xtce:CriticalRange minInclusive="-2" maxInclusive="2"/>
</xtce:StaticAlarmRanges>
```

In the above example, the ranges are as follows, in order from most severe to least, logically:

– the critical range is −2 <= *x* <= 2;

– the warning range is −25 <= *x* <= 25;

– otherwise the value is normal.

Therefore normal always points away from the checked bands, in that order.

### 4.3.2.3.7.3 ChangeAlarmRanges—Delta Alarm or Change Over Time

#### 4.3.2.3.7.3.1 General

ChangeAlarmRanges is used to describe delta alarms, which compare multiple samples of a parameter to determine if the difference exceeds the change allowed.

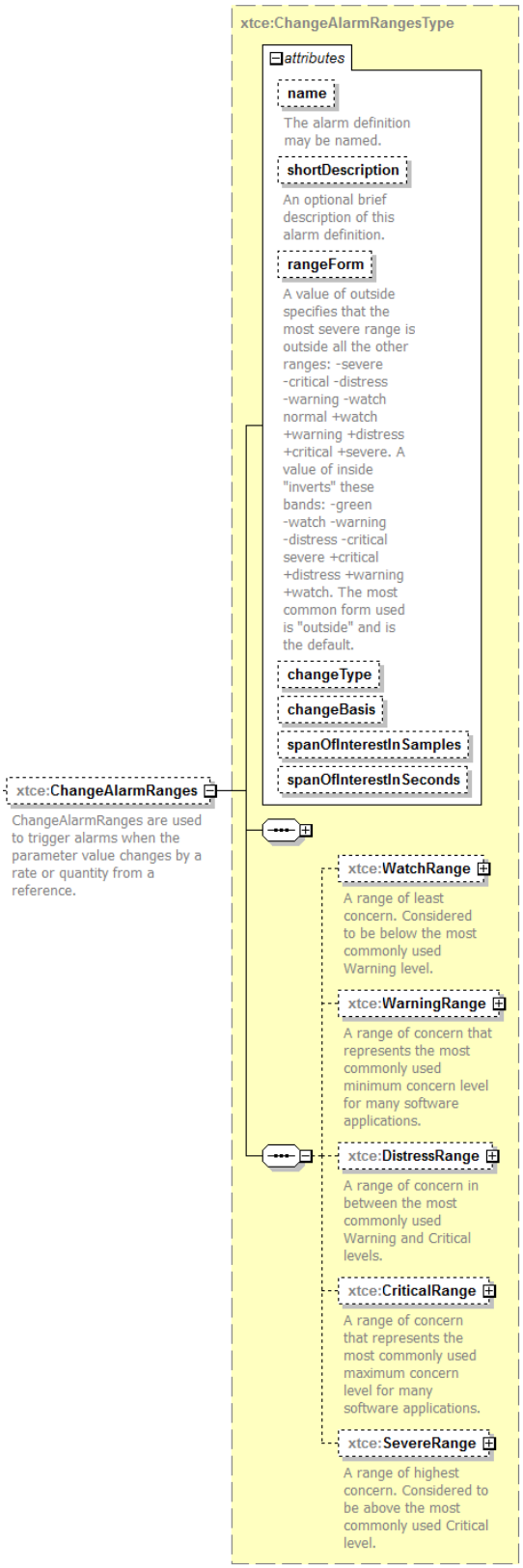



**Figure 4-33: ChangeAlarmRanges**

ChangeAlarmRanges are similar to StaticAlarmRanges, and similar rules apply.

#### 4.3.2.3.7.3.2 changeType Attribute

A value of ‘changePerSample’ designates a delta-style alarm. A value of ‘changePerSecond’ is a rate-style alarm.

#### 4.3.2.3.7.3.3 changeBasis Attribute

If the value is ‘absoluteChange’, the absolute value change between samples is calculated. If the value is set to ‘percentageChange’, the percentage change between samples is calculated.

#### 4.3.2.3.7.3.4 spanOfInterestInSamples Attribute

The default value of ‘one’ indicates that every sample is used in the calculations. If the value is ‘two’, every other sample is used in the calculation. This attribute should be ignored if changeType is set to ‘changePerSecond’.

#### 4.3.2.3.7.3.5 spanOfInterestInSeconds Attribute

The spanOfInterestInSeconds attribute default is ‘zero’. It must be set to a positive value if changeType is set to ‘changePerSecond’.

### 4.3.2.3.7.4 ChangeAlarmRanges—Delta

The ChangeAlarmRanges element forms a delta style alarm when changeType is set to ‘changePerSample’. For example:

```
<xtce:ChangeAlarmRanges changeBasis="absoluteChange" changeType="changePerSample"
spanOfInterestInSamples="1">
    <xtce:SevereRange maxInclusive="10" minInclusive="-10"/>
</xtce:ChangeAlarmRanges>
```

In this example, the alarm ranges are the following per sample:

– normal range: -10 < *x* < 10;

– severe range: *x* <= -10 and *x* >= 10.

### 4.3.2.3.7.5 ChangeAlarmRanges—Rate of Change

The ChangeAlarmRanges element can also be used to detect rates of change. By slightly varying the attributes, the delta alarm can be changed to detect rate of change. The following example shows a percentage rate of change:

```
<xtce:ChangeAlarmRanges changeBasis="percentageChange" changeType="changePerSecond"
spanOfInterestInSeconds="1">
    <xtce:SevereRange maxInclusive="10" minInclusive="-10"/>
</xtce:ChangeAlarmRanges>
```

In this example, the alarm ranges are the following per second:

– normal range: -10% < *x* < 10%;

– severe range: *x* <= -10% and *x* >= 10%.

#### 4.3.2.3.7.6 ChangeAlarm rangeForm—Outside/Inside

The rangeForm attribute is available in the ChangeAlarm element. Inside, the default, and outside here have a consistent interpretation as the numeric ranges.

### 4.3.2.3.8 MultiRange

New for XTCE 1.2, the numeric multi-range allows one to set any number of ranges in different forms. They would be processed so that the most severe range is the level of the alarm.

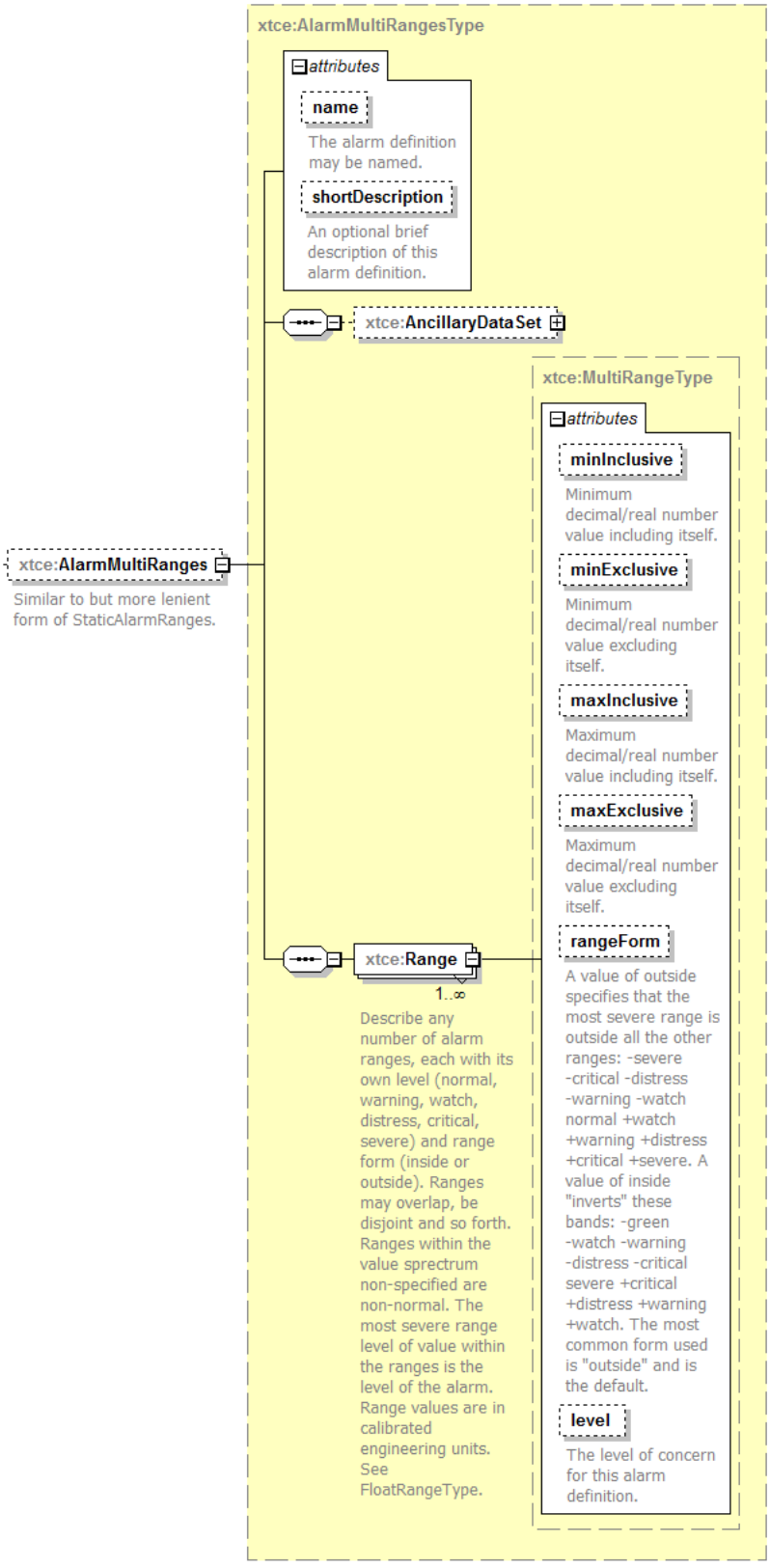



**Figure 4-34: AlarmMultiRanges**

This is a very general kind of alarm definition because the number of ranges is unlimited, and they may overlap, be of the same level, and so forth.

#### 4.3.2.3.8.1 AlarmMultiRange—rangeForm

In this element, rangeForm is set to inside; min and max are interpreted as either:

- [min, max];
- (min, max);
- [min, max);
- (min, max].

In this case (above), the rangeForm should be set to the opposite of its default, to inside.

For the value of outside, the ranges are interpreted as:

- (-inf, min];
- (-inf, min);
- [max, inf);
- (max, inf);

where in each case, the notation of () means exclusive, and [] means inclusive.

Inside, the default, and outside here have a consistent interpretation as the numeric ranges.

### 4.3.2.3.9 ValidRange and Alarms

Although this element is not part of numeric alarms, it is worth considering briefly because it is often a point of confusion as to its nature and relation to them.

The valid range, which is optional, captures the legal range of values for this numeric DataEncoding after calibration (typically).

It has in some sense nothing directly to do with alarms, which are tests against bands within this overall range.

An example of this might be something like a speedometer.

Perhaps a speedometer is defined from to show speeds from 0 to 200. This would be considered its valid range.

But a particular car model that uses that speedometer may simply not be designed to go those speeds, in which case alarm ranges may be set within that overall range for various levels.

Given that, values outside of this range are marked as invalid in relation to the speedometer, but are not really alarmed either (although this is an implementation-specific issue).

Likewise, the kind of alarm does not really affect the valid range or whether it is an inside or outside alarm, and so forth.

#### 4.3.2.4 ParameterTypes

##### 4.3.2.4.1 General

XTCE has ten ParameterTypes that are discussed in the subsections below, within two groups. In the first group, the scalar ParameterTypes are: String, Enumerated, Binary, Boolean, Integer, Float, and String. In the second group, the complex types are: Array and Aggregate (‘struct’).

##### 4.3.2.4.2 StringParameterType

###### 4.3.2.4.2.1 General

StringParameterType describes strings. The DataEncoding will normally be StringDataEncoding when using this ParameterType if it is telemetered. If the syntax diagram is large, it will split into several sections. The first part includes name, descriptions, and other familiar attributes and elements.

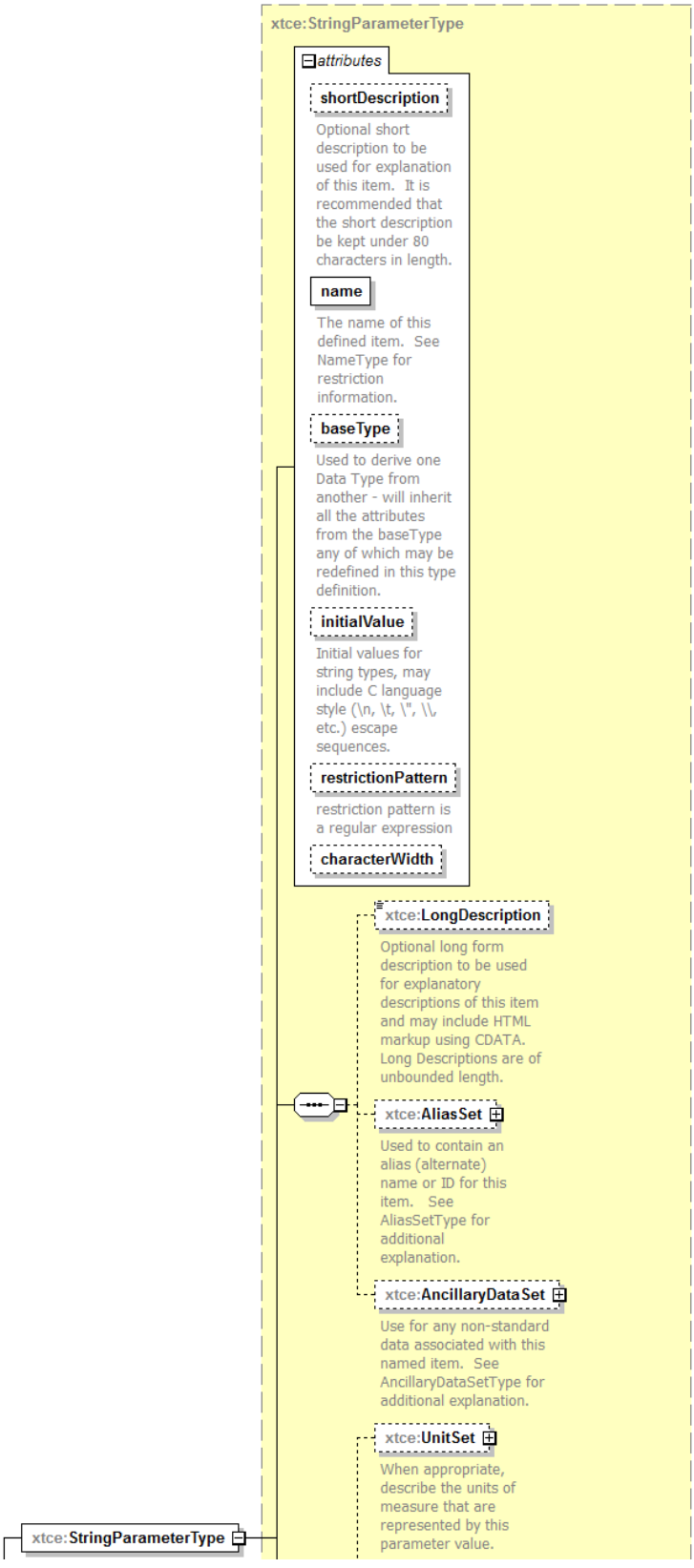



**Figure 4-35: StringParameterType—Part 1**

The next part of StringParameterType is the DataEncoding section through the SizeRangeInCharacters.

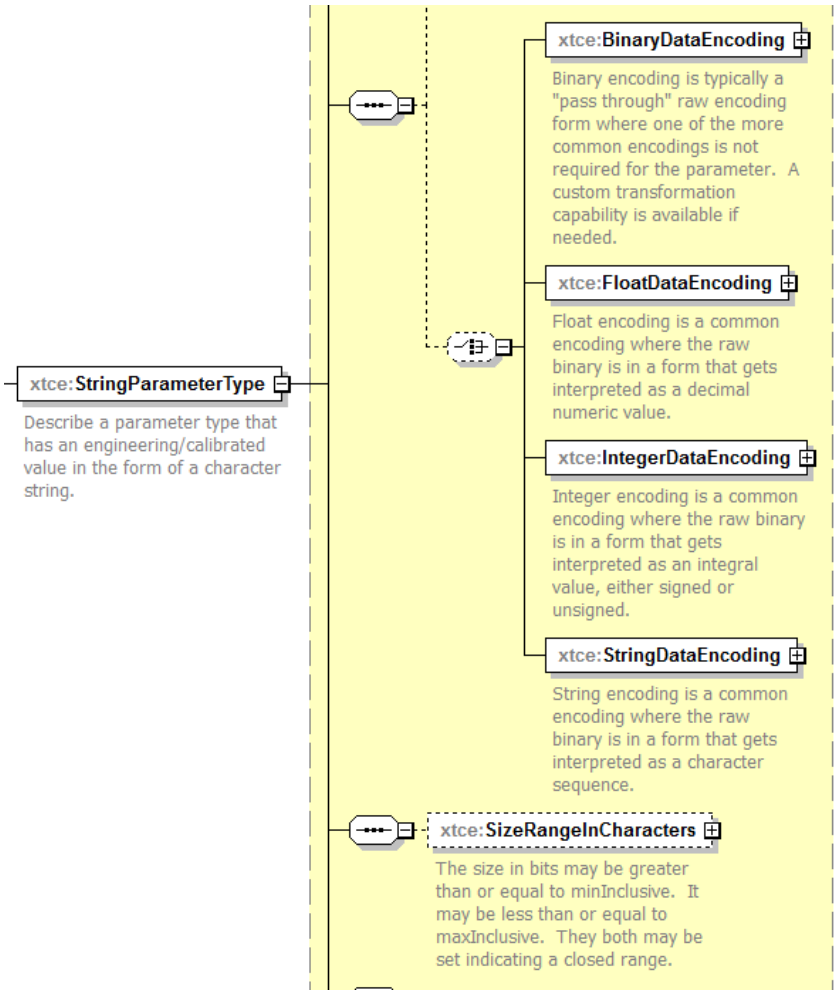



**Figure 4-36: StringParameterType—Part 2**

Like all scalar ParameterTypes, StringParameterType has four DataEncodings. However, StringDataEncoding is typically used in this element and will be discussed further below.

Finally, the last part of StringParameterType are the alarms.

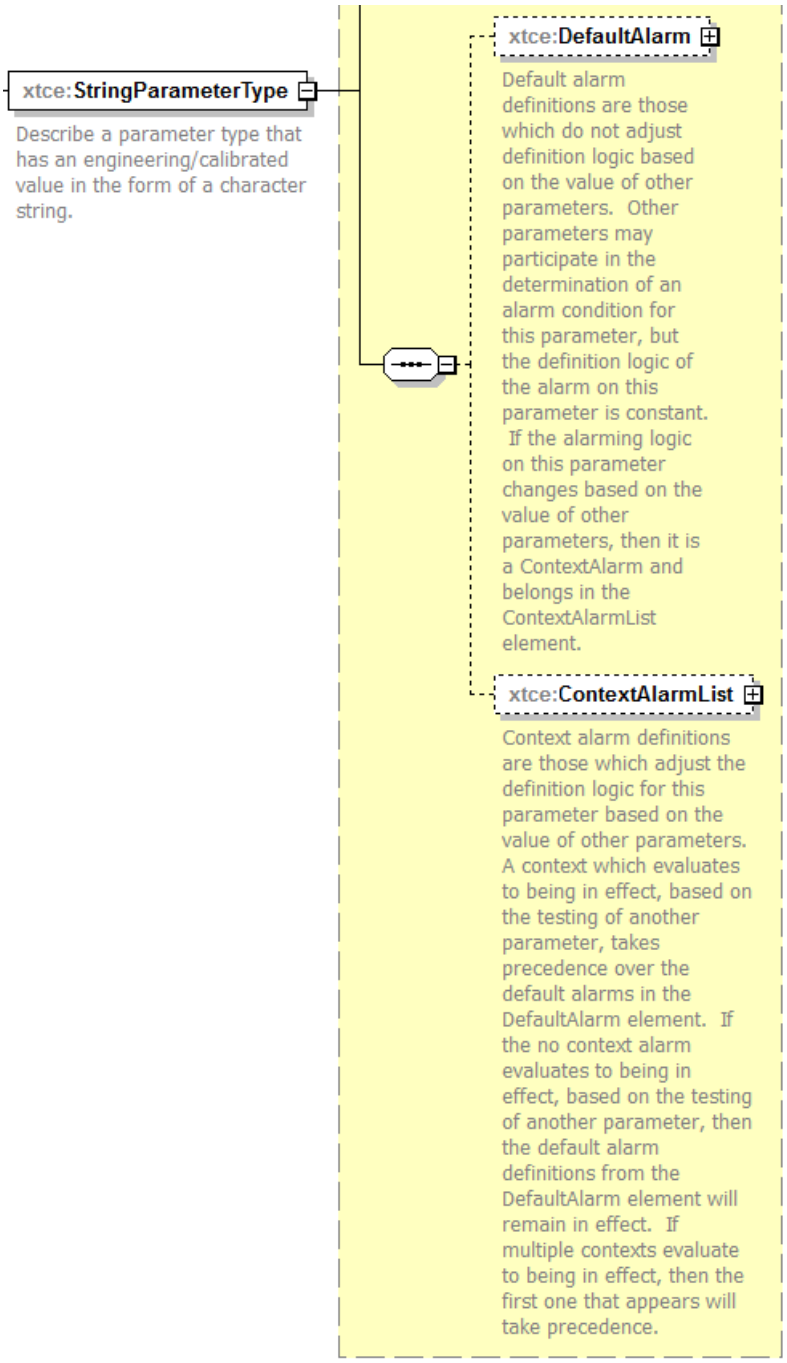



**Figure 4-37: StringParameterType—Part 3**

StringParameterType specific alarms will be discussed below.

#### 4.3.2.4.2.2 Name, ShortDescription, BaseType, LongDescription, AliasSet, and AncillaryData

These attributes and elements have been described in other sections below.

#### 4.3.2.4.2.3 initialValue Attribute

The initialValue attribute uses \u for Unicode escapes characters.

#### 4.3.2.4.2.4 RestrictionPattern Attribute

The RestrictionPattern attribute is used to restrict the string with this regular expression.

#### 4.3.2.4.2.5 CharacterWidth Attribute

If the CharacterWidth attribute is unspecified, it is assumed the ParameterType can hold the entire Unicode set. Otherwise, the number of bits supported is specified.

#### 4.3.2.4.2.6 UnitSet

UnitSet is discussed in 4.3.2.2.4.

#### 4.3.2.4.2.7 StringDataEncoding Element

##### 4.3.2.4.2.7.1 General

For most applications, a StringParameterType will have a StringDataEncoding element. If the StringDataEncoding element cannot describe the string encoding properly, BinaryDataEncoding can be used. FloatDataEncoding and IntegerDataEncoding are not explicitly defined for StringParameterType.

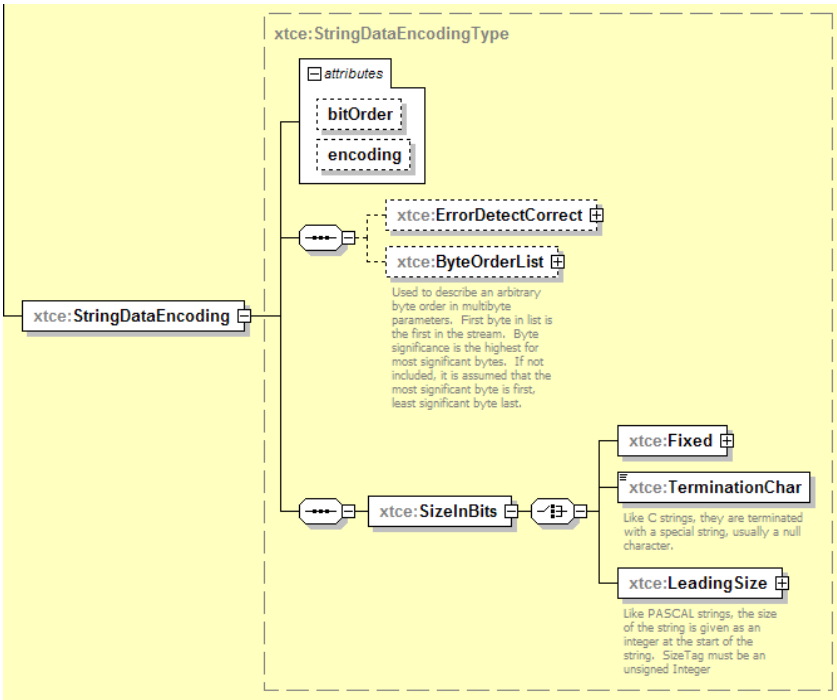



**Figure 4-38: StringDataEncoding**

#### 4.3.2.4.2.7.2 Encoding Attribute

The options are now:

- US-ASCII;
- ISO-8859-1;
- Windows-1252;
- UTF-8;
- UTF-16;
- UTF-16LE;
- UTF-16BE;
- UTF-32;
- UTF-32LE;
- UTF-32BE.

NOTE – UTF-8 and UTF-16 may be represented by multi-unit sequences that must be accounted for in the size. Likewise, some have byte order marks or other meta data that should be accounted for in the size.

#### 4.3.2.4.2.7.3 SizeInBits Attribute

Subsection 4.3.2.2.5.5 contains a description of the SizeInBits attribute.

### 4.3.2.4.2.8 SizeRangeInCharacters

SizeRangeInCharacters is used to clip the character set. The values are given as integers in hex, binary, or octal representation. The Unicode range is specified. The processing system will convert to the specified encoding: UTF-8 or UTF-16. In the example below, the character range is clipped to ASCII.

```
<xtce:SizeRangeInCharacters minInclusive="0x00" maxInclusive="0x7F"/>
```

#### 4.3.2.4.2.9 StringAlarm Element

##### 4.3.2.4.2.9.1 General

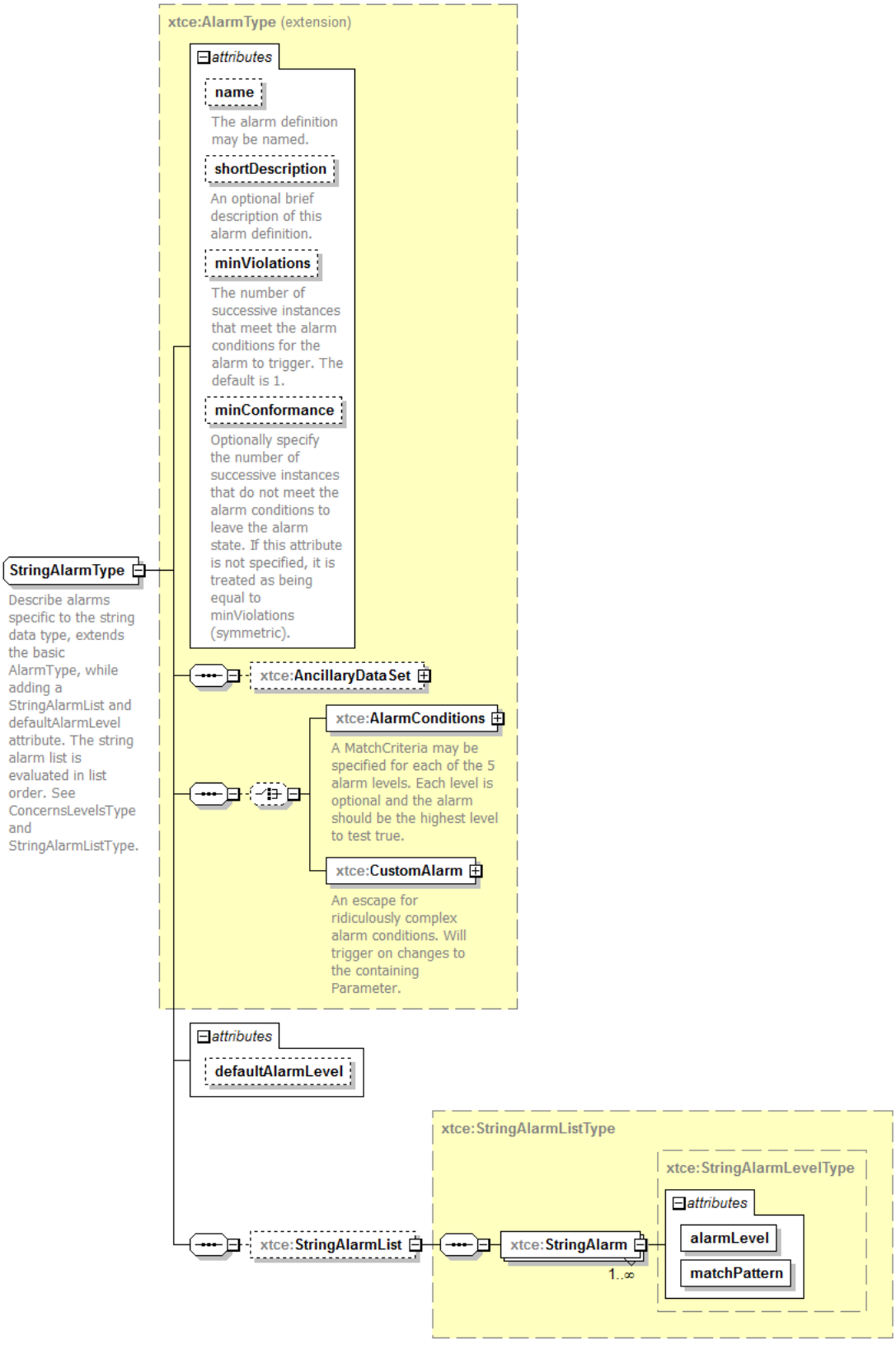



**Figure 4-39: StringAlarm Element**

The StringAlarm element is used to set an alarm against an associated string pattern.

#### 4.3.2.4.2.9.2 minViolations Attribute

The minViolations attribute identifies the number of successive instances that meet the alarm conditions for the alarm to trigger.

#### 4.3.2.4.2.9.3 minConformance

(See 4.3.2.3.3.2.)

#### 4.3.2.4.2.9.4 defaultAlarmLevel Attribute

The defaultAlarmLevel attribute identifies the alarm level for strings not matched in the StringAlarmList. In the following example, the StringParameterType is restricted to strings that start with 'REDALERT'. In this case, the *critical* alarm regarding 'SHIELDS DOWN TO' is ignored three times. Any other 'REDALERT' string defaults to simply 'warning' level.

```
<xtce:StringParameterType name="" restrictionPattern="^REDALERT">
    <xtce:UnitSet/>
    <xtce:StringDataEncoding>
        <xtce:SizeInBits>
            <xtce:Fixed>
                <xtce:FixedValue>40</xtce:FixedValue>
            </xtce:Fixed>
        </xtce:SizeInBits>
    </xtce:StringDataEncoding>
    <xtce:DefaultAlarm minViolations="3" defaultAlarmLevel="warning">
        <xtce:StringAlarmList>
            <xtce:StringAlarm alarmLevel="critical" matchPattern="SHIELDS DOWN TO"/>
        </xtce:StringAlarmList>
    </xtce:DefaultAlarm>
</xtce:StringParameterType>
```

### 4.3.2.4.3 EnumeratedParameterType

#### 4.3.2.4.3.1 General

EnumeratedParameterType supports the description of enumerations, which are a list of values and their associated labels.

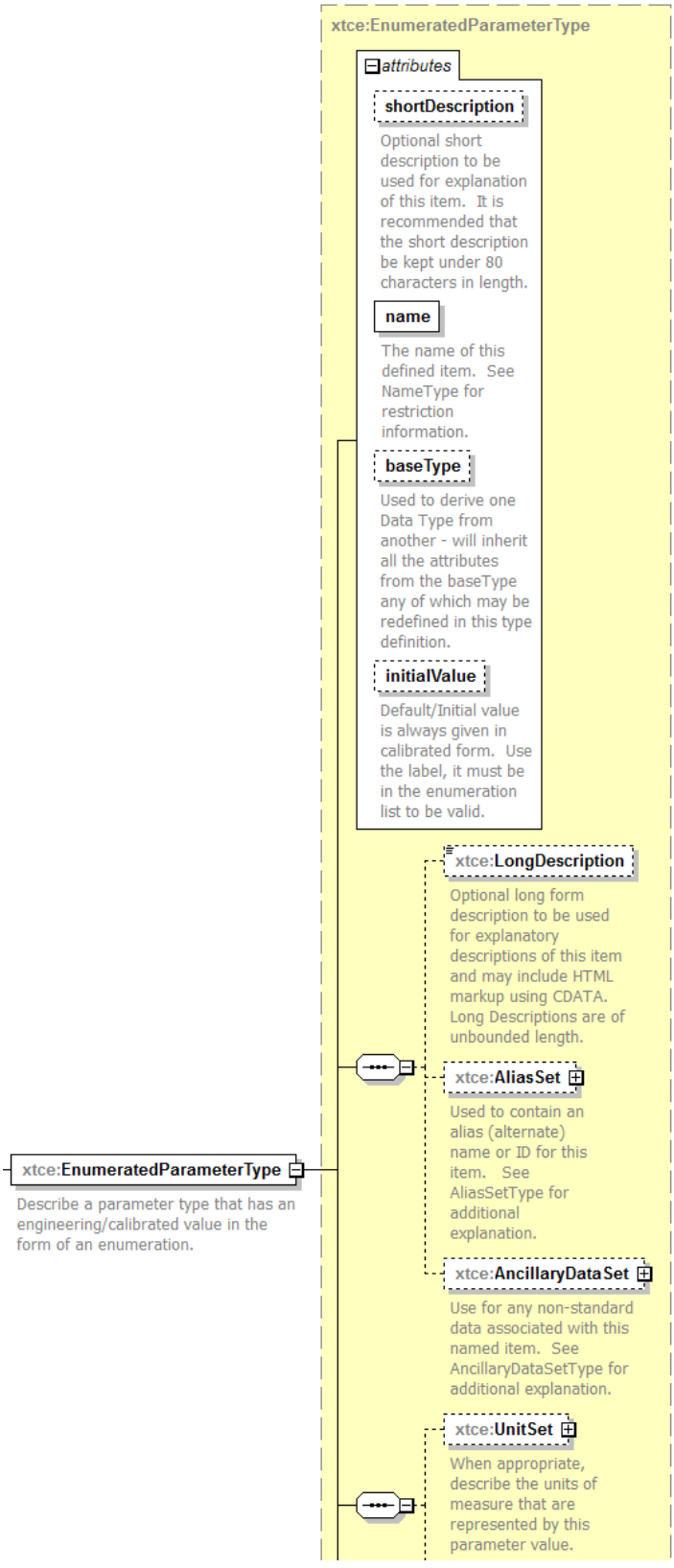



**Figure 4-40: EnumeratedParameterType—Part 1**

The first part of EnumeratedParameterType looks similar to the other scalar ParameterTypes. InitialValue is slightly specialized, though, which is discussed below.

The EnumeratedParameterType continues with the standard DataEncodings.

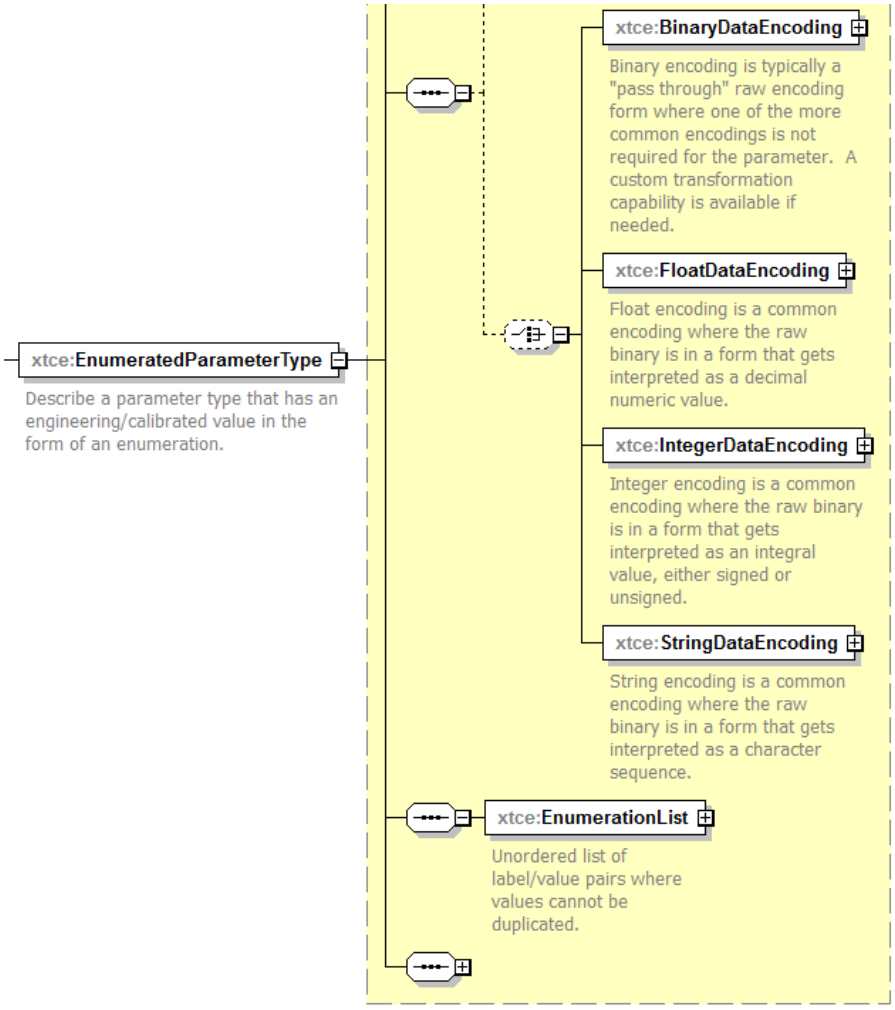



**Figure 4-41: EnumeratedParameter—Part 2**

The standard DataEncodings are then followed by the EnumerationList, which is also discussed below.

Finally, the alarms complete the EnumeratedParameterType.

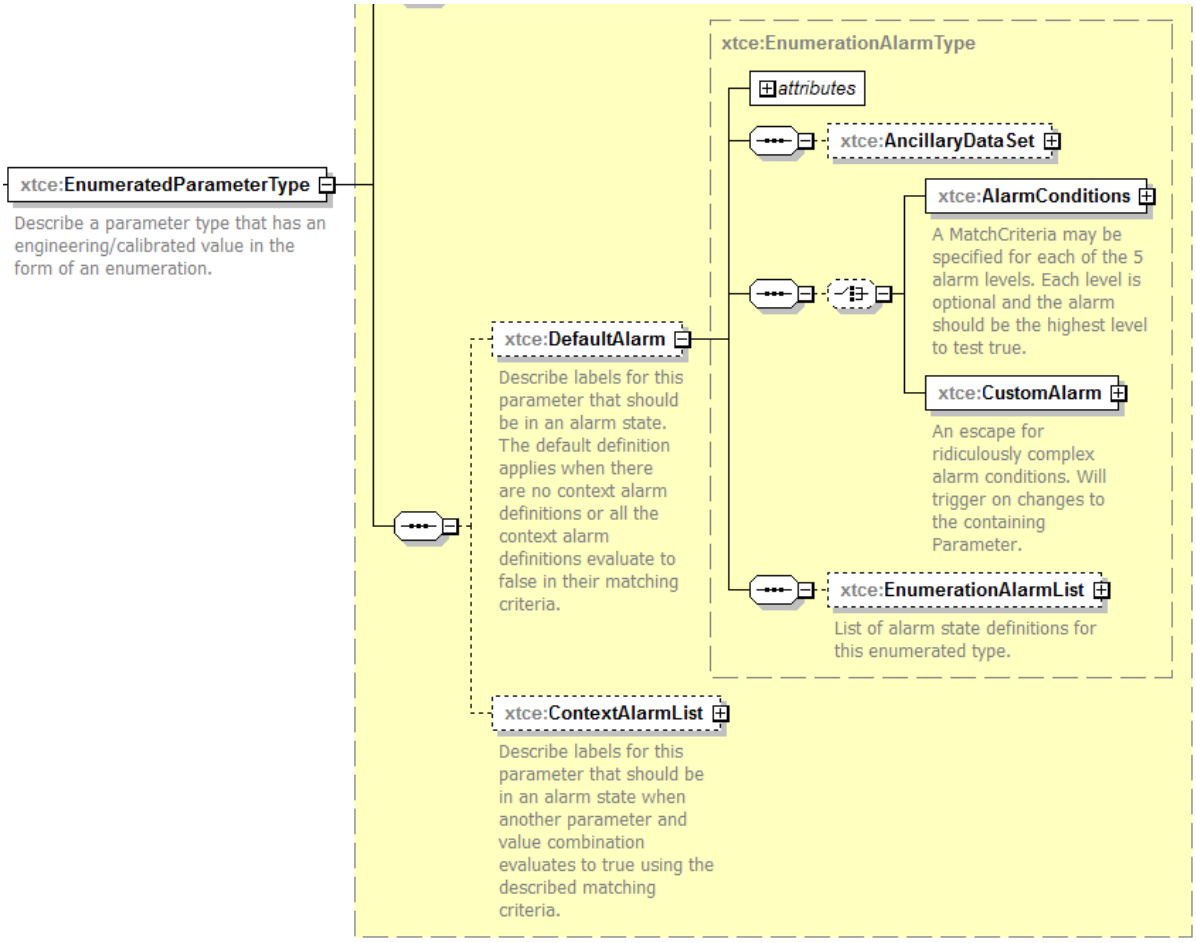



**Figure 4-42: EnumeratedParameterType—Part 3**

EnumeratedParameterType has a specialized alarm in the EnumerationAlarmList, and this is discussed below as well.

#### 4.3.2.4.3.2 Name, ShortDescription, BaseType, LongDescription, AliasSet, and AncillaryData

These attributes and elements have been described in other sections below.

#### 4.3.2.4.3.3 initialValue Attribute

The intialValue attribute is set to one of the defined labels. The mapping of values to labels is defined by the implementation.

#### 4.3.2.4.3.4 UnitSet

UnitSet is discussed in 4.3.2.2.4.

#### 4.3.2.4.3.5 IntegerDataEncoding Element

Enumerated telemetered values are most often integers. Therefore the IntegerDataEncoding is likely to be used in the EnumeratedParameterType. They can be calibrated or uncalibrated.

(See 4.3.2.2.5.6 for more information on IntegerDataEncoding.)

#### 4.3.2.4.3.6 EnumerationList/Enumeration Element

The EnumerationList/Enumeration element specifies the value/label pairs of the enumeration.

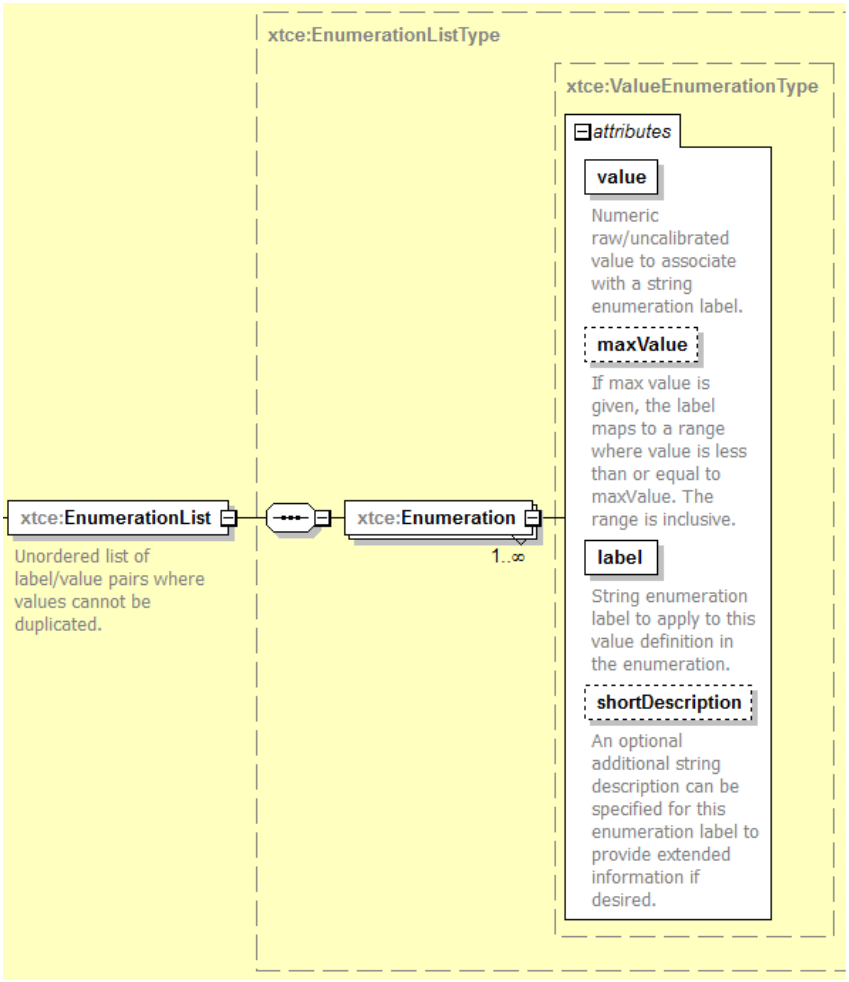



**Figure 4-43: EnumerationList**

#### 4.3.2.4.3.7 Value Attribute

The value attribute is the integer value associated with the label.

#### 4.3.2.4.3.8 MaxValue Attribute

The maxValue attribute is optional. If used, the value and maxValue attributes form a range associated with the label.

#### 4.3.2.4.3.9 Label Attribute

The label is the text representation of the value or range.

#### 4.3.2.4.3.10 ShortDescription Attribute

ShortDescription has been added in XTCE 1.2 to further describe each label/value pair.

An example is as follows.

```
<xtce:EnumerationList>
    <xtce:Enumeration label="OFF" value="1"/>
    <xtce:Enumeration label="ON" value="2"/>
    <xtce:Enumeration label="TRIPPED" value="3"/>
    <xtce:Enumeration label="NOTWORKING" value="4"/>
</xtce:EnumerationList>
```

#### 4.3.2.4.3.11 EnumerationAlarm Element

##### 4.3.2.4.3.11.1 General

The alarm elements of EnumeratedParameterType are similar to the other base alarms in the other ParameterType, except the EnumerationAlarm element, which specifies the alarm level and enumeration value to trip the alarm.

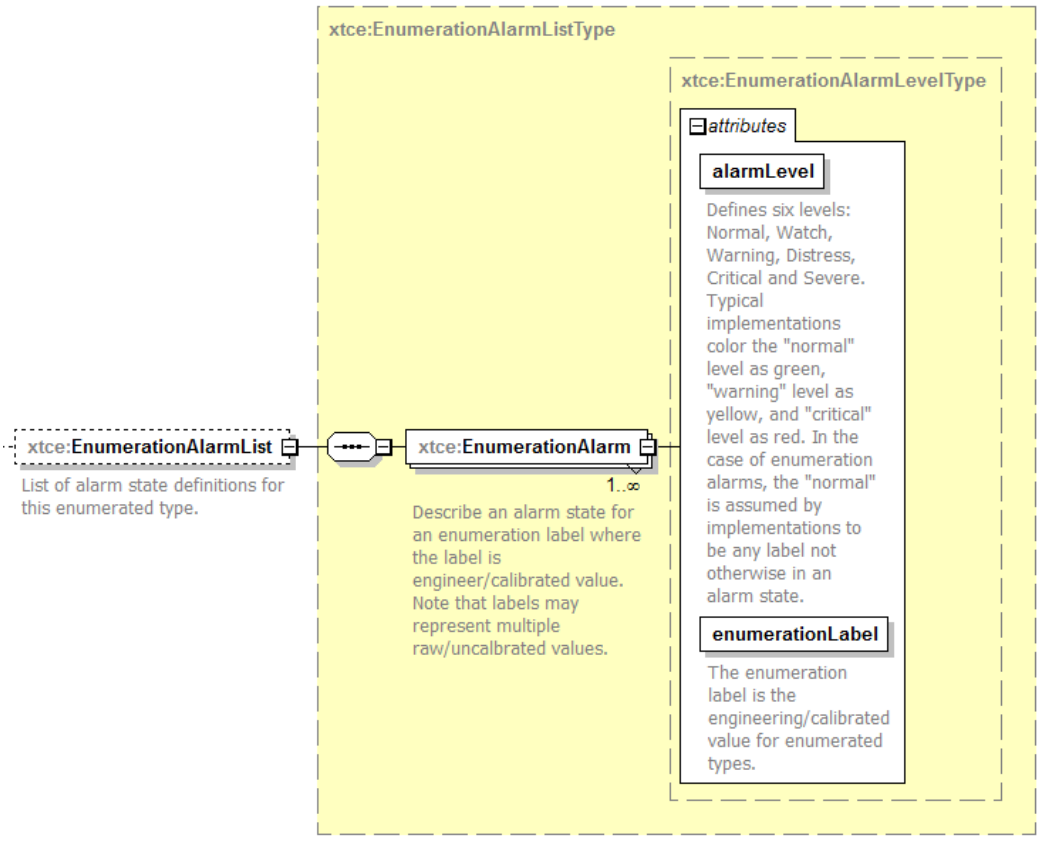



**Figure 4-44: EnumerationAlarm Details**

The enumerationValue is a string.

```
<xtce:EnumerationAlarmList>
    <xtce:EnumerationAlarm alarmLevel="distress" enumerationValue="TRIPPED"/>
</xtce:EnumerationAlarmList>
```

#### 4.3.2.4.3.11.2 minViolations Attribute

The minViolations attribute refers to the number of successive instances that meet the alarm conditions for the alarm to trigger.

#### 4.3.2.4.3.11.3 defaultAlarmLevel Attribute

The defaultAlarmLevel attribute indicates the alarm level for labels/values not matched in the EnumerationAlarmList.

#### 4.3.2.4.4 IntegerParameterType

##### 4.3.2.4.4.1 General

IntegerParameterType describes integer telemetry value data types if it is telemetered, or other local data types if the DataEncoding is not present (and the corresponding Parameter/ParameterProperties/dataSource is set).

As is shown below, the IntegerParameterType is similar to the scalars in construction overall.

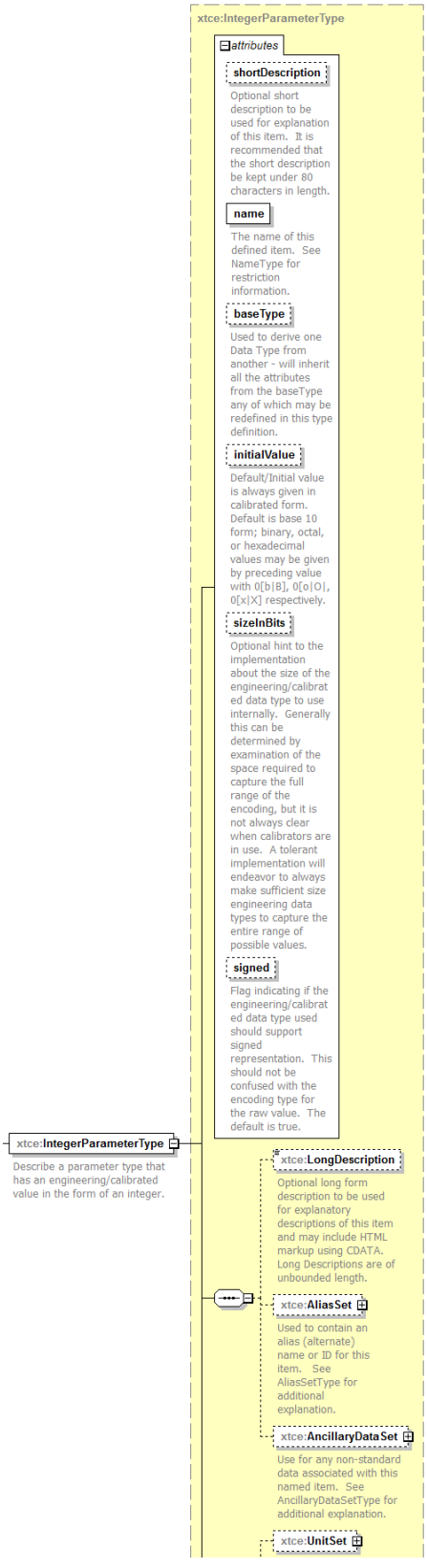



**Figure 4-45: IntegerParameterType—Part 1**

Part 2 shows the standard DataEncodings.

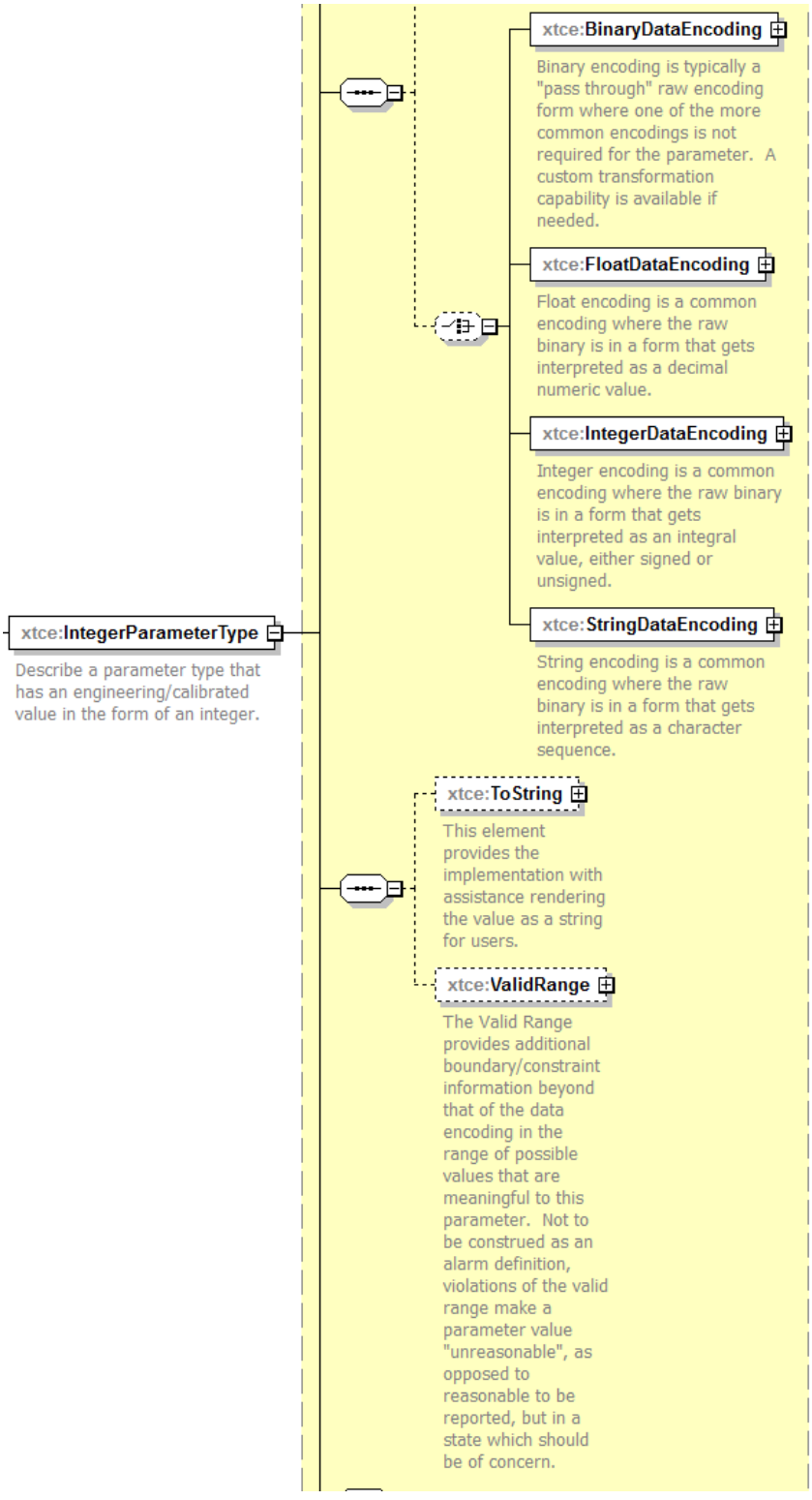



**Figure 4-46: IntegerParameterType—Part 2**

And specific to the ‘numerics’ are the ToString and ValidRange elements, which are discussed below.

Finally, Part 3 includes the numeric alarms.

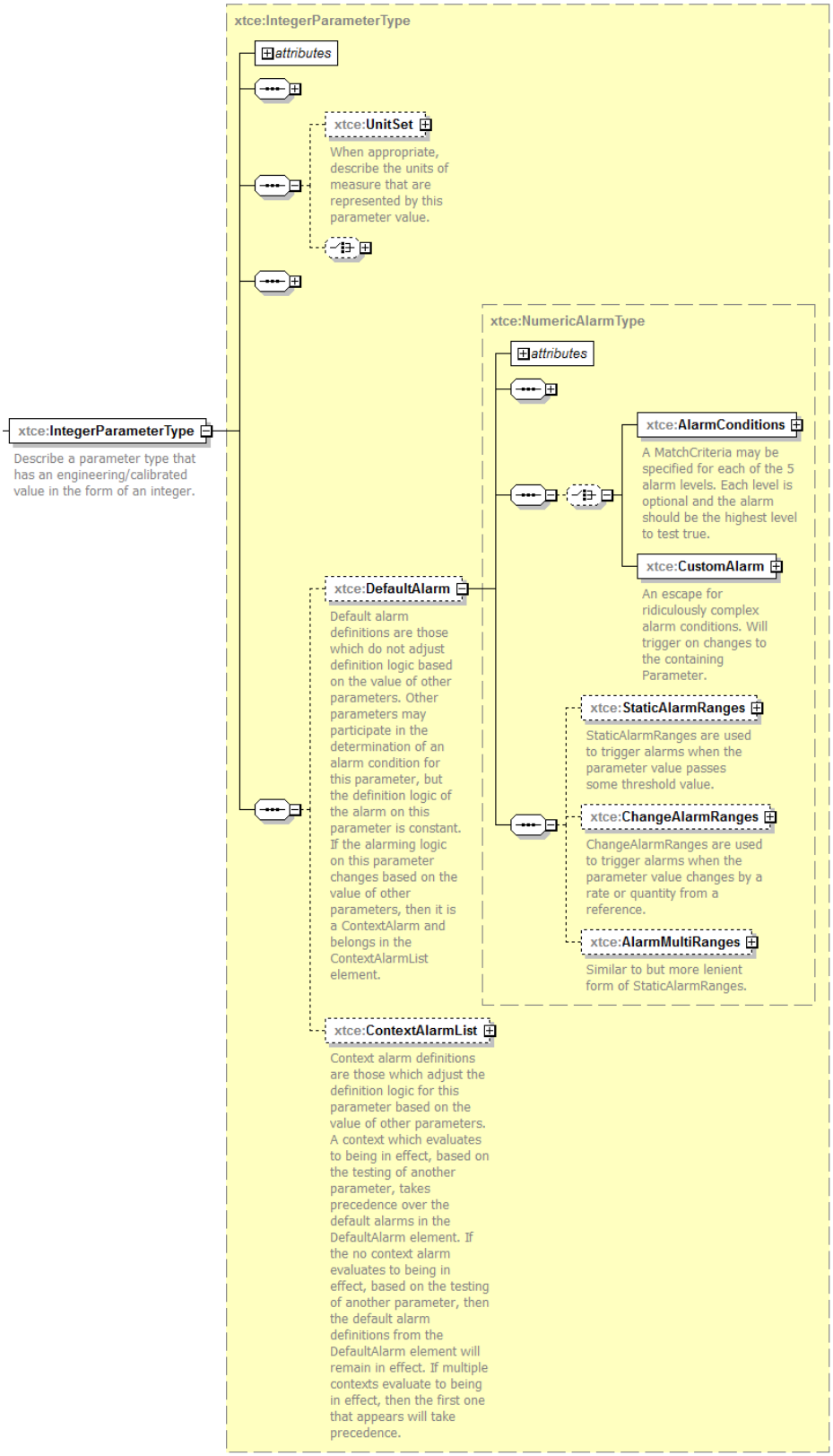



**Figure 4-47: IntegerParameterType—Part 3**

The specific alarms to this ParameterType are: StaticAlarmRange, ChangeAlarmRange, and AlarmMultiRanges. These are discussed below.

#### 4.3.2.4.4.2 validRangeAppliesToCalibrated Attribute

ValidRange occurs on the raw values, before Calibration (4.3.2.2). The valid range itself is defined in the element ValidRange (4.3.2.4.4.7).

#### 4.3.2.4.4.3 sizeInBits Attribute

The IntegerParameterType's sizeInBits attribute should match or exceed the precision needed to capture the DataEncoding.

NOTE – This attribute has caused a lot of confusion. It is not the number of bits in the telemetered item. That is the DataEncoding/SizeInBits. Instead, this says something about the range of the data types the local implementation needs to support. It is something of a hint since XTCE has no way to enforce implementation details. As a general rule, matching this value or exceeding the one in the DataEncoding is good practice.

#### 4.3.2.4.4.4 signed Attribute

The signed attribute is set to 'unsigned' if encoding is unsigned (default is 'signed').

#### 4.3.2.4.4.5 IntegerDataEncoding Element

Most IntegerParameterTypes will select IntegerDataEncoding (uncalibrated or calibrated). BinaryDataEncoding is also an option. StringDataEncoding and FloatDataEncoding are undefined for IntegerParameterType, and their use would be user dependent.

#### 4.3.2.4.4.6 ToString Element

The ToString Element is one of the few places in XTCE that has any information regarding display. It is used to print an integer on the screen.

#### 4.3.2.4.4.7 ValidRange Element

For telemetry, the valid range supplies an overall possible range of value in question. It is not the size in bits of the data type necessarily.

One example might be something like a speedometer, in which the other valid range exceeds a particular model of car using it. The speedometer's range would be the ValidRange in this scenario. It also is not an alarm.

#### 4.3.2.4.4.8 StaticAlarmRanges Element

The StaticAlarmRanges element is used to define numeric based bands of increasing levels of severity. Inside and outside versions are supported. (See 4.3.2.3.7.2 for a discussion of static ranges.)

#### 4.3.2.4.4.9 ChangeAlarm Element

The ChangeAlarm element is used to describe delta or change in time alarms. Inside and outside versions are supported. (See 4.3.2.3.7.3.)

#### 4.3.2.4.4.10 AlarmMultiRange

AlarmMultiRange is a very general kind of alarm where any number of range sets can be defined with any level (even duplicates and overlap). Inside and outside forms are supported for each individual range. This alarm is more fully discussed in the numeric alarm section below.

### 4.3.2.4.5 BinaryParameterType

#### 4.3.2.4.5.1 General

BinaryParameterType is a general-purpose ‘data type-less’ type that is used to describe a block of bits that cannot be captured with other ParameterTypes.

The constructions follows the pattern of the other ParameterTypes.

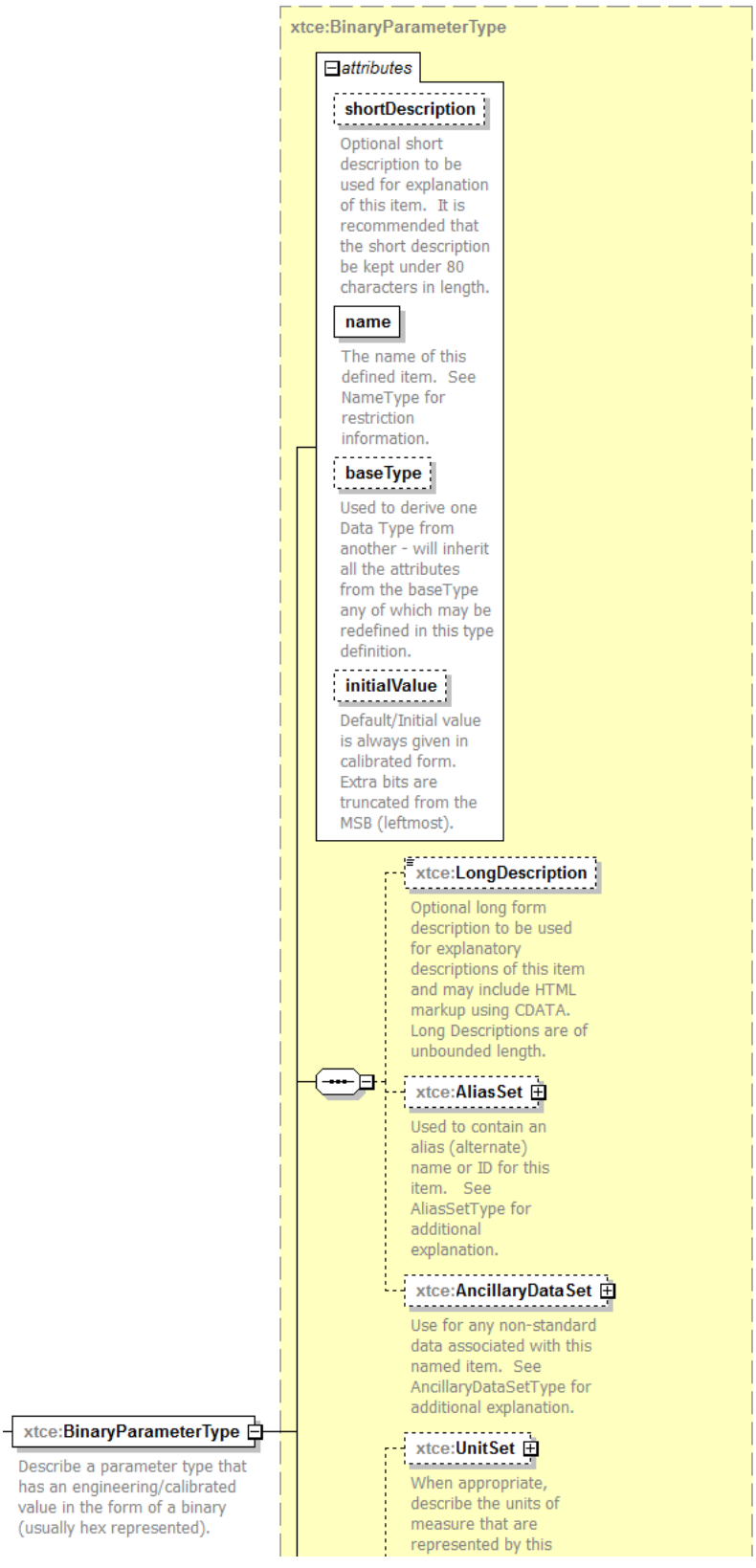



**Figure 4-48: BinaryParameterType—Part 1**

The second half shown below includes the four data encodings, and alarms.

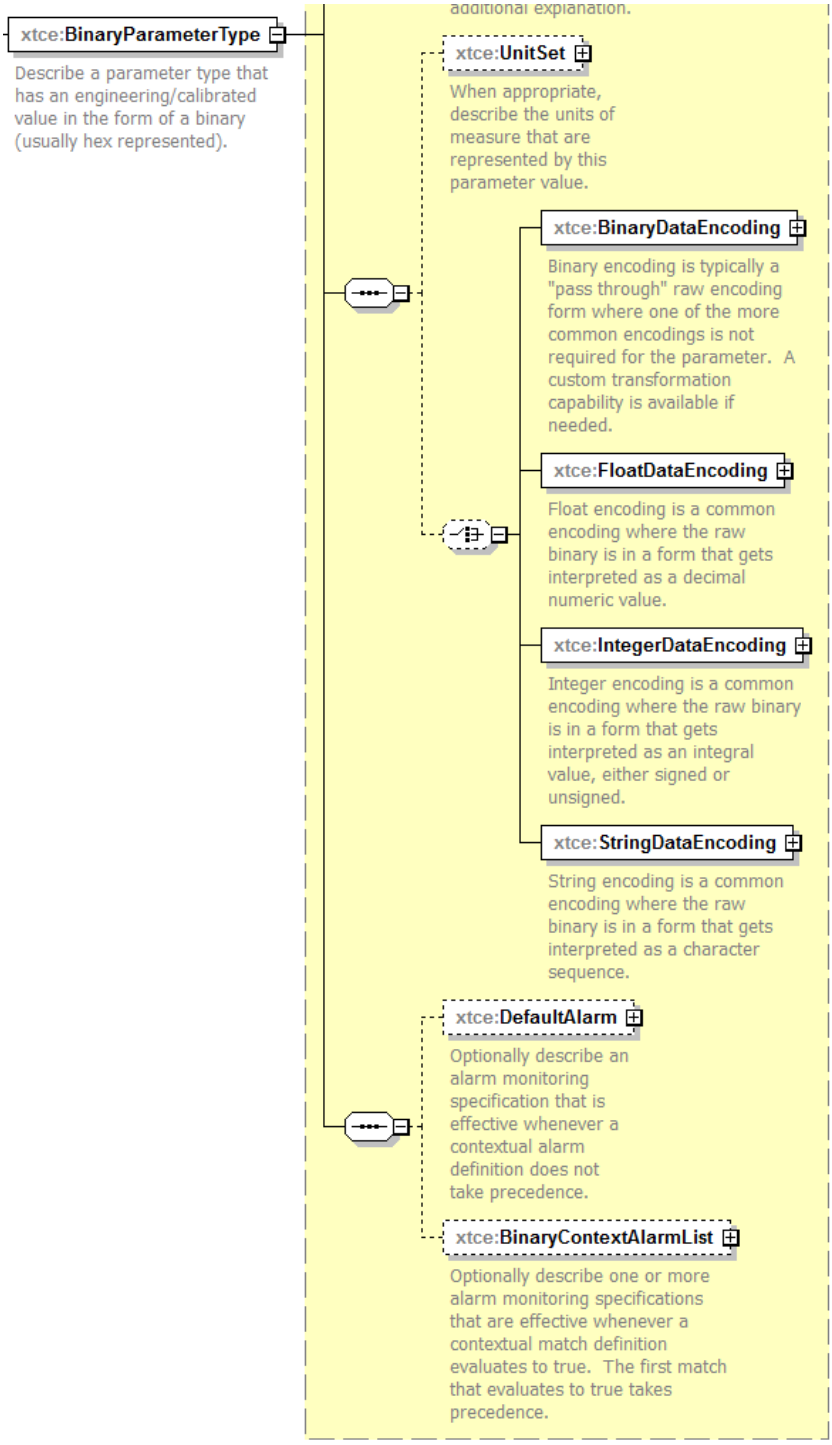



**Figure 4-49: ParameterType—Part 2**

These items are explored more fully below.

#### 4.3.2.4.5.2 initialValue Attribute

The initialValue attribute is specified in hex binary. The bits are truncated from the leftmost bit, MSB.

#### 4.3.2.4.5.3 BinaryDataEncoding Element

##### 4.3.2.4.5.3.1 General

Typically BinaryDataEncoding is used with BinaryParameterType.

##### 4.3.2.4.5.3.2 BinaryDataEncoding—SizeInBits Element

SizeInBits is used to specify the length using FixedValue, DynamicValue, or DiscreteLookup.

##### 4.3.2.4.5.3.3 BinaryDataEncoding—FromBinaryTransformAlgorithm or ToBinaryTransformAlgorithm Elements

FromBinaryTransformAlgorithm and ToBinaryTransformAlgorithm Elements are optional elements to capture a custom algorithm for transforming encoded binary data to the destination's data type.

#### 4.3.2.4.6 FloatParameterType

##### 4.3.2.4.6.1 General

FloatParameterType is used to describe either a float data encoding (source data type) or a calibrated integer data encoding (source data type) to a destination's float data type.

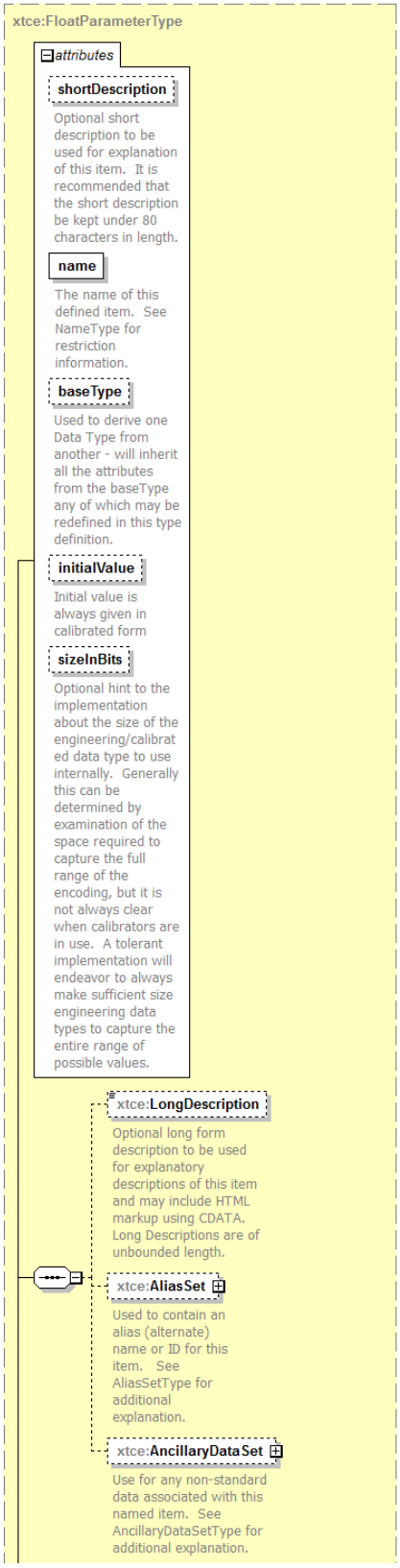



**Figure 4-50: FloatParameterType—Part 1**

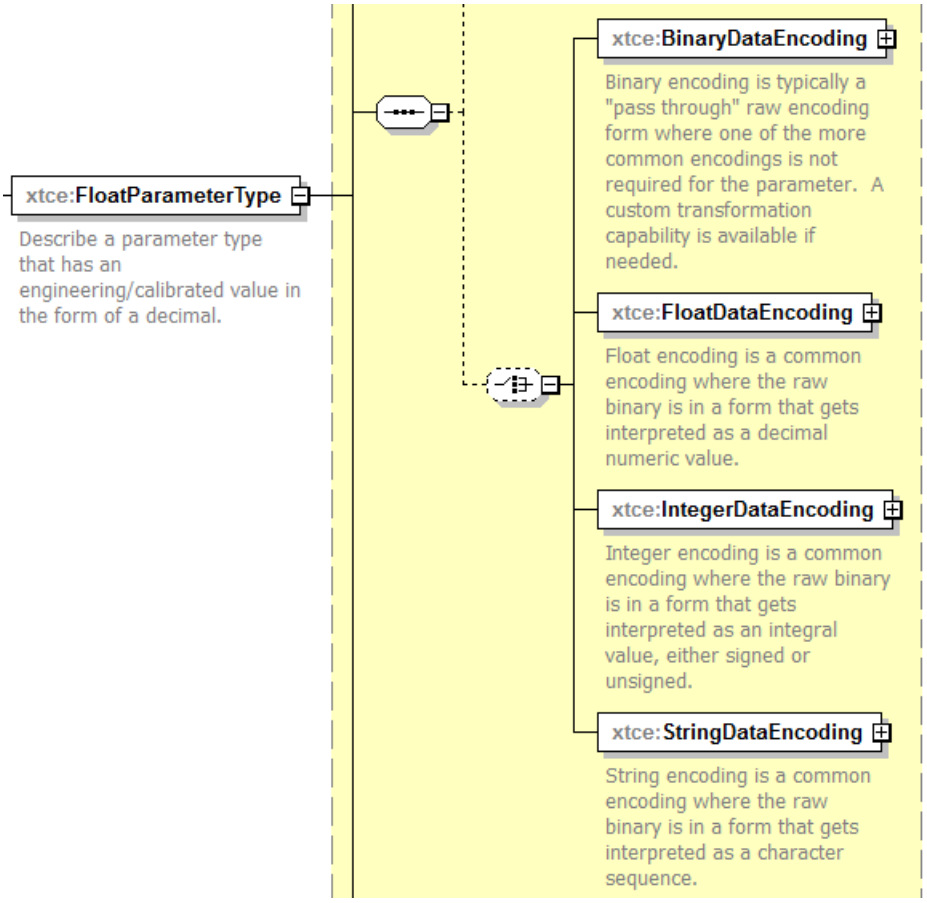



**Figure 4-51: FloatParameterType—Part 2**

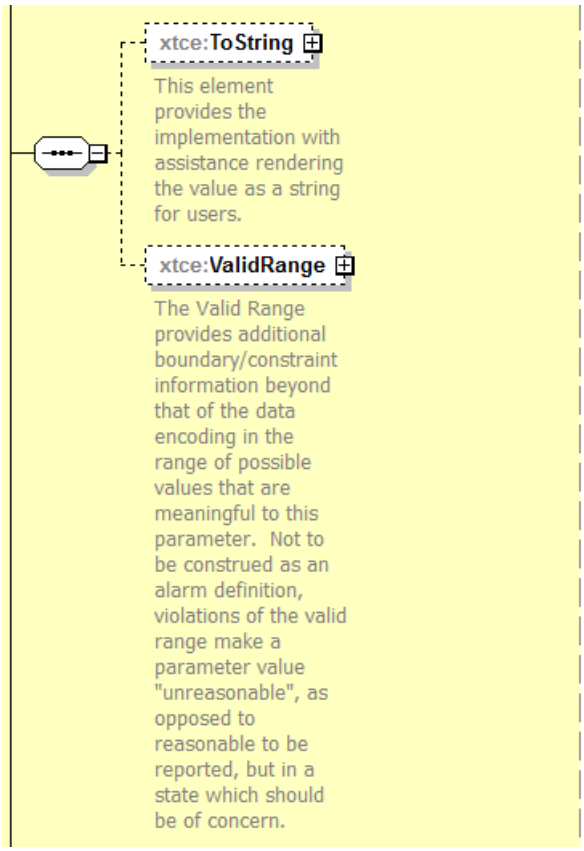



**Figure 4-52: FloatParameterType—Part 3**

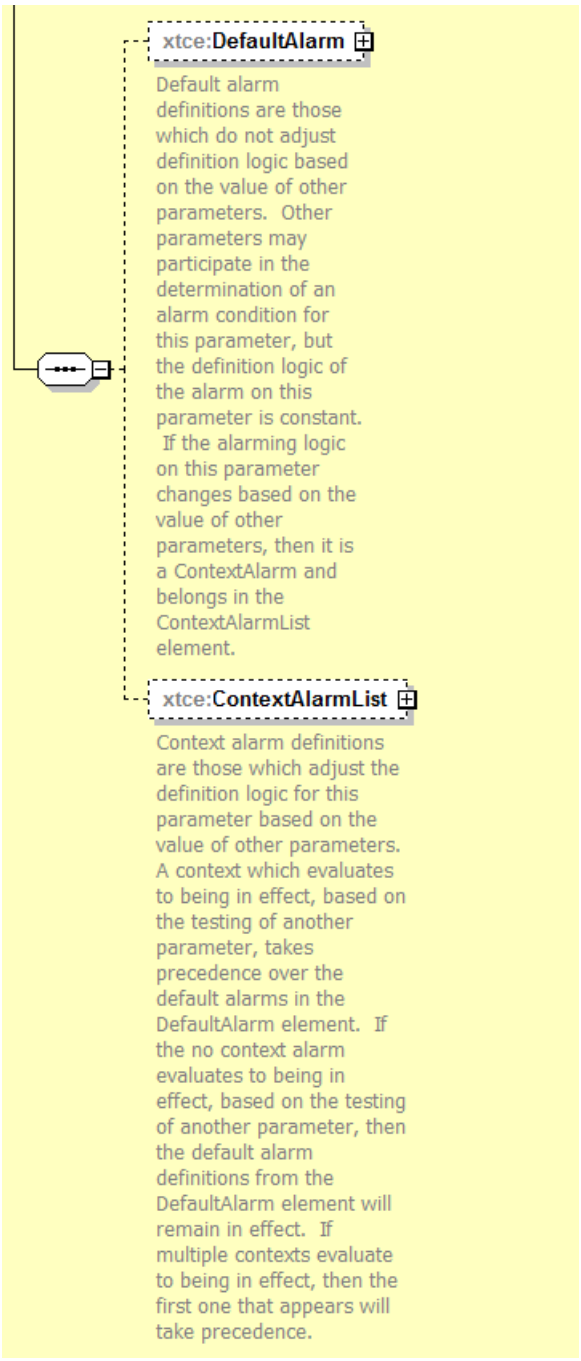



**Figure 4-53: FloatParameterType—Part 4**

#### 4.3.2.4.6.2 validRangeAppliesToCalibrated Attribute

The validRangeAppliesToCalibrated attribute is ignored since the ValidRange check is always applied before calibration (see 4.3.2.2).

#### 4.3.2.4.6.3 sizeInBits Attribute

In FloatParameterType, the sizeInBits attribute is used to describe the precision of the output of a calibrator (typically an integer count). For single precision, 32 is used; for double, 64; for extended, 80; and for quad precision, 128. The destination should match or exceed the precision needed by the encoding. For example, if the FloatDataEncoding is 32 bits, the FloatParameterType sizeInBits should be at least 32 bits.

#### 4.3.2.4.6.4 BinaryDataEncoding and FloatDataEncoding Elements

FloatDataEncoding supports IEEE or MIL-1750 formats. If other formats are needed, BinaryDataEncoding can be used to supply the basic information. The AncillaryData is used to supply other aspects of the format.

#### 4.3.2.4.6.5 IntegerDataEncoding Element

If IntegerDataEncoding is defined in FloatParameterType, it is used to designate a raw count to engineering unit conversion. The calibrator is defined in the IntegerDataEncoding element.

#### 4.3.2.4.6.6 ValidRange Element

The ValidRange element is similar to the element of the same name in IntegerParameterType (see 4.3.2.4.4.7) but for floating point.

### 4.3.2.4.7 BooleanParameterType

#### 4.3.2.4.7.1 General

BooleanParameterType is used to describe true/false items. This is a restricted form of enumeration.

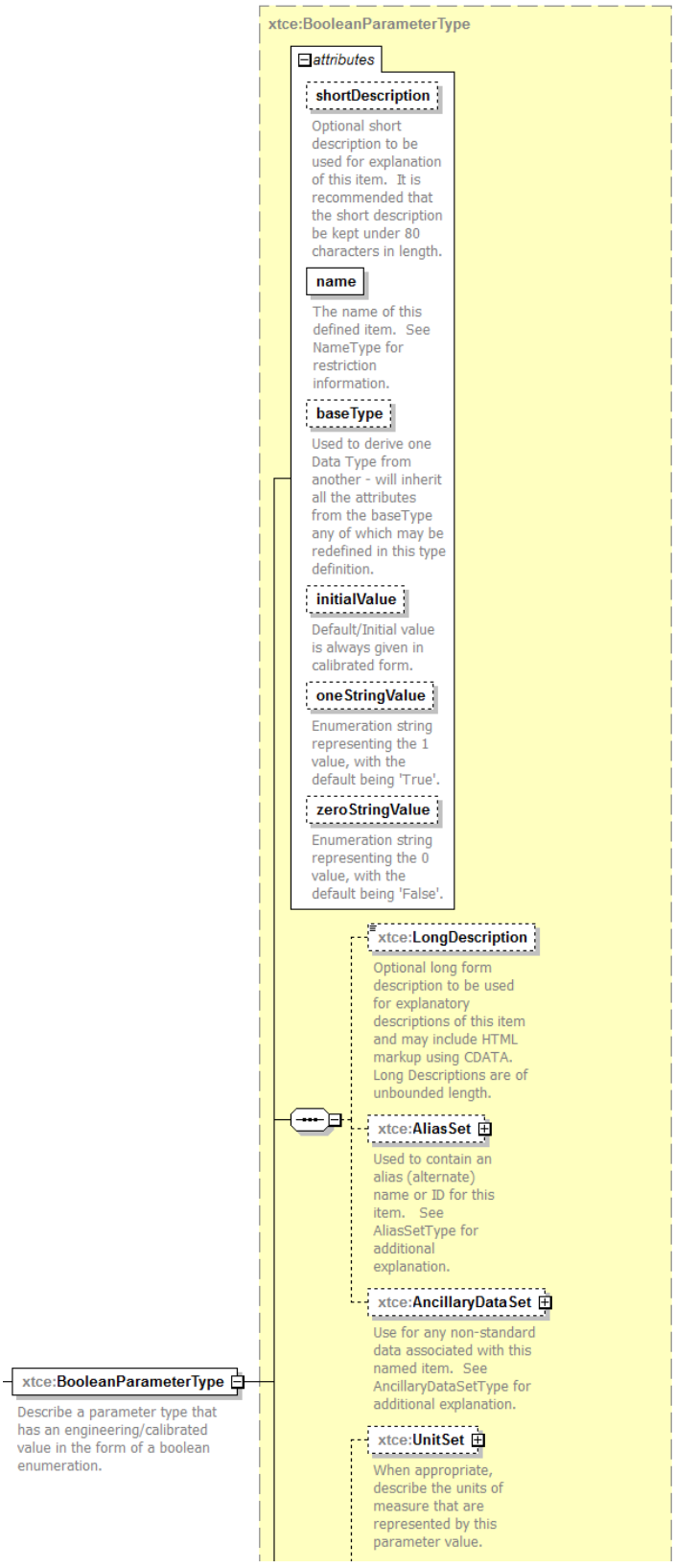



**Figure 4-54: BooleanParameterType—Part 1**

Second Part.

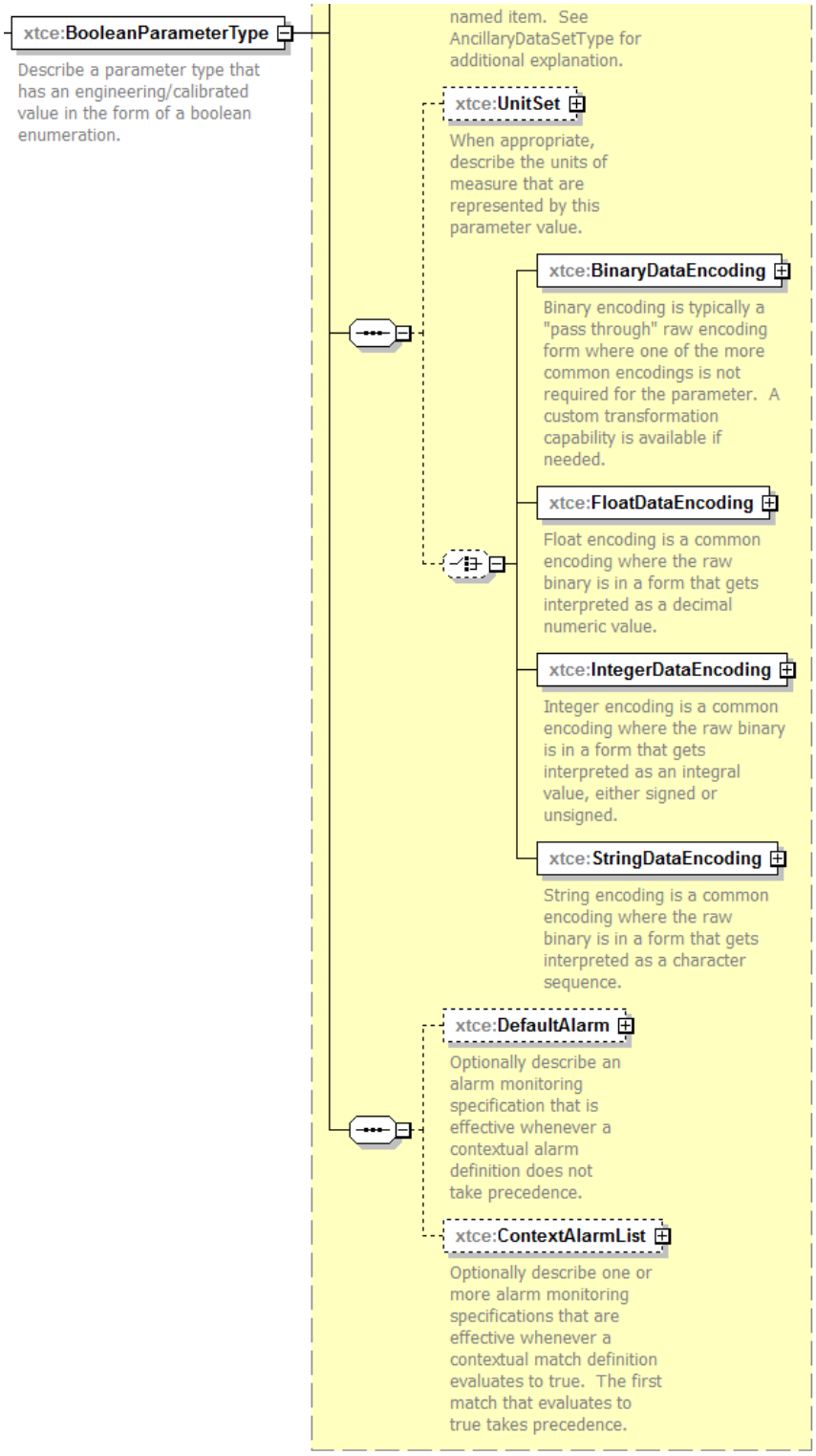



**Figure 4-55: BooleanParameterType—Part 2**

#### 4.3.2.4.7.2 oneStringValue Attribute

The oneStringValue attribute is used to set the string associated with value ‘1’ (default is ‘true’).

#### 4.3.2.4.7.3 zeroStringValue Attribute

The zeroStringValue attribute is used to set the string associated with value ‘0’ (default is ‘false’).

#### 4.3.2.4.7.4 IntegerDataEncoding Element

IntegerDataEncoding is used for encoding.

### 4.3.2.4.8 RelativeTimeParameterType

#### 4.3.2.4.8.1 General

RelativeTimeParameterType is a relative time description offset from an absolute time.

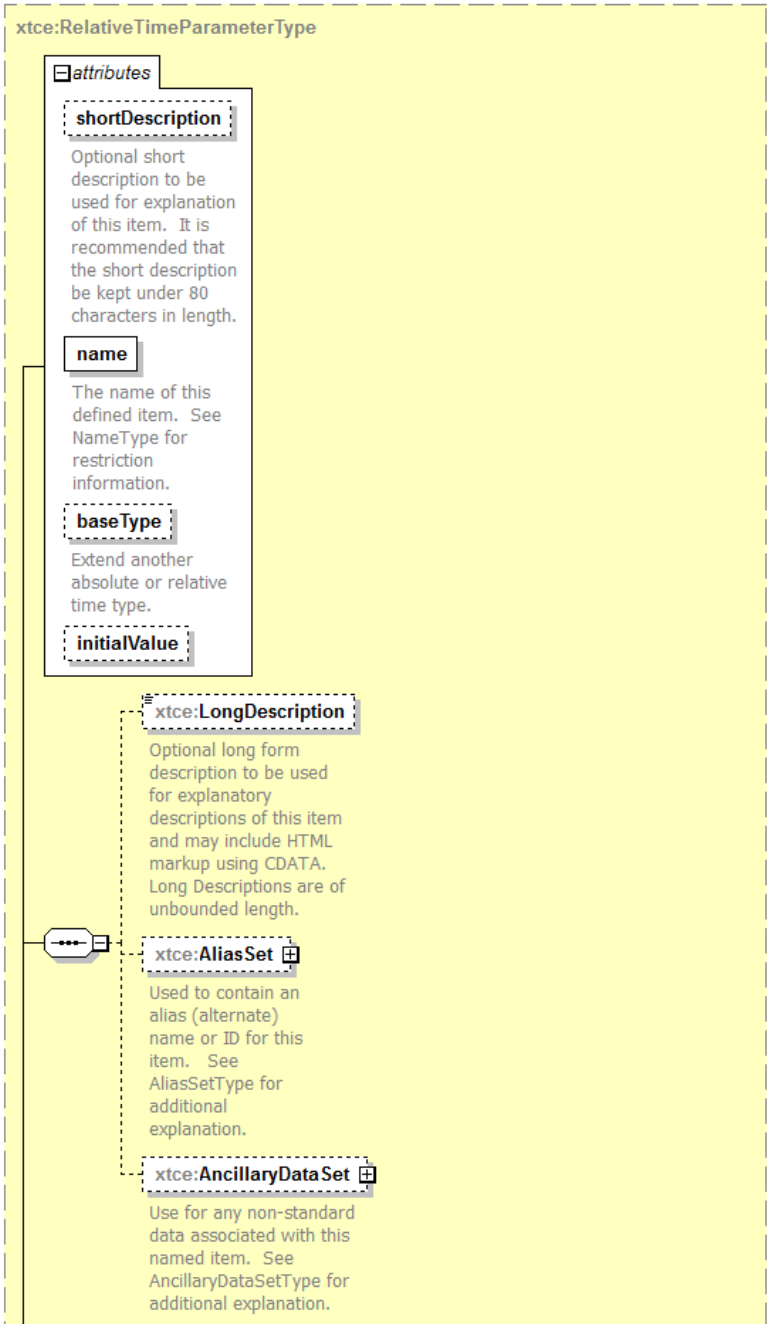



**Figure 4-56: RelativeTimeParameterType—Part 1**

Second Part.

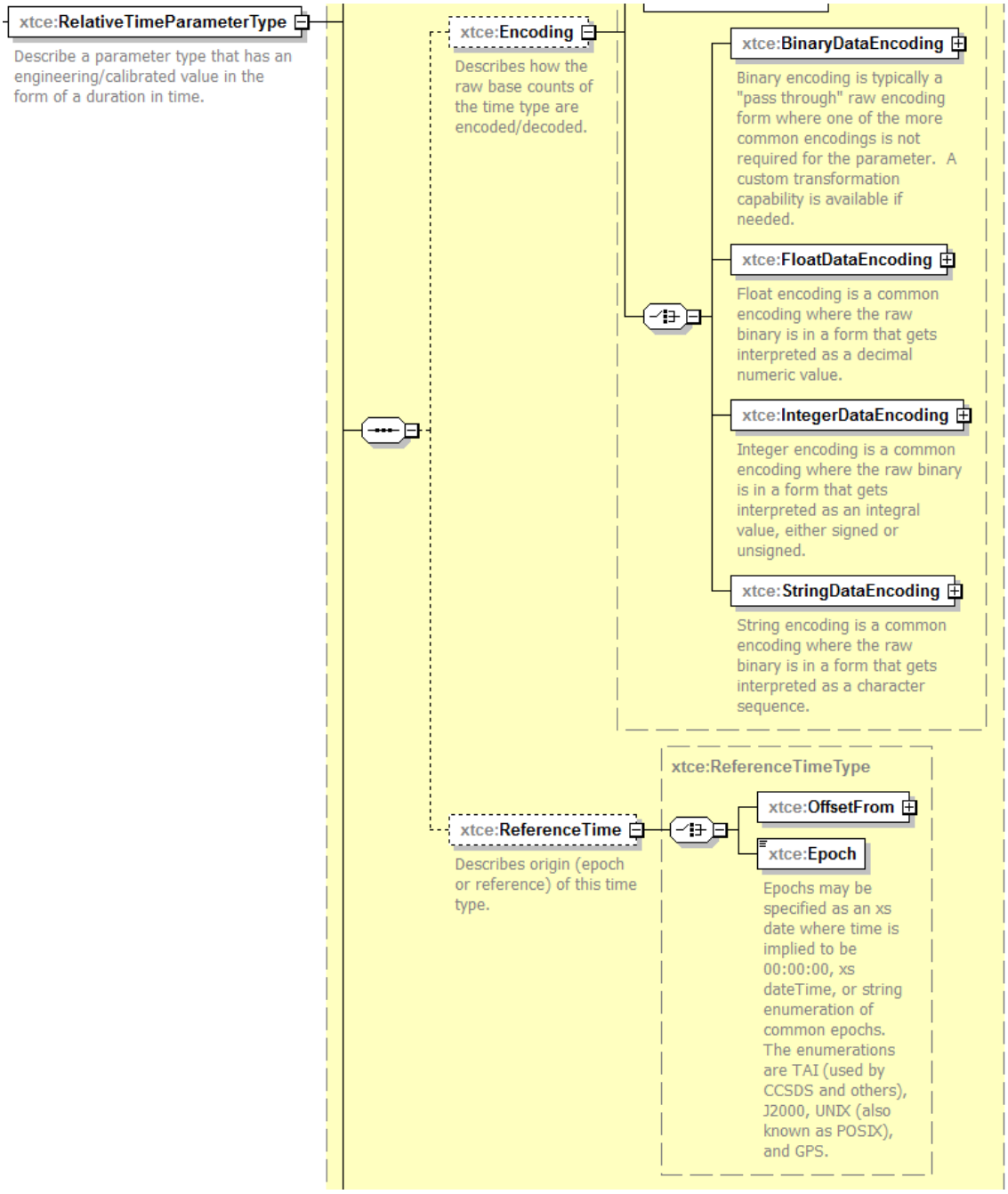



**Figure 4-57: RelativeTimeParameterType—Part 2**

Third part.

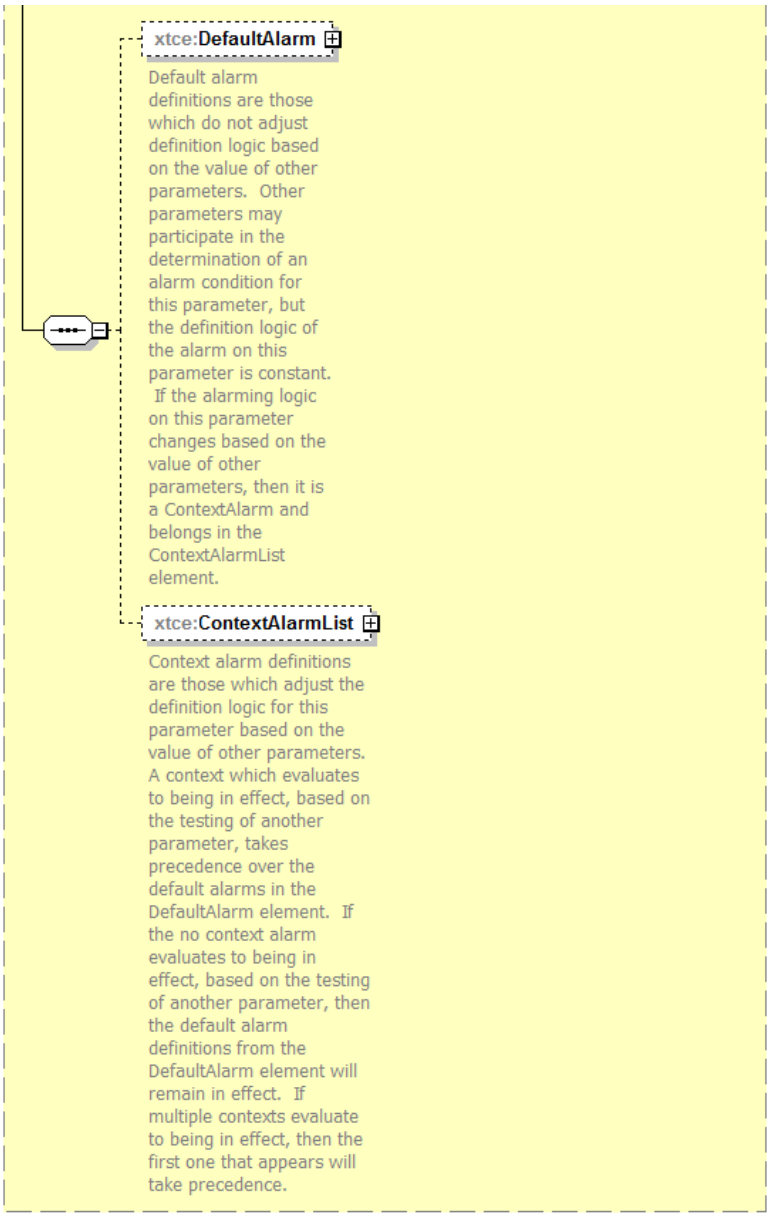



**Figure 4-58: RelativeTimeParameterType—Part 3**

#### 4.3.2.4.8.2 initialValue Attribute

- The initialValue attribute is a string formatted to xsd:duration. (See javax.xml.datatype for an implementation of this type.)
- An example of time specified for less than one second is PT0.5S for half a second.
- Longer time spans can be specified, such as: -P120D (-120 days) and P1Y2M3DT10H30M (1 year, 2 months, 3 days, 10 hours, and 30 minutes).

#### 4.3.2.4.8.3 units, scale, and offset Attributes

a) The units attribute is used to supply units related specifically to time.

b) The scale attribute supplies a scaling factor for the encoded value.

c) The offset attribute is used to set an offset.

#### 4.3.2.4.8.4 DataEncoding Elements

For DataEncoding Elements, IntegerDataEncoding is used. BinaryDataEncoding may be necessary for some cases. The other encodings are not defined.

#### 4.3.2.4.8.5 ReferenceTime—OffsetFrom Element

ReferenceTime—OffsetFrom Element is used to reference another time parameter. It allows for the stringing together of several dissimilar but related time parameters.

#### 4.3.2.4.8.6 ReferenceTime—Epoch Element

ReferenceTime—Epoch Element is used to specify a starting epoch for the date. This schema type is a union between xsd:date and the string International Atomic Time (TAI, January 1, 1958).

#### 4.3.2.4.8.7 Default and ContextAlarms

Alarms may be specified for RelativeTimeParameterType. The alarm forms include AlarmConditions, CustomAlarm, and StaticAlarmRanges.

The ChangePerSecondAlarmRanges is used to describe rates of changes that are either too fast or too slow. Its attribute '@timeUnits' specifies the unit of time.

### 4.3.2.4.9 AbsoluteTimeParameterType

#### 4.3.2.4.9.1 General

AbsoluteTimeParameterType describes absolute time values. This is similar to RelativeTimeParameterType in construction, except there are no alarms.

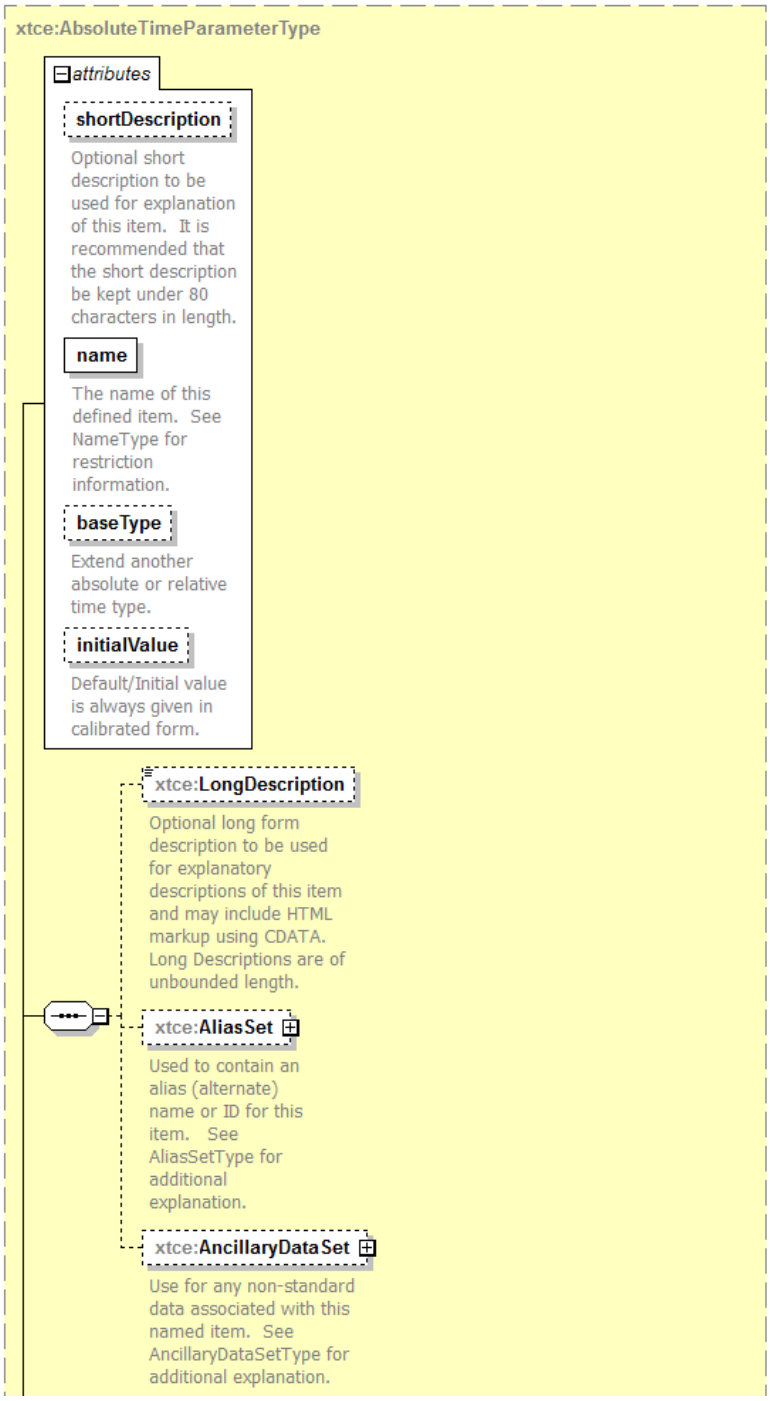



**Figure 4-59: AbsoluteTimeParameterType—Part 1**

Second part.

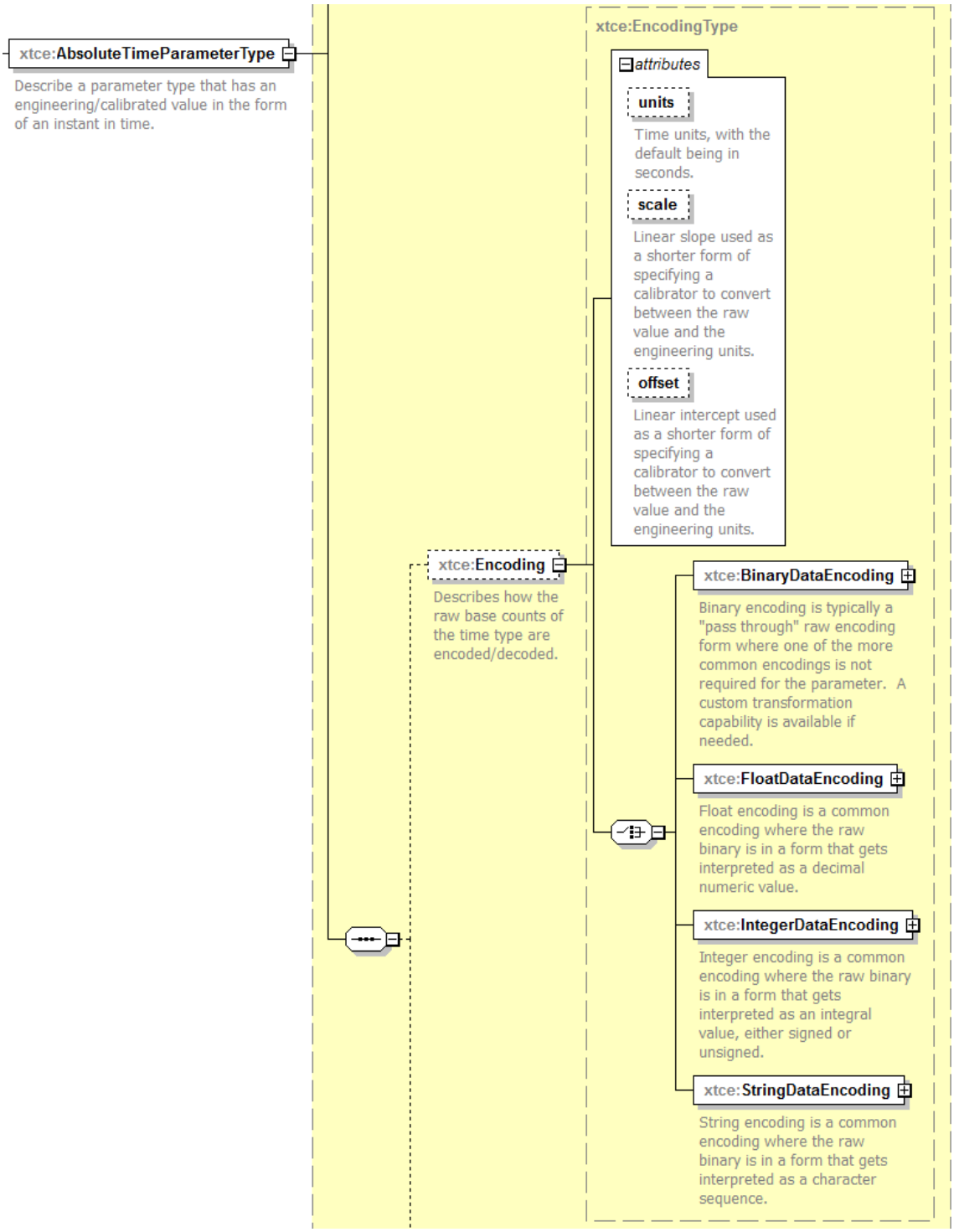



**Figure 4-60: AbsoluteTimeParameterType—Part 2**

Third part.

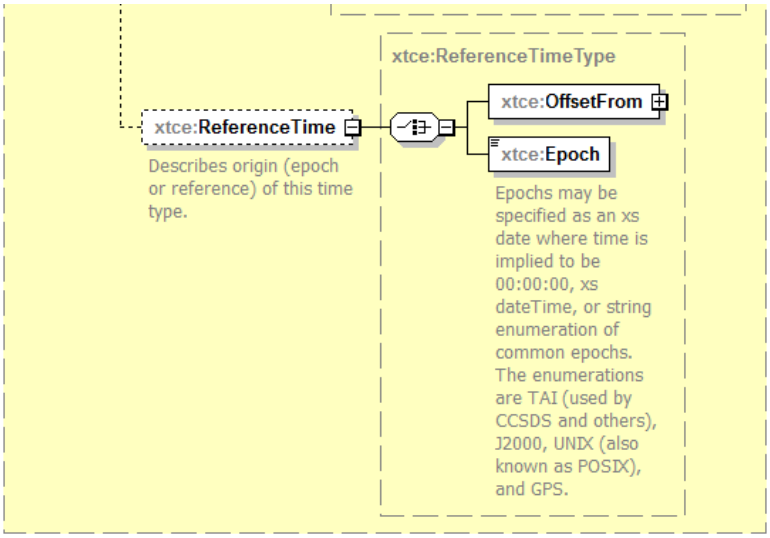



**Figure 4-61: AbsoluteTimeParameterType—Part 3**

#### 4.3.2.4.9.2 initialValue Attribute

The XML Schema dateTime type is used to describe absolute time initialValues. (See javax.xml.datatype.XMLGregorianCalendar for an implementation of dateTime.) The W3C Schema data type specifies the detailed explanation of accepted formats.

Examples include: 2002-10-10T12:00:00-05:00 (noon on 10 October 2002, Central Daylight Savings Time as well as Eastern Standard Time in the United States).

#### 4.3.2.4.9.3 ReferenceTime—Epoch Element

The ReferenceTime—Epoch element is used to hold the epoch.

### 4.3.2.4.10 ArrayParameterType

#### 4.3.2.4.10.1 General

ArrayParameterType is used to describe arrays of other ParameterTypes. The way arrays are defined has been revised in XTCE1.2. In XTCE 1.1 the number of dimensions was defined at the ArrayParameterType, but the size of each dimension was specified at the point of use in the container's EntryList.

This proved to be problematic. For many, it was cumbersome, but for some, it made defining Aggregates with members that were arrays impossible with creative syntax use.

So in XTCE 1.2 this has been revised so that the Dimensions are defined in each ArrayParameterType. This is explored more below.

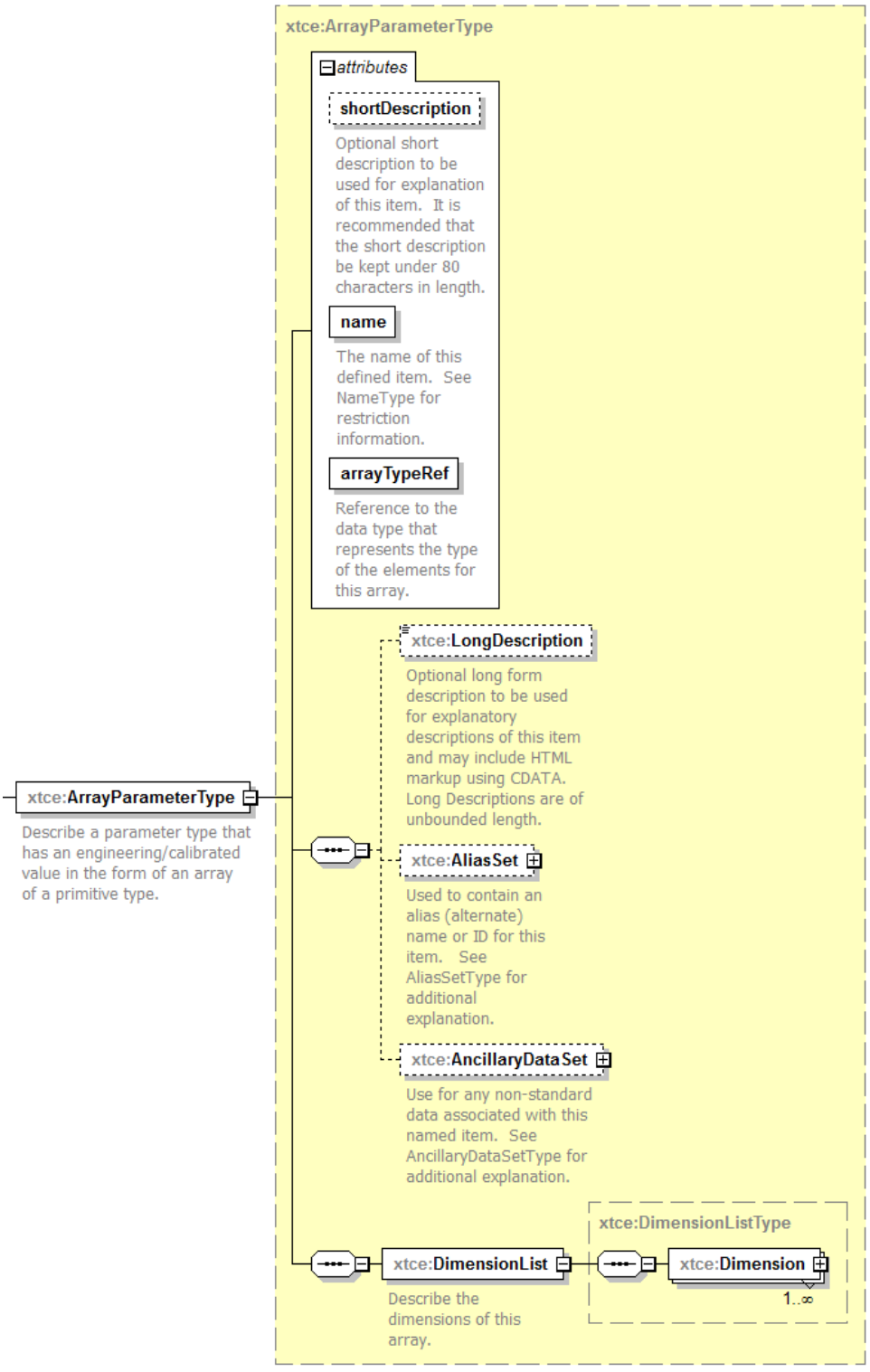



**Figure 4-62: ArrayParameterType**

#### 4.3.2.4.10.2 arrayTypeRef Attribute

ArrayTypeRef attribute is a NameReference to another ParameterType from which the array cells are formed (i.e., its data type).

#### 4.3.2.4.10.3 DimensionList Element

This is a change from XTCE 1.1, and it replaces the numberOfDimensions attribute and EntryList DimensionList combination in that version.

Instead, the number of dimensions and their sizes are set here. The EntryList still retains the ArrayParameterRefEntry element, and it has a DimensionList there, which is optional, but it may only be used to subset the DimensionList specified here in the ParameterType.

### 4.3.2.4.11 AggregateParameterType

#### 4.3.2.4.11.1 General

AggregateParameterType is used to describe aggregates. It is similar to C-structs or records in other languages.

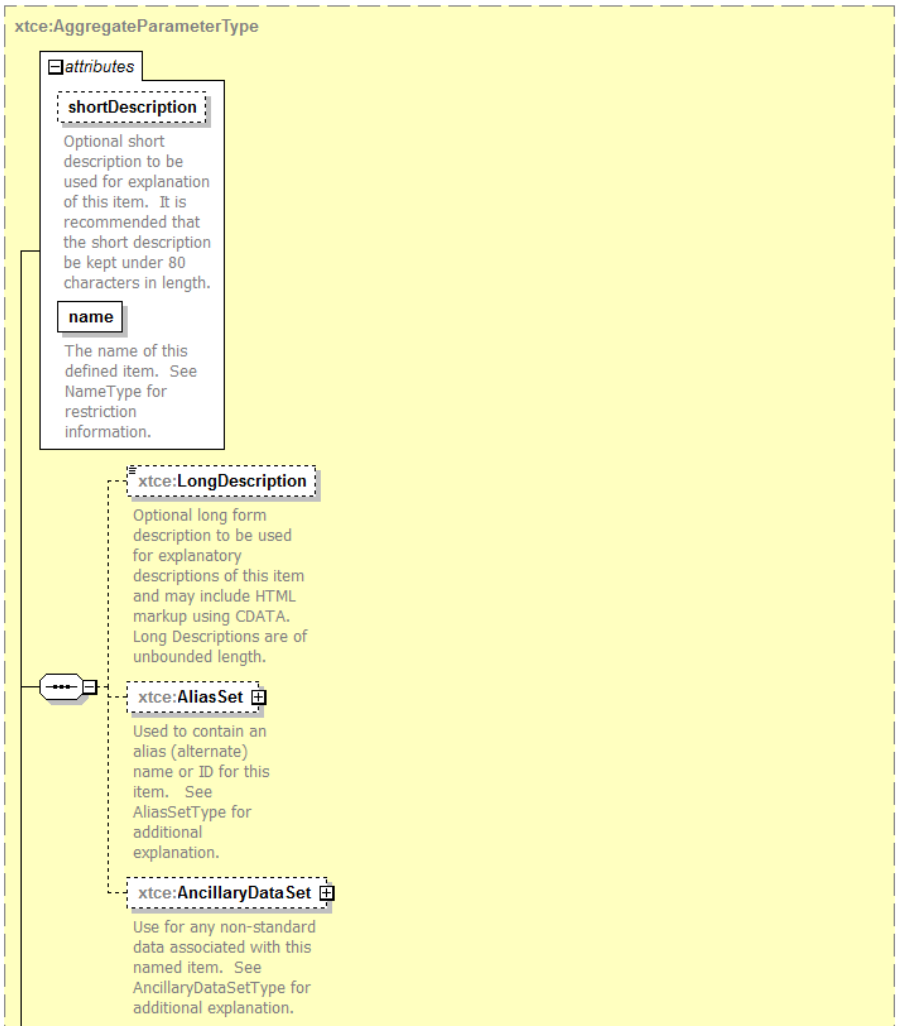



**Figure 4-63: AggregateParameterType—Part 1**

Second Part.

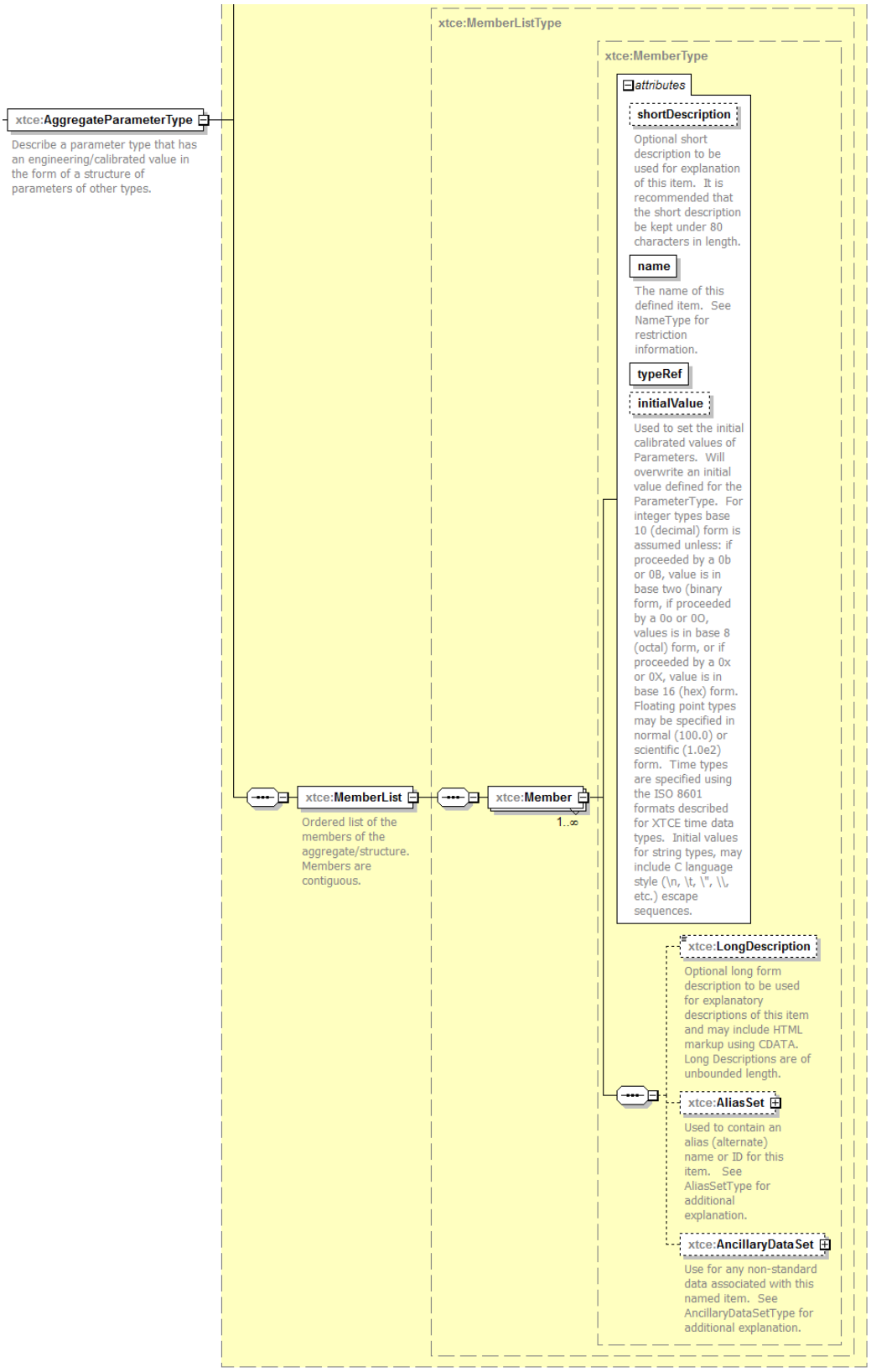



**Figure 4-64: AggregateParameterType—Part 2**

As can be seen, the Aggregate has a number of new elements related to documentation and initial value.

#### 4.3.2.4.11.2 MemberList/Member Elements

MemberList/Member Elements are lists of fields of the aggregate; typeRef is a NameReference to another ParameterType.

NOTE – ParameterInstanceRefs to an aggregate use the syntax 'AggregateName.MemberName'. Further members can now point to ArrayParamaterTypes.

### 4.3.2.5 ParameterType and Encoding Tables—Recommendations

#### 4.3.2.5.1 General

Tables 4-4–4-18 provide examples of ParameterTypes and encodings. Examples of each are given after the tables. Defining other ParameterType encodings is the responsibility of the end user. Default attribute values are used when possible to decrease the size of the description.

#### 4.3.2.5.2 Uncalibrated ParameterTypes

**Table 4-4: Settings for Uncalibrated StringParameterType**

| **Destination Side** | | **Encoding (Source) Side** | | |
|---|---|---|---|---|
| **ParameterType** | **@characterWidth** | **DataEncoding** | **@encoding** | **SizeInBits** |
| String | ignore (use default); interpret as Unicode | String | UTF-8 (use default) | Set to size taking into account codes per character |
| String | ignore (use default); interpret as Unicode | String | UTF-16 | Set to size taking into account codes per character |

**UTF-8**

```
<xtce:StringParameterType name="UTF8Type">
 <xtce:UnitSet/>
 <xtce:StringDataEncoding>
 <!—Ex. Length in bits is 11 characters,
 8-bits per ASCII-type character -->
      <xtce:SizeInBits>
 <xtce:Fixed>
 <xtce:FixedValue>88</xtce:FixedValue>
 </xtce:Fixed>
 </xtce:SizeInBits>
 </xtce:StringDataEncoding>
</xtce:StringParameterType>
```

**UTF-16**

```
<xtce:StringParameterType name="UTF16Type">
 <xtce:UnitSet/>
 <xtce:StringDataEncoding encoding="UTF-16">
 <!-- Length in bits is 11 characters,
 16-bits per character -->
 <xtce:SizeInBits>
 <xtce:Fixed>
 <xtce:FixedValue>176</xtce:FixedValue>
 </xtce:Fixed>
 </xtce:SizeInBits>
 </xtce:StringDataEncoding>
</xtce:StringParameterType>
```

**Table 4-5: Settings for Uncalibrated Unsigned IntegerParameterType**

| Destination Side | | | Encoding (Source) Side | | |
|---|---|---|---|---|---|
| **ParameterType** | **@signed** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Integer | unsigned | 32 (use default) | Integer | unsigned (use default) | 1–7 |
| Integer | unsigned | 32 (use default) | Integer | unsigned (use default) | 8 (use default) |
| Integer | unsigned | 32 (use default) | Integer | unsigned (use default) | 9–16 |
| Integer | unsigned | 32 (use default) | Integer | unsigned (use default) | 17–32 |
| Integer | unsigned | 64 | Integer | unsigned (use default) | 33–64 |
| Integer | unsigned | Set to next multiple | Integer | unsigned (use default) | 64+ |

```
<xtce:IntegerParameterType name="UnsignedLongType" signed="false">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding encoding="unsigned" sizeInBits="32"/>
</xtce:IntegerParameterType>
```

**Table 4-6: Settings for Uncalibrated Unsigned IntegerParameterType**

| Destination Side | | | Encoding (Source) Side | | |
|---|---|---|---|---|---|
| **ParameterType** | **@signed** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Integer | signed<br>(use default) | 32<br>(use default) | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 2–7 |
| Integer | signed<br>(use default) | 32<br>(use default) | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 8<br>(use default) |
| Integer | signed<br>(use default) | 32<br>(use default) | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 9–16 |
| Integer | signed<br>(use default) | 32<br>(use default) | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 17–32 |
| Integer | signed<br>(use default) | 64 | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 33–64 |
| Integer | signed<br>(use default) | Set to next multiple | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 64+ |

```
<xtce:IntegerParameterType name="SignedLongType">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding sizeInBits="32"/>
</xtce:IntegerParameterType>
```

**Table 4-7: Settings for Uncalibrated BCD IntegerParameterType**

| Destination Side | | | Encoding (Source) Side | | |
|---|---|---|---|---|---|
| **ParameterType** | **@signed** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Integer | unsigned | 32<br>(use default) | Integer | BCD<br>packedBCD | 4+ in 4 or 8 bit increments up to values 2^32 |
| Integer | unsigned | 32<br>(use default) | Integer | BCD<br>packedBCD | 8<br>(use default) |
| Integer | signed<br>(use default) | 32<br>(use default) | Integer | BCD<br>packedBCD | 4+ in 4 or 8 bit increments up to values of +/–2^32–1 |
| Integer | signed<br>(use default) | 32<br>(use default) | Integer | BCD<br>packedBCD | 8 (use default) |
| Integer | unsigned | Set to bit precision to hold largest value | Integer | BCD<br>packedBCD | Size needed to hold value |
| Integer | signed<br>(use default) | Set to bit precision to hold max/min values | Integer | BCD<br>packedBCD | Size needed to hold value |

```
<xtce:IntegerParameterType name="BCD0Thru9">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding sizeInBits="40" encoding="BCD"/>
</xtce:IntegerParameterType>
```

**Table 4-8: Settings for Uncalibrated FloatParameterType**

| Destination Side | | | Encoding (Source) Side | | |
|---|---|---|---|---|---|
| **ParameterType** | **@sizeInBits** | **AncillaryData** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Float | 32 (use default) | | Float | IEEE754_1985 (use default) | 32 (use default) |
| Float | 64 | | Float | IEEE754_1985 (use default) | 64 |
| Float | 128 | @name= "IEEE754_1985_ EXTENDED" Element holds 80 | Float | IEEE754_1985 (use default) | 32 (use default); interpret as 80 |
| Float | 128 | | Float | IEEE754_1985 (use default) | 128 |
| Float | 32 (use default) | | Float | MILSTD_1750A | 32 (use default) |
| Float | 64 | @name= "MILSTD_1750A_ EXTENDED" Element holds 48 | Float | MILSTD_1750A | 32 (use default); interpret as 48 |

```
<xtce:FloatParameterType name="SinglePrecisionFloatType">
    <xtce:UnitSet/>
    <xtce:FloatDataEncoding/>
</xtce:FloatParameterType>
```

**Table 4-9: Settings for Uncalibrated EnumeratedParameterType**

| Destination Side | Encoding (Source) Side | | |
|---|---|---|---|
| **ParameterType** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Enumerated | Integer | unsigned (use default) | 1–7 |
| Enumerated | Integer | unsigned (use default) | 8 (use default) |
| Enumerated | Integer | unsigned (use default) | 9+ |

```
<xtce:EnumeratedParameterType name="PowerStateType">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding />
    <xtce:EnumerationList>
        <xtce:Enumeration label="ON" value="1"/>
        <xtce:Enumeration label="OFF" value="0"/>
    </xtce:EnumerationList>
</xtce:EnumeratedParameterType>
```

**Table 4-10: Settings for Uncalibrated BinaryParameterType**

| Destination Side | Encoding (Source) Side | |
|---|---|---|
| **ParameterType** | **DataEncoding** | **SizeInBits** |
| Binary | Binary | 1+ |

```
<xtce:BinaryParameterType name="Blob">
    <xtce:UnitSet/>
    <xtce:BinaryDataEncoding>
        <xtce:SizeInBits>
            <xtce:FixedValue>255</xtce:FixedValue>
        </xtce:SizeInBits>
</xtce:BinaryDataEncoding>
</xtce:BinaryParameterType>
```

**Table 4-11: Settings for Uncalibrated BooleanParameterType**

| Destination Side | Encoding (Source) Side | | |
|---|---|---|---|
| **ParameterType** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Boolean | Integer | unsigned (use default) | 1 |

```
<xtce:BooleanParameterType name="Boolean">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding sizeInBits="1"/>
</xtce:BooleanParameterType>
```

**Table 4-12: Settings for Uncalibrated Relative and AbsoluteParameterTypes**

| Destination Side | Encoding (Source) Side | |
|---|---|---|
| **ParameterType** | **DataEncoding** | **Attributes and Sizes** |
| Relative and Absolute | Integer and String likely | Follow suggestions above |

```
<xtce: RelativeTimeParameterType name="SecondsTimeType">
    <xtce:Encoding units="seconds">
        <xtce:IntegerDataEncoding sizeInBits="32"/>
    </xtce:Encoding>
    <xtce:ReferenceTime>
        <xtce:OffsetFrom parameterRef="SubSeconds"/>
    </xtce:ReferenceTime>
</xtce:RelativeTimeParameterType>
<xtce:RelativeTimeParameterType name="SubSecondsTimeType">
    <xtce:Encoding scale="0.00001" units="seconds">
        <xtce:IntegerDataEncoding sizeInBits="32"/>
    </xtce:Encoding>
```

```
        <xtce:ReferenceTime>
            <xtce:Epoch>TAI</xtce:Epoch>
        </xtce:ReferenceTime>
    </xtce:AbsoluteTimeParameterType>
    <xtce:ParameterSet>
        <xtce:Parameter name="Time" parameterTypeRef="SecondsTimeType"/>
        <xtce:Parameter name="SubSeconds" parameterTypeRef="SubSecondsTimeType"/>
    </xtce:ParameterSet>
    <xtce:RelativeTimeParameterType name="SecondsTimeType">
        <xtce:Encoding units="seconds">
            <xtce:IntegerDataEncoding sizeInBits="32"/>
        </xtce:Encoding
        <xtce:ReferenceTime>
        <xtce:OffsetFrom parameterRef="SubSeconds"/>
        </xtce:ReferenceTime>
    </xtce:RelativeTimeParameterType>
    <xtce:RelativeTimeParameterType name="SubSecondsTimeType">
        <xtce:Encoding scale="0.00001" units="seconds">
            <xtce:IntegerDataEncoding sizeInBits="32"/>
        </xtce:Encoding>
        <xtce:ReferenceTime>
            <xtce:Epoch>TAI</xtce:Epoch>
        </xtce:ReferenceTime>
    </xtce:RelativeTimeParameterType>
    <xtce:ParameterSet>
        <xtce:Parameter name="Time" parameterTypeRef="SecondsTimeType"/>
        <xtce:Parameter name="SubSeconds" parameterTypeRef="SubSecondsTimeType"/>
    </xtce:ParameterSet>
```

**Table 4-13: Settings for Alternative Uncalibrated ParameterTypes**

| **Destination Side** | | | **Encoding (Source) Side** | |
|---|---|---|---|---|
| **ParameterType** | **@sizeInBits** | **AncillaryData** | **DataEncoding** | **SizeInBits** |
| Integer | Set to size large enough for precision; use default when possible | @name="INTEGER_FORMAT" Element holds format name convention | Binary | 1+ |
| Float | Set to size large enough for precision; use default when possible | @name="FLOAT_FORMAT" Element holds format name convention | Binary | 1+ |
| String | Set to size large enough for precision; use default when possible | @name="STRING_FORMAT" Element holds format name convention | Binary | 1+ |

```
<xtce:FloatParameterType name="NotVeryCommon">
    <xtce:AncillaryDataSet>
        <xtce:AncillaryData name="FLOAT_FORMAT">
                                Motorola68881PackedDecimalReal
        </xtce:AncillaryData>
    </xtce:AncillaryDataSet>
    <xtce:UnitSet/>
    <xtce:BinaryDataEncoding>
        <xtce:SizeInBits>
            <xtce:FixedValue>82</xtce:FixedValue>
        </xtce:SizeInBits>
    </xtce:BinaryDataEncoding>
</xtce:FloatParameterType>
```

#### 4.3.2.5.3 Calibrated ParameterTypes

Only certain integer telemetry parameters and float parameters will use calibrators (see table 4-14).

**Table 4-14: Settings for Calibrated Unsigned IntegerParameterType**

| **Destination Side** | | | **Encoding (Source) Side** | | |
|---|---|---|---|---|---|
| **ParameterType** | **@signed** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Integer | unsigned | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | unsigned (use default) | 1–7 |
| Integer | unsigned | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | unsigned (use default) | 8 (use default) |
| Integer | unsigned | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | unsigned (use default) | 9–16 |
| Integer | unsigned | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | unsigned (use default) | 17–32 |
| Integer | unsigned | Long Long Precision: 64 | Integer | unsigned (use default) | 33–64 |
| Integer | unsigned | Set to next multiple | Integer | unsigned (use default) | 64+ |

**Table 4-15: Settings for Calibrated Signed IntegerParameterTypes**

| **Destination Side** | | | **Encoding (Source) Side** | | |
|---|---|---|---|---|---|
| **ParameterType** | **@signed** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Integer | signed<br>(use default) | Long Precision: 32<br>(use default)<br>Long Long Precision: 64 | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 2–7 |
| Integer | signed<br>(use default) | Long Precision: 32<br>(use default)<br>Long Long Precision: 64 | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 8<br>(use default) |
| Integer | signed<br>(use default) | Long Precision: 32<br>(use default)<br>Long Long Precision: 64 | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 9–16 |
| Integer | signed<br>(use default) | Long Precision: 32<br>(use default)<br>Long Long Precision: 64 | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 17–32 |
| Integer | signed<br>(use default) | Long Long Precision: 64 | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 33–64 |
| Integer | signed<br>(use default) | Set to next multiple | Integer | twosComplement<br>onesComplement<br>signedMagnitude | 64+ |

**Table 4-16: Settings for Calibrated BCD IntegerParameterType**

| **Destination Side** | | | **Encoding (Source) Side** | | |
|---|---|---|---|---|---|
| **ParameterType** | **@signed** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Integer | unsigned | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | BCD<br>packedBCD | 4+ in 4 or 8 bit increments up to values 2^32 |
| Integer | unsigned | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | BCD<br>packedBCD | 8<br>(use default) |
| Integer | signed (use default) | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | BCD<br>packedBCD | 4+ in 4 or 8 bit increments up to values of +/- 2^32-1 |
| Integer | signed (use default) | Long Precision: 32 (use default)<br>Long Long Precision: 64 | Integer | BCD<br>packedBCD | 8<br>(use default) |
| Integer | unsigned | Set to bit precision to hold largest value | Integer | BCD<br>packedBCD | Size needed to hold value |
| Integer | signed (use default) | Set to bit precision to hold max/min values | Integer | BCD<br>packedBCD | Size needed to hold value |

**Table 4-17: Settings for Calibrated FloatParameterType**

| Destination Side | | | Encoding (Source) Side | | |
|---|---|---|---|---|---|
| ParameterType | @sizeInBits | AncillaryData | DataEncoding | @encoding | @sizeInBits |
| Float | At least Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | | Float | IEEE754_1985 (use default) | 32 (use default) |
| Float | At least Double Precision: 64<br>Quad Precision: 128 | | Float | IEEE754_1985 (use default) | 64 |
| Float | Quad Precision: 128 | @name= "IEEE754_1985_EXTENDED" | Float | IEEE754_1985 (use default) | 32 (use default); interpret as 80 |
| Float | Quad Precision: 128 | | Float | IEEE754_1985 (use default) | 128 |
| Float | At least Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | | Float | MILSTD_1750A | 32 (use default) |
| Float | At least Double Precision: 64<br>Quad Precision: 128 | @name= "MILSTD_1750A_EXTENDED" | Float | MILSTD_1750A | 32 (use default); interpret as 48 |

```
<xtce:FloatParameterType name="YourType" sizeInBits="64">
    <xtce:UnitSet/>
    <xtce:FloatDataEncoding>
        <xtce:DefaultCalibrator>
            <xtce:PolynomialCalibrator>
                <xtce:Term exponent="0" coefficient="-48.446"/>
                <xtce:Term exponent="1" coefficient="1.4091"/>
                <xtce:Term exponent="2" coefficient="-0.0048"/>
                <xtce:Term exponent="3" coefficient="0"/>
                <xtce:Term exponent="4" coefficient="0"/>
                <xtce:Term exponent="5" coefficient="0"/>
                <xtce:Term exponent="6" coefficient="0"/>
                <xtce:Term exponent="7" coefficient="0"/>
            </xtce:PolynomialCalibrator>
        </xtce:DefaultCalibrator>
    </xtce:FloatDataEncoding>
</xtce:FloatParameterType>
```

**Table 4-18: Settings for Calibrated FloatParameterType from Integer**

| Destination Side | | Encoding (Source) Side | | |
|---|---|---|---|---|
| **ParameterType** | **@sizeInBits** | **DataEncoding** | **@encoding** | **@sizeInBits** |
| Float | Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | Integer | unsigned<br>(use default) | 1–7 |
| Float | Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | Integer | unsigned<br>(use default) | 8<br>(use default) |
| Float | Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | Integer | unsigned<br>(use default) | 9+ |
| Float | Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | Integer | signed | 2–7 |
| Float | Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | Integer | signed | 8<br>(use default) |
| Float | Single Precision: 32 (use default)<br>Double Precision: 64<br>Quad Precision: 128 | Integer | signed | 9+ |

```
<xtce:FloatParameterType name="YourType" sizeInBits="64">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding sizeInBits="16"/>
        <xtce:DefaultCalibrator>
            <xtce:SplineCalibrator>
                <xtce:SplinePoint raw="0" calibrated="1"/>
                <xtce:SplinePoint raw="65535" calibrated="100"/>
            </xtce:SplineCalibrator>
        </xtce:DefaultCalibrator>
    </xtce:IntegerDataEncoding/>
</xtce:FloatParameterType>
```

#### 4.3.2.5.4 Complex Type Examples

##### 4.3.2.5.4.1 ArrayParameterType

The sizes of the array dimensions can be subset in a container in ArrayParameterRefEntry, although they are now set in XTCE 1.2 in the ArrayParameterType itself.

```
<xtce:ArrayParameterType name="BVArray" arrayTypeRef="BatVolt">
    <xtce:DimensionList>
        <xtce:Dimension>
            <xtce:StartingIndex>
                <xtce:FixedValue>0</xtce:FixedValue>
            </xtce:StartingIndex>
```

```
                    <xtce:EndingIndex>
                        <xtce:FixedValue>4</xtce:FixedValue>
                    </xtce:EndingIndex>
                </xtce:Dimension>
            </xtce:DimensionList>
<xtce:ArrayParameterType>
```

#### 4.3.2.5.4.2 AggregateParameterType

The following is an example of AggregateParameterType:

```
<xtce:MemberList>
    <xtce:Member typeRef="TimeType" name="TimeStart"/>
    <xtce:Member typeRef="TimeType" name="TimeEnd"/>
    <xtce:Member typeRef="IntType" name="Count"/>
    <xtce:Member typeRef="IntType" name="ErrorCnt"/>
</xtce:MemberList>
```

Aggregate are 'structs' or records. Aggregate members may refer to another aggregate. It should be noted that in XTCE 1.2, member may refer to an ArrayParameterType.

### 4.3.3 PARAMETERSET—PARAMETERS

#### 4.3.3.1 General

A ParameterSet holds Parameters, which refer to ParameterTypes using a NameReference to form complete mnemonic definitions. A single Parameter and ParameterType fully describes a single telemetry mnemonic (sample) within XTCE except for its bit location within Container, so that it can be decommutated into a ground side data type variable.

A ParameterSet may also define a ParameterRef, which is used to import a Parameter defined in another SpaceSystem into this SpaceSystem ParameterSet.

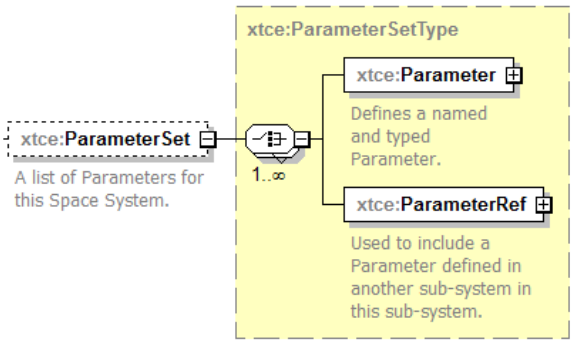



**Figure 4-65: ParmeterSet**

Unlike ParameterType, Parameter has few descriptive elements; often the only attributes set are the name and the parameterTypeRef attributes. However, one should look more deeply into ParameterProperties and set the dataSource attribute. (ParameterProperties is optional, and so the default interpretation without this attribute set is that the Parameter is telemetered.)

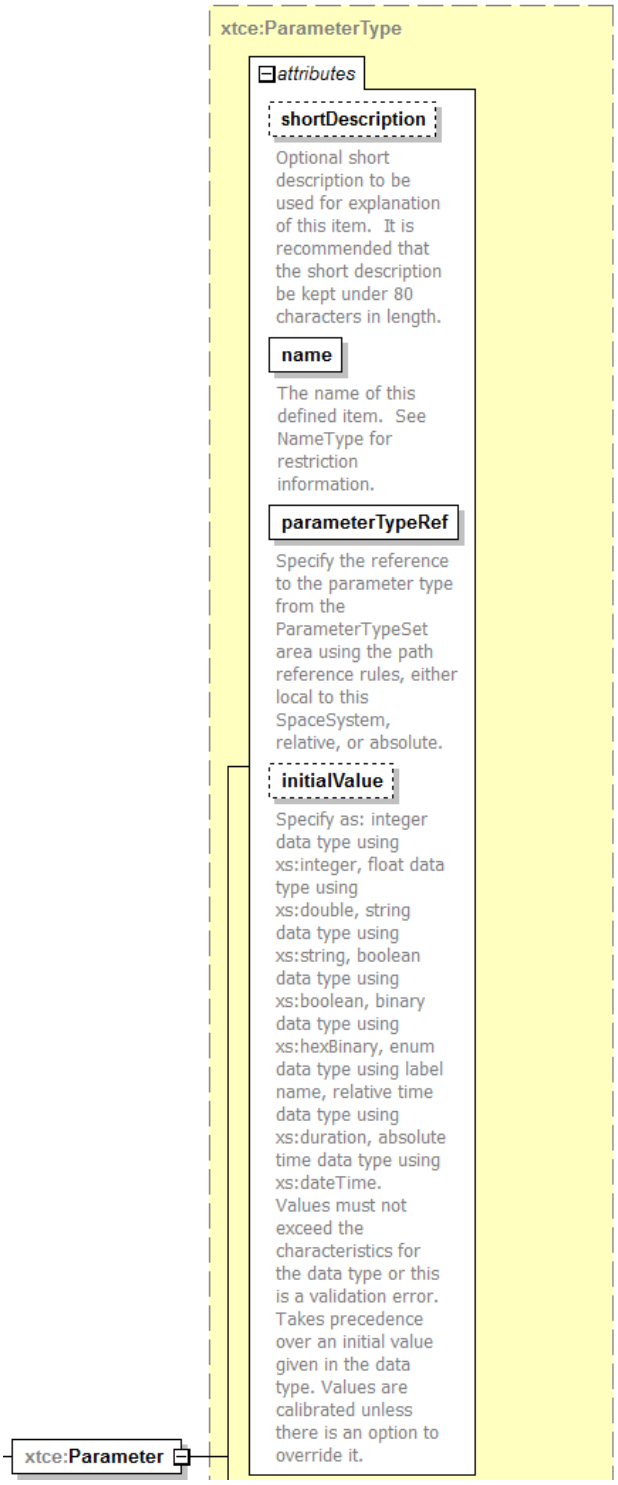



**Figure 4-66: ParameterSet Parameters—Part 1**

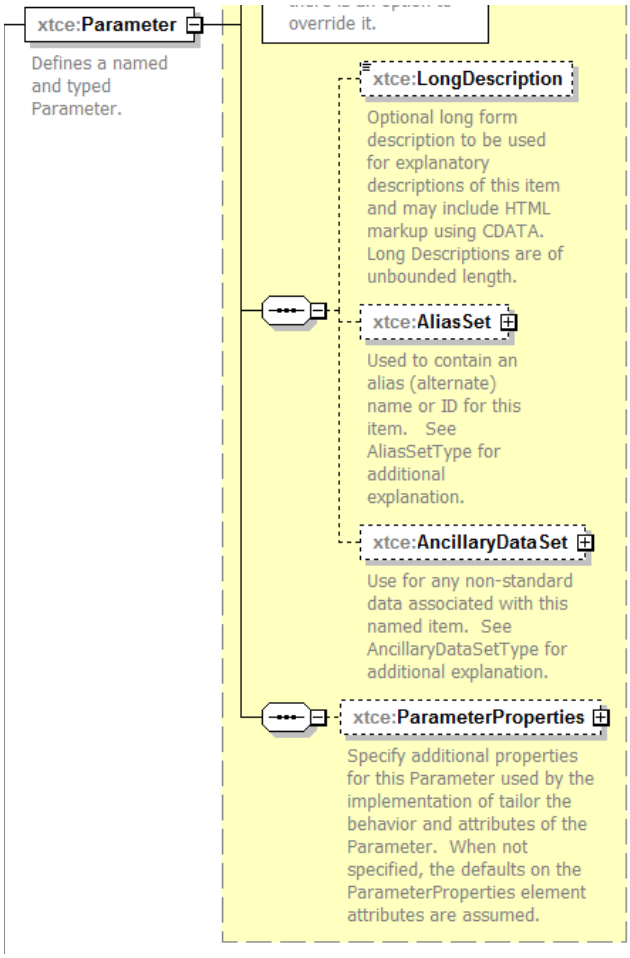



**Figure 4-67: ParameterSet Parameters—Part 2**

The principal differences in XTCE 1.2 are the addition of the documentation-related elements and the expansion of the annotation associated with initialValue.

Finally, in ParameterProperties, the element TimeAssociation has been added to the EntryList elements, while existing here as well.

#### 4.3.3.2 NameDescriptionType

Subsection 3.4.2 contains more information on the NameDescriptionType element and its attributes of name, shortDescription, LongDescription, AliasSet, and AncillaryDataSet.

#### 4.3.3.3 parameterTypeRef Attribute

The parameterTypeRef attribute is the NameReference to the ParameterType for this Parameter. It may be qualified or unqualified.

#### 4.3.3.4 initialValue Attribute

Any supplied initialValue should be interpreted as a format compatible with the referenced ParameterType. If initialValue is set in Parameter, it will override any initialValue in the ParameterType.

A typical Parameter construction simply consists of the Parameter's name and parameterTypeRef, as shown in this example:

```
<xtce:Parameter name="MyParameter" parameterTypeRef="MyParameterType"/>
```

#### 4.3.3.5 ParameterProperties

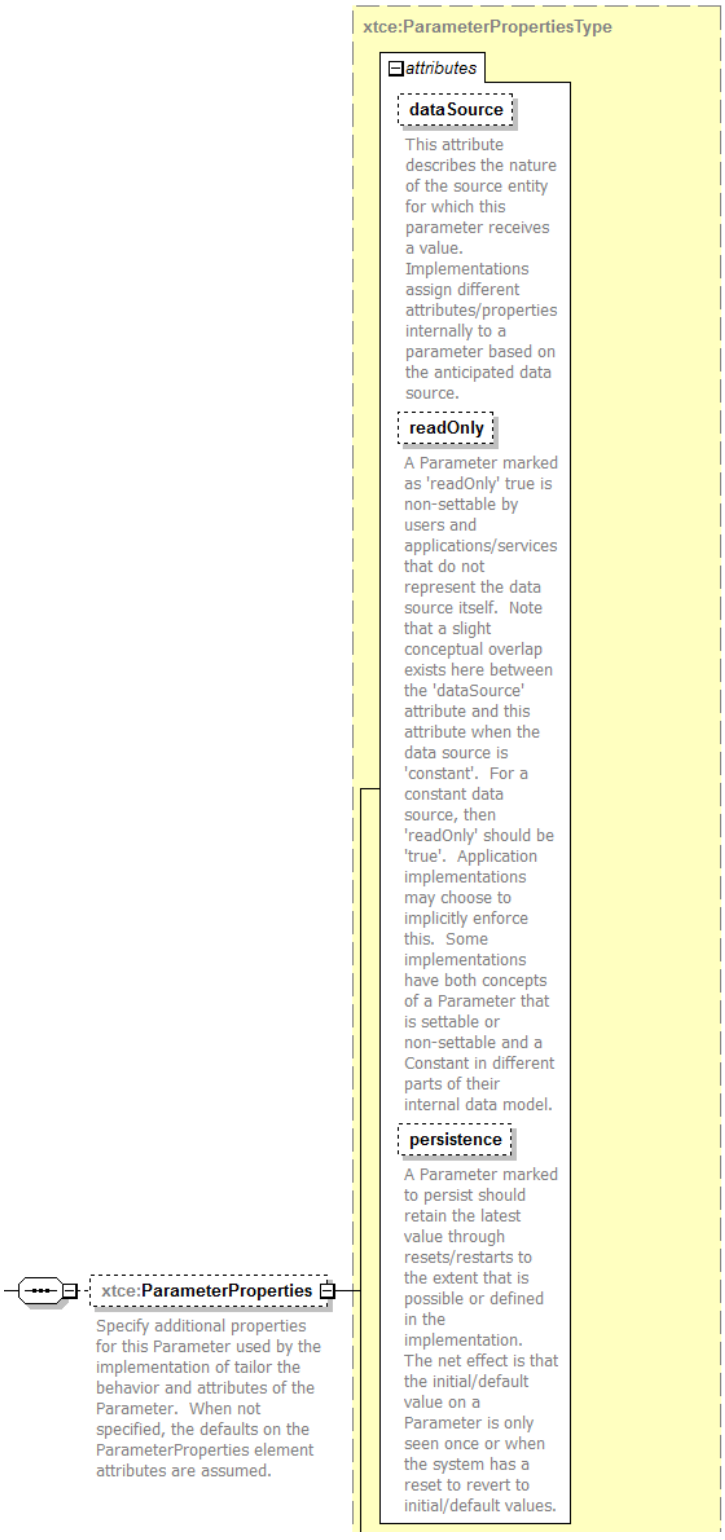



**Figure 4-68: ParameterProperties—Part 1**

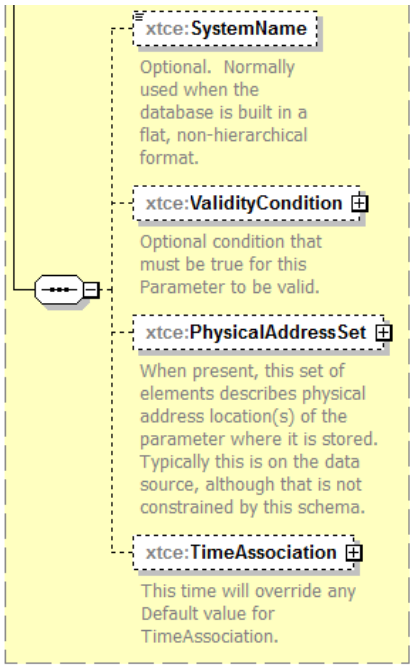



**Figure 4-69: ParameterProperties—Part 2**

### 4.3.3.6 dataSource Attribute

The dataSource attribute is used to further describe a telemetry parameter's origin.

For telemetry:

- telemetered: a product of telemetry; a parameter is considered telemetered unless it is a session variable;
- derived: a product of an algorithm using other telemetered parameters as input;
- constant: an unchanging telemetered value;
- local: a nontelemetered parameter such as a session variable;
- telemetered/derived/local/constant: should be either not set or ignored.

### 4.3.3.7 readOnly Attribute

For telemetry, setting the readOnly attribute to 'true' indicates that no user or system process can override its value once the item has started to be received. The initialValue should still be applied first if it is specified.

#### 4.3.3.8 persistance Attribute

The persistance Attribute is an implemenation-specific value indicating that the parameter value should persist through system cycles.

#### 4.3.3.9 SystemName Element

The SystemName element holds the subsystem or system name if SpaceSystem hierarchies are not being used; that is, it is used if a single SpaceSystem is defined and all the information is in a 'flat file'. This allows for a single SpaceSystem and one level of named subsystems. Parameters, Arguments, and MetaCommands all have this element; however, Telemetry containers do not, so they must be marked with AncillaryData.

#### 4.3.3.10 ValidityCondition Element

The element ValidityCondition specifies optional conditions that must be true for the telemetry to be valid. For example, in the case in which one telemetry parameter indicates that a particular subsystem is either 'On' or 'Off', if its value is 'Off', then a second parameter's values are invalid until the first parameter is set to 'On'. (See 3.4.3.6 for a description of MatchCriteria.)

#### 4.3.3.11 PhysicalAddressSet Element

The PhysicalAddressSet element is a way to capture hardware addresses associated with some forms of telemetry. What constitutes a hardware address is user defined.

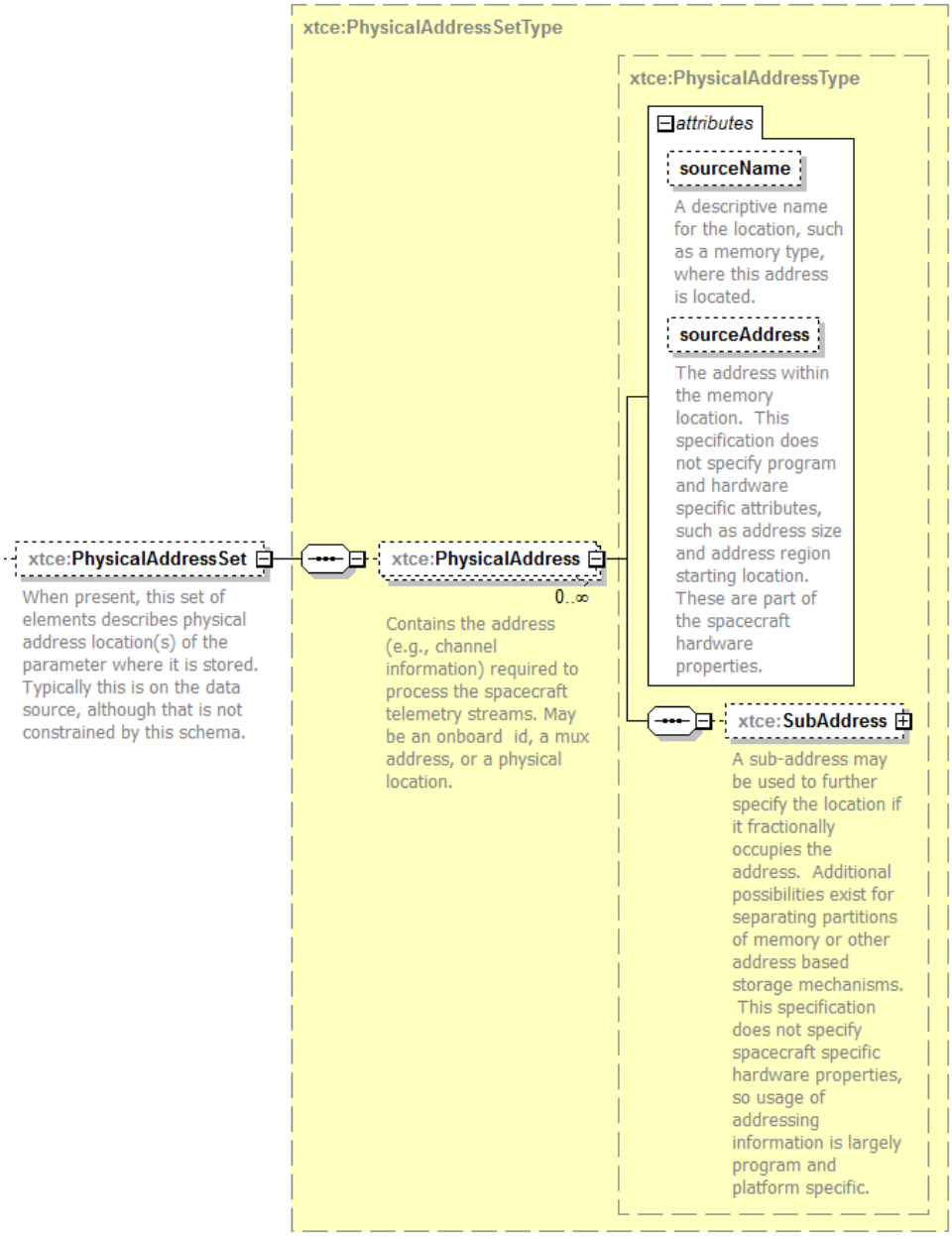



**Figure 4-70: PhysicalAddress**

```
<xtce:PhysicalAddress sourceName="RT" sourceAddress="5">
    <xtce:SubAddress sourceName="SA" sourceAddress="27"/>
</xtce:PhysicalAddress>
```

In the example above, the telemetered item is from a MIL-STD-1553 bus. Details about the bus address are captured using PhysicalAddress.

### 4.3.3.12 TimeAssociation Element

#### 4.3.3.12.1 General

The TimeAssociation element is used to capture offset in time within a packet or minor frame from a known timestamp.

NOTE – For XTCE 1.2, this now also exists within the EntryList elements. This is mainly because a parameter may be shared by more than one container, and so any TimeAssociation offset may be different per container. If this element is present, it overrides any in the EntryList.

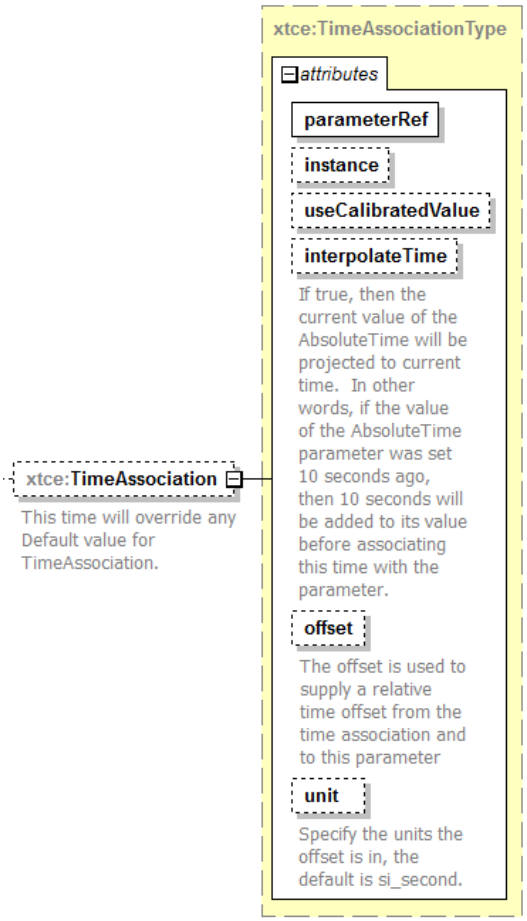



**Figure 4-71: TimeAssociation**

#### 4.3.3.12.2 parameterRef Attribute

The parameterRef attribute specifies a time reference type.

#### 4.3.3.12.3 instance Attribute

The instance attribute specifies which recorded value or instance number to use. The parameterRef is actually an instance reference. (See 3.4.4 for a discussion of ParameterInstanceReferences.)

#### 4.3.3.12.4 useCalibratedValue Attribute

The useCalibratedValue attribute is part of the instance reference. It is used by an event that refers to a time parameter that uses a calibrator. The value of this attribute is set to ‘true’ or ‘false’.

#### 4.3.3.12.5 interpolateTime Attribute

The interpolateTime attribute is used to match the current absolute time stamp to the wall clock time.

#### 4.3.3.12.6 offset Attribute

The offset attribute specifies the delta offset from a known absolute time stamp. In XTCE 1.2., the data type for this attribute is now a double, in the units in the next attribute.

#### 4.3.3.12.7 unit Attribute

The unit Attribute specifies the offset unit; the default unit is seconds.

### 4.3.4 CONTAINERSET—CONTAINERS AND TELEMETRY PACKETS

#### 4.3.4.1 General

ContainerSet holds SequenceContainers (i.e., ‘containers’) that can be used to describe CCSDS packets, minor frames, or other data blocks.

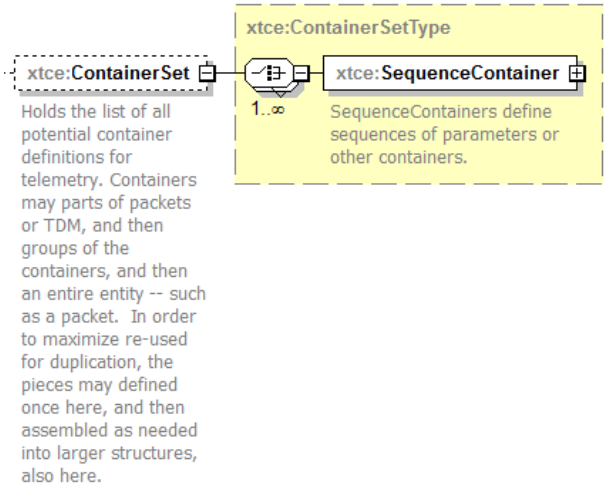



**Figure 4-72: ContainerSet**

A single telemetry packet may use several SequenceContainers to form a single packet definition. For example, one container may be used to describe the packet header (i.e., CCSDS Primary Header), while another container may be used to describe the packet body. Both containers can then be combined to form another container.

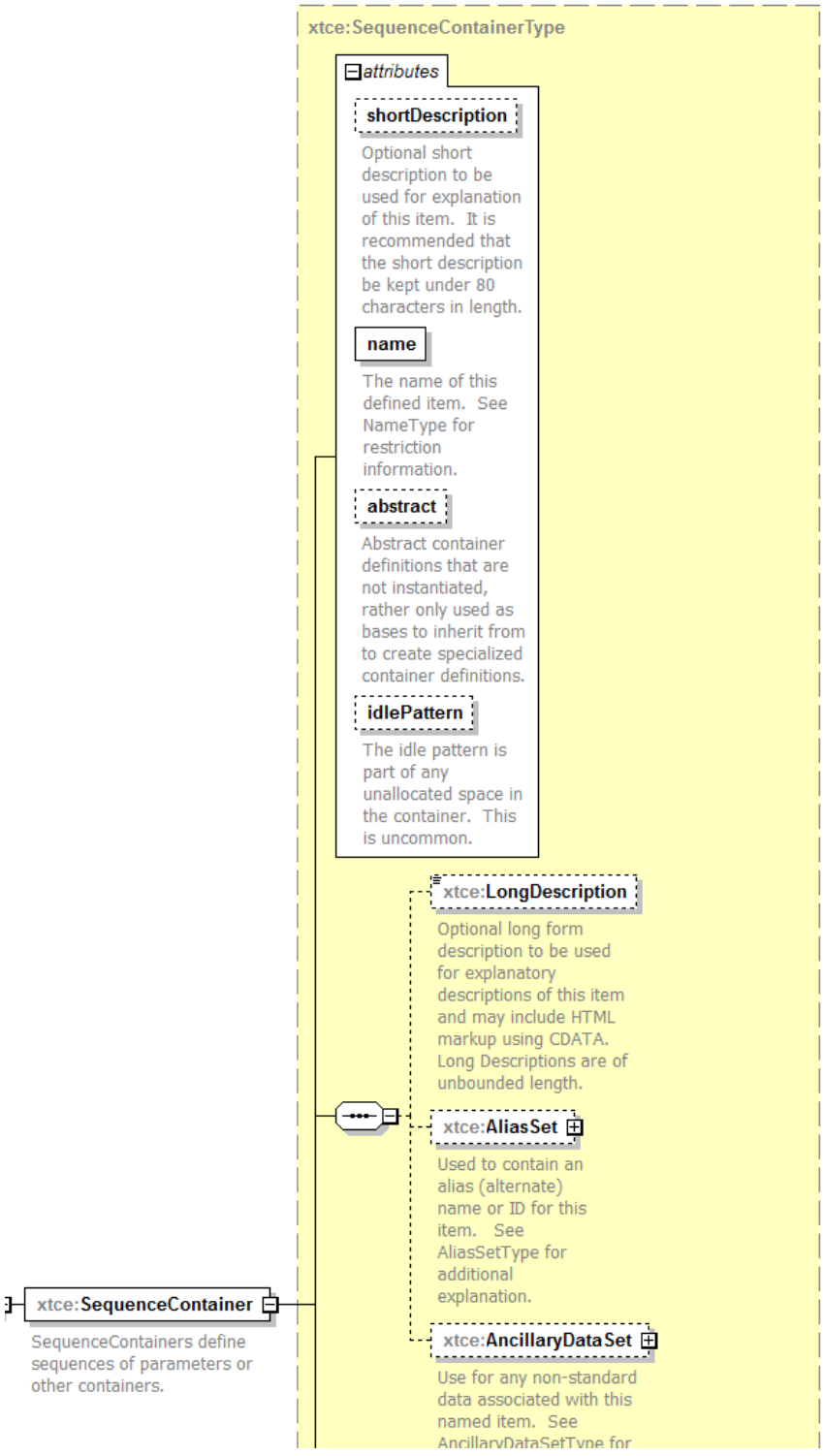



**Figure 4-73: The SequenceContainer Element—Part 1**

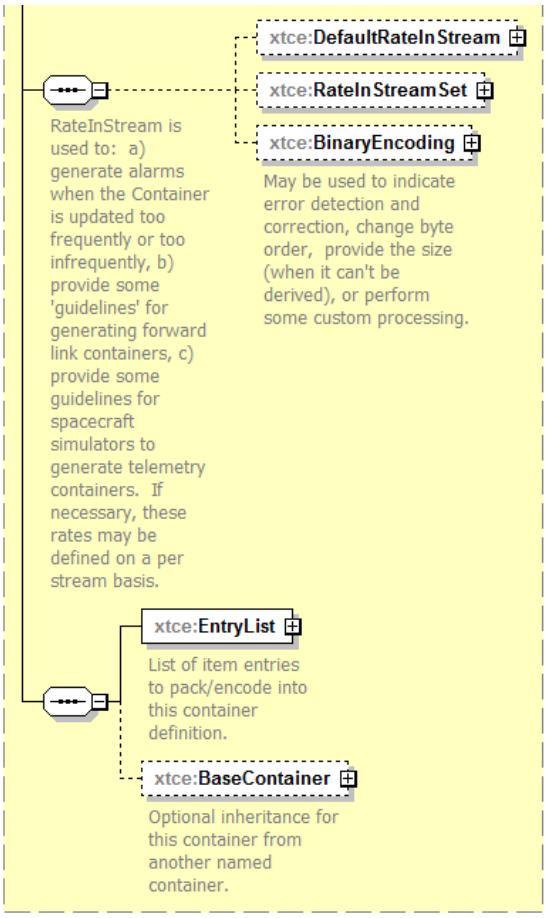



**Figure 4-74: The SequenceContainer Element—Part 2**

The SequenceContainer element lists the parameters in the packet using EntryList. The default is to list the parameters in the order they appear.

#### 4.3.4.2 NameDescription

Subsection 3.4.2 contains a description of the elements and attributes associated with name, shortDescription, LongDescription, AliasSet, and AncillaryDataSet.

#### 4.3.4.3 abstract Attribute

The abstract attribute is used in conjunction with container inheritance to specify that the container represents an abstraction or generic form. This flag is often set in an inheritance hierarchy until the construction is complete. For example, when defining a CCSDS packet with several levels of inheriting containers, all packets will be set to ‘abstract’ until the final container. These concepts are more fully explored in subsections below.

#### 4.3.4.4 idlePattern Attribute

The idlePattern attribute is the expected container pattern when no telemetry is received. For CCSDS missions, this applies to special idle packets, as shown in the following example:

```
<xtce:SequenceContainer name="IdlePacket" idlePattern="0xabba505">
    <xtce:EntryList/>
    <xtce:BaseContainer containerRef="Header">
        <xtce:RestrictionCriteria>
            <xtce:Comparison parameterRef="ID" value="2047"/>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

#### 4.3.4.5 DefaultRateInStream Element

The DefaultRateInStream is an expected receive data rate for the container. The user defines the action taken if the expected rate is out of range.

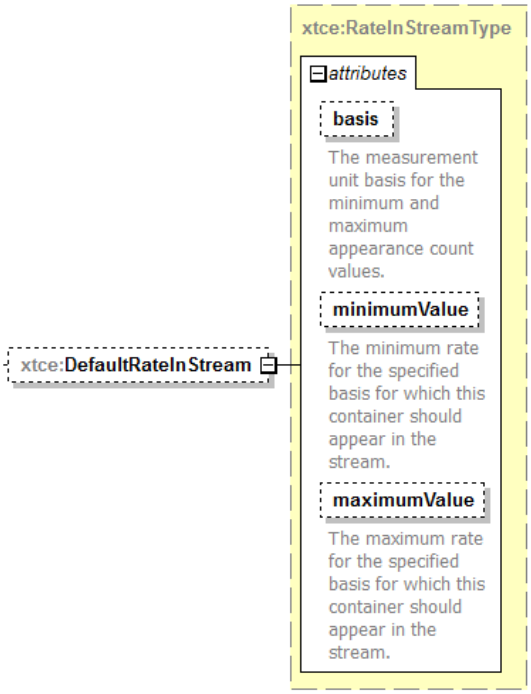



**Figure 4-75: DefaultRateInStream**

The following is an example of DefaultRateInStream:

```
<xtce:SequenceContainer name="SlowPacket">
    <xtce:DefaultRateInStream basis="perSecond" minimumValue="1.5"
   maximumValue="0.5"/>
    <xtce:EntryList/>
```

```
        <xtce:BaseContainer containerRef="CCSDSHeader">
            <xtce:RestrictionCriteria>
                <xtce:Comparison parameterRef="APID" value="20"/>
            </xtce:RestrictionCriteria>
        </xtce:BaseContainer>
    </xtce:SequenceContainer>
```

### 4.3.4.6 RateInStreamSet Element

The element RateInStreamSet is shown in figure 4-76.

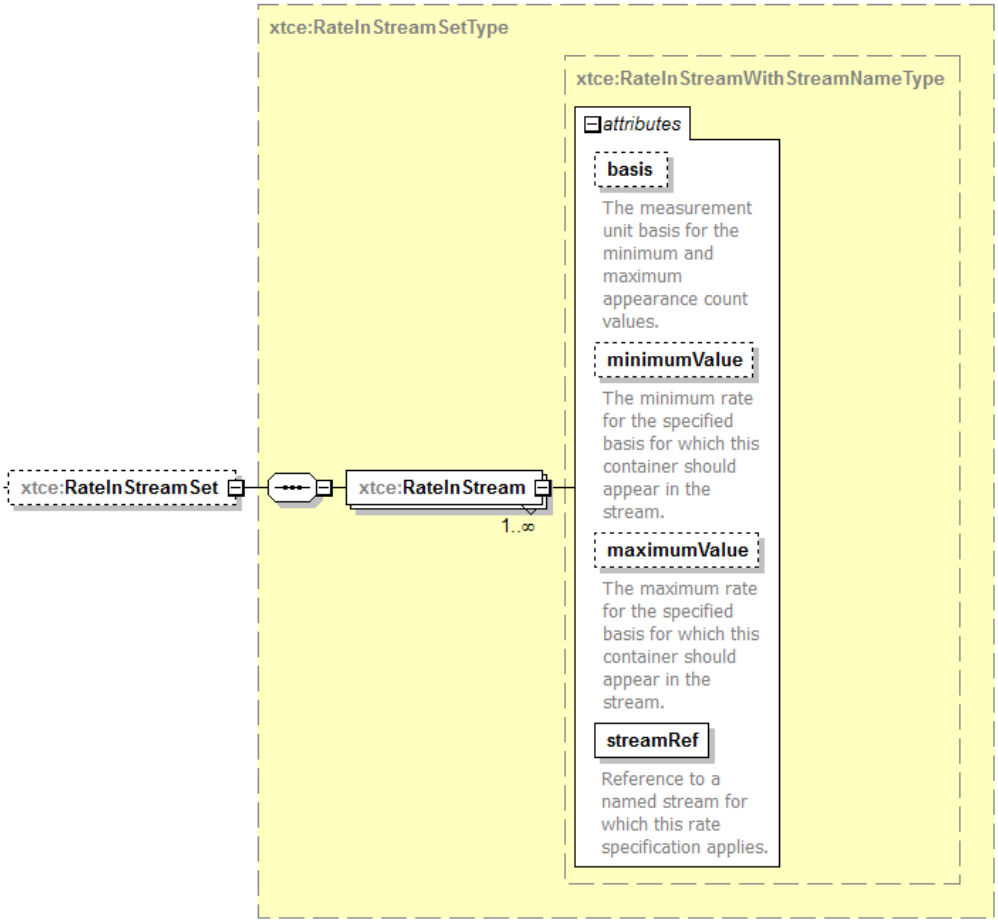



**Figure 4-76: RateInStreamSet Element**

A set of expected rates may be specified and associated with a stream (see 4.3.6 for an explanation of StreamSet).

```
    <xtce:SequenceContainer name="SlowPacket">
        <xtce:RateInStreamSet>
            <xtce:RateInStream streamRef="XBand" basis="perSecond" minimumValue="1"
            maximumValue="1"/>
```

```
            <xtce:RateInStream streamRef="TDRSS" basis="perSecond" minimumValue="2"
            maximumValue="2"/>
        </xtce:RateInStreamSet>
        <xtce:EntryList/>
        <xtce:BaseContainer containerRef="CCSDSHeader">
            <xtce:RestrictionCriteria>
                <xtce:Comparison parameterRef="APID" value="20"/>
            </xtce:RestrictionCriteria>
        </xtce:BaseContainer>
    </xtce:SequenceContainer>
```

Streams in this example could be defined to represent different communication paths for the container.

#### 4.3.4.7 BinaryEncoding Element

The BinaryEncoding element is used to explicitly set the size of the container. It may be used to indicate error detection and correction, change byte order, provide the size, or perform some custom processing. It has the same content as BinaryDataEncoding.

NOTES

1 The Container/BinaryEncoding/@bitOrder and BinaryEncoding/ByteOrderList should be ignored.

2 Subsection 3.4.5 discusses use of the SizeInBits element (a child element of BinaryEncoding) to set the size value.

3 Depending on the complexity of the entries (i.e., variable length), it may be easier to derive the size of the container by the bit lengths of its entries.

#### 4.3.4.8 EntryList

##### 4.3.4.8.1 General

A container's EntryList is used to describe its contents by referencing parameters, containers, and streams.

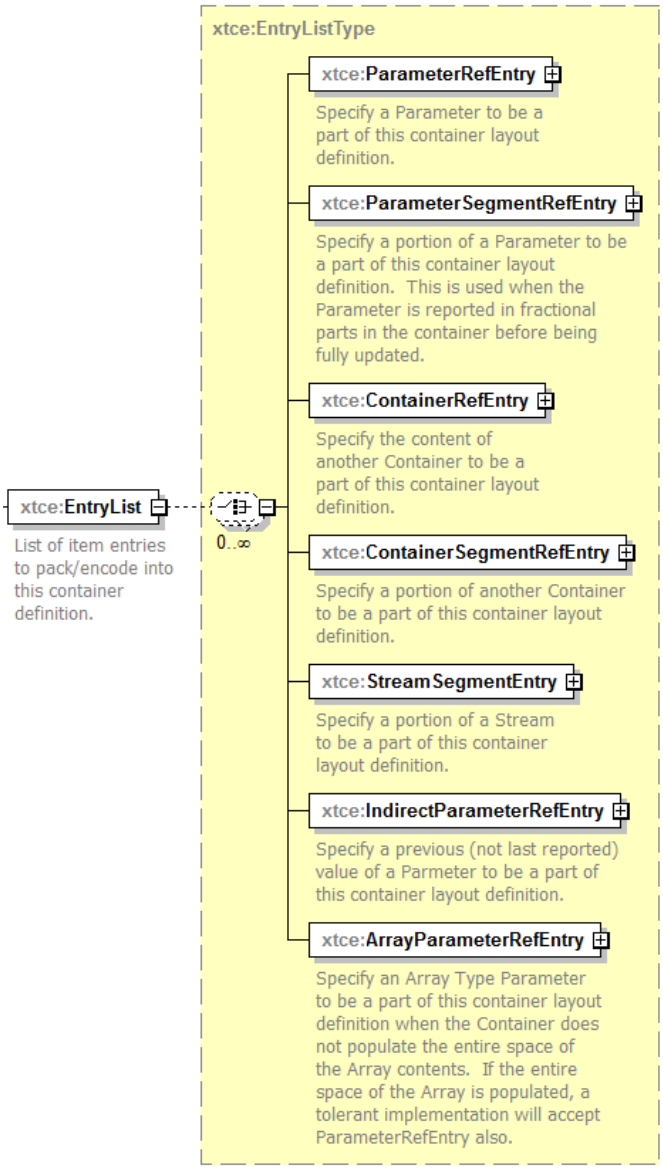



**Figure 4-77: The SequenceContainer EntryList**

The size of a container is calculated by summing the lengths of the DataEncodings (or segment sizes) of the parameters specified in the EntryList. The various child elements described below need to be included.

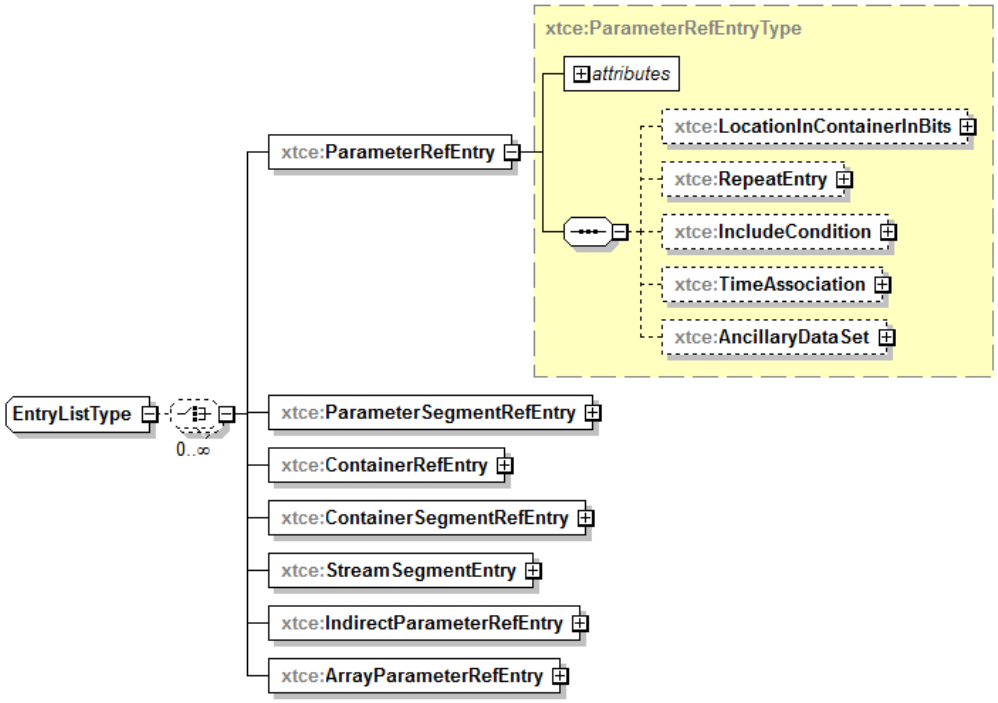



**Figure 4-78: EntryList Elements and Pattern**

#### 4.3.4.8.2 EntryList Entry Pattern

The EntryList elements are very similar in construction and follow a pattern like the one shown in figure 4-78 for ParameterRefEntry. All the items in EntryList are 'RefEntry'-named items.

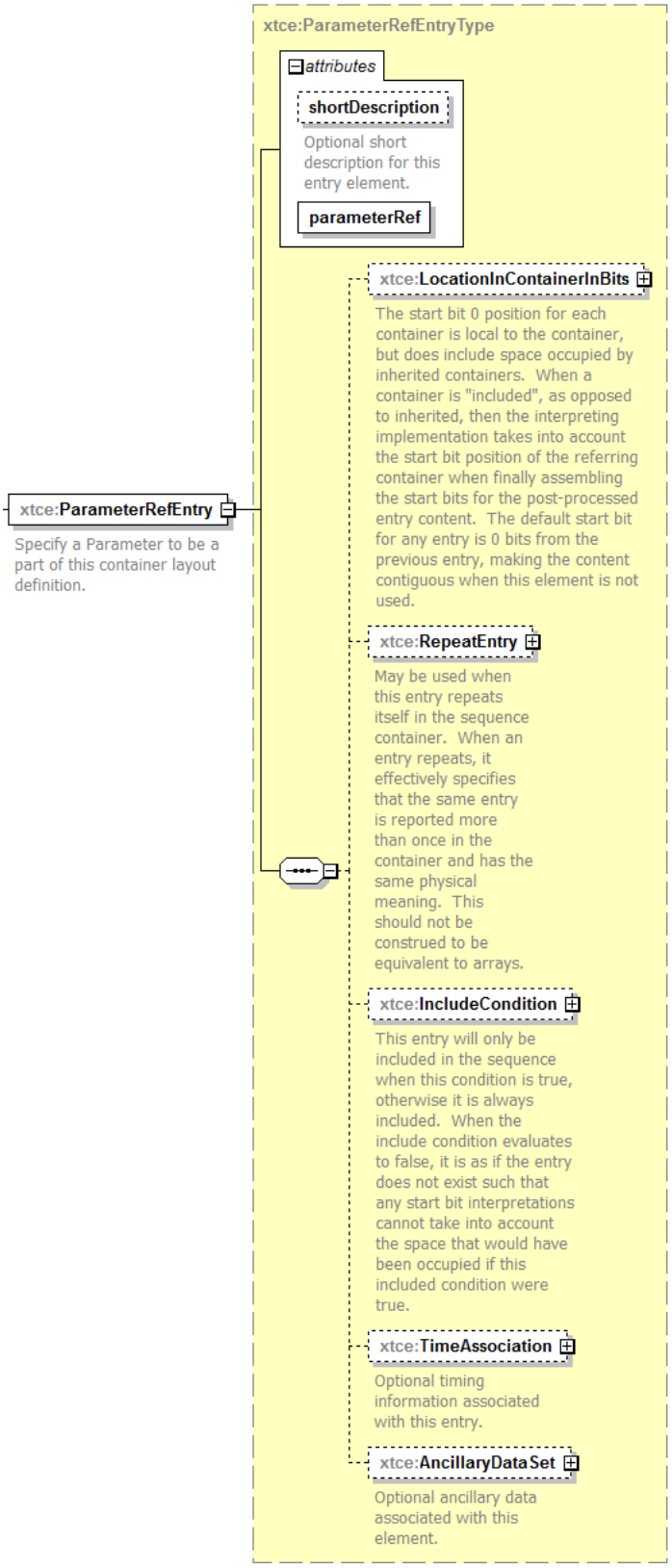



**Figure 4-79: Inside of ParameterRefEntries**

a) Each entry contains an attribute for a NameReference (see 3.4.2) and then the elements LocationInContainerInBits, RepeatEntry, and IncludeCondition:

   – LocationInContainerInBits allows for the explicit setting of an entry's address (see 4.3.4.8.10.2);

   – RepeatEntry allows the description of a multiple-sampled ('super-sampled' or 'super-commutated') parameter (see 4.3.4.8.10.3);

   – IncludeCondition allows for the optional inclusion of an item in the EntryList given some condition (see 4.3.4.8.10.4).

b) By default (without any specification of LocationInContainerInBits), items in the EntryList are assumed to be 'packed' together based on their bit widths.

c) Using this information, the size of the container can be derived. If also specified in the BinaryEncoding/SizeInBits element, then the two values should match.

Examples of EntryLists are shown in 5.2.

#### 4.3.4.8.3 ParameterRefEntry Element

ParameterRefEntry specifies a parameter defined in a ParameterSet in SequenceContainer.

```
<xtce:EntryList>
    <xtce:ParameterRefEntry parameterRef="myParameter1"/>
    <xtce:ParameterRefEntry parameterRef="myParameter2"/>
</xtce:EntryList>
```

The bit width of the parameter is not set here; it is taken from the ParameterType encoding area as has been shown in 4.3.2.

#### 4.3.4.8.4 ParameterSegmentRefEntry

The segmented version of ParameterRefEntry is used to define the segment's width.

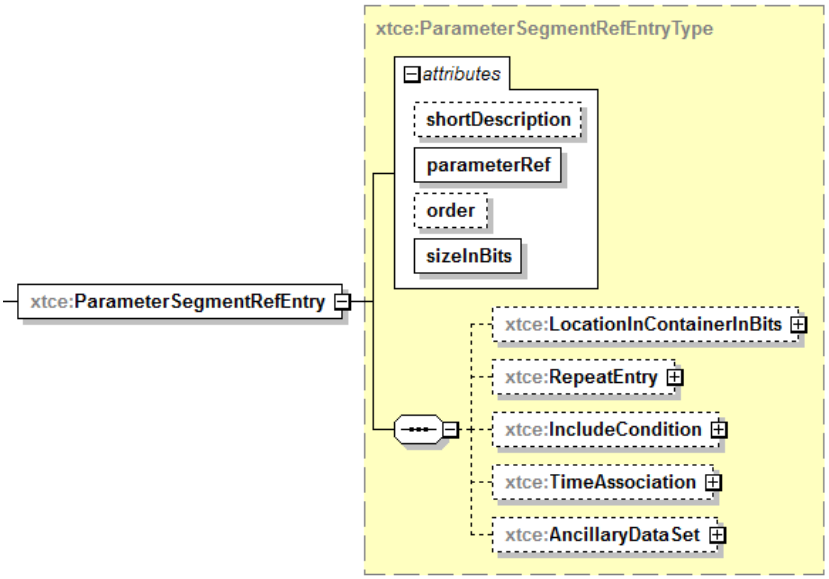



**Figure 4-80: ParameterSegmentRefEntry**

If the actual width of the parameter underflows the width specified in the SegmentEntry, then an error or warning should be produced. The following example shows a 1024-bit BinaryParameter segmented to 116 bits.

```
<xtce:ParameterSegmentRefEntry parameterRef="P1_BLOB_1024" sizeInBits="116"/>
```

#### 4.3.4.8.5 ContainerRefEntry

##### 4.3.4.8.5.1 General

The ContainerRefEntry is used to include another container defined in ContainerSet.

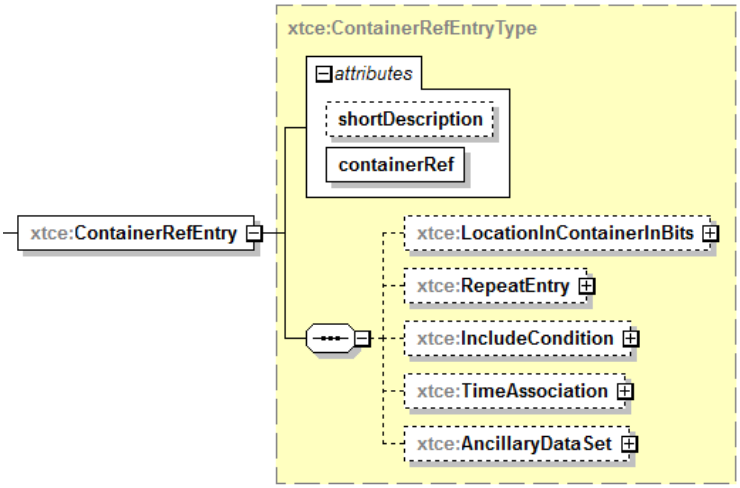



**Figure 4-81: ContainrRefEntry**

The container being referred to should be treated as a single unit with its addresses resolved first. The ContainerRefEntry may have LocationInContainerInBits or Repeat elements set. If set, these should be treated against the entire container being inserted into the EntryList.

#### 4.3.4.8.5.2 Dynamic Container Matching

Dynamic container matching is a special XTCE object-oriented case of ContainerRefEntry. It occurs when a container being referred to is an abstract container, and it has at least one derived concrete container.

a) If this occurs, then the derived container is inserted if its RestrictionCriteria matches during real-time processing of the incoming stream. Since it is part of the inheritance chain, its EntryList and its parent's EntryList are inserted.

b) If no match occurs, and the original abstract container being referred to has content in its EntryList, then that container is treated as any other container and is included.

c) If multiple derived containers of that abstract container match, all will be dynamically included as a union.

Subsection 5.4.4 contains full explanation of this concept.

### 4.3.4.8.6 ContainerSegmentRefEntry

The segmented version of ContainerRefEntry is similar to the other segmented entries.

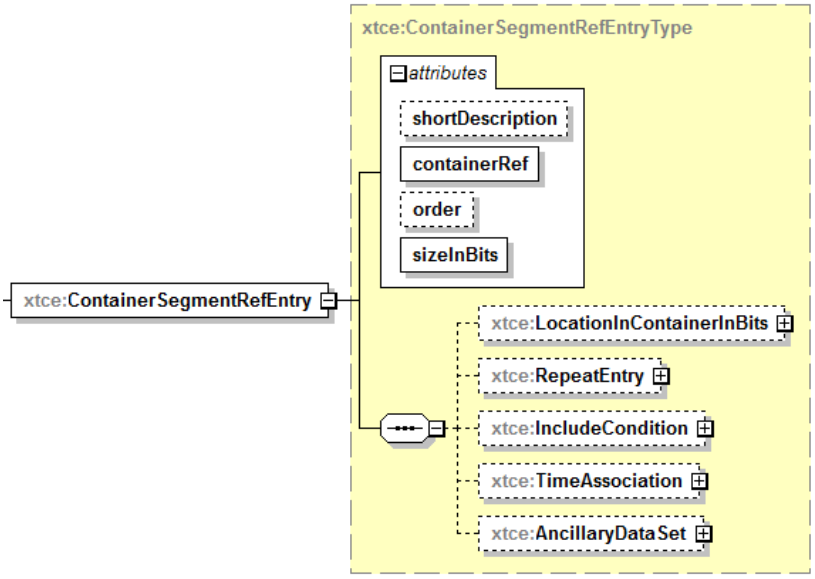



**Figure 4-82: ContainerSegmentRefEntry**

The segment size is determined by reducing from the end of the referenced container. If the referenced container's size underflows the specified segment, an error or warning should be

produced. In the following example, the image is too large and will be segmented across more than one container. The optional order attribute can be used to mark the order of the segments.

```
<xtce:ContainerSegmentRefEntry containerRef="LARGE_CCD_IMAGE" sizeInBits="32768"/>
```

#### 4.3.4.8.7 StreamSegmentRefEntry

A portion of a stream reference by a streamRef is included in the EntryList.

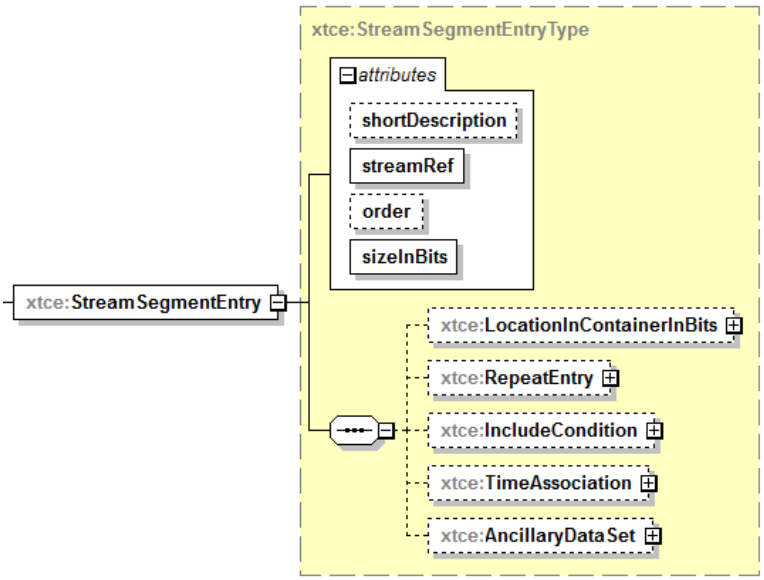



**Figure 4-83: StreamSegmentEntry**

a) The order attribute is optional and is used to mark the order of the segments in several containers.

b) The sizeInBits attribute must be set.

```
<xtce:StreamSegmentEntry streamRef="VID_FRAME_22of1024" order="21"
sizeInBits="1024"/>
```

#### 4.3.4.8.8 IndirectParameterRefEntry

An IndirectParameterRefEntry reads a parameter instance that contains the name of the parameter to include in the container.

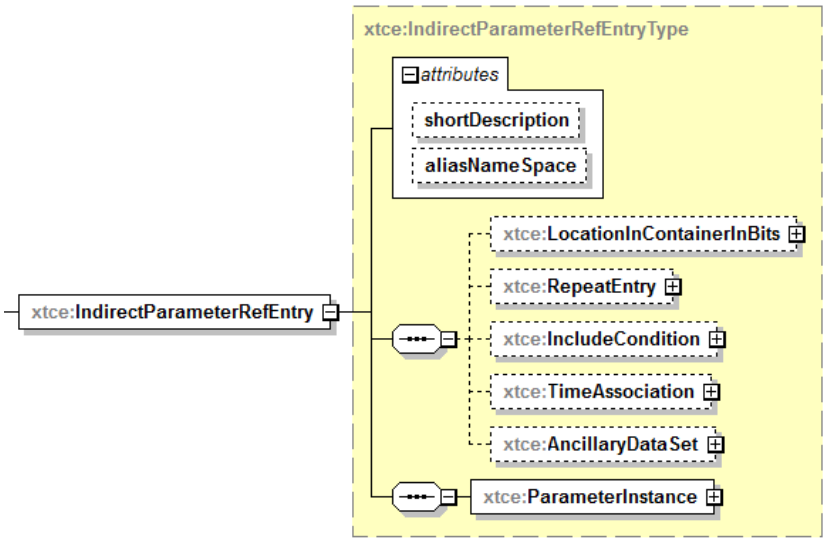



**Figure 4-84: IndirectParameterRefEntry**

This entry may be used to implement telemetry streams. The value of the parameter in parameter instance must use either the name of the parameter or its alias. If it is an alias, the alias namespace is supplied as an attribute.

```
<xtce:IndirectParameterRefEntry>
    <xtce:ParameterInstance parameterRef="ParameterToLookupStatusParameter"/>
</xtce:IndirectParameterRefEntry>
```

The ParameterInstance should hold the name of a parameter that is a NameReference. This is the only location in the schema that states that an alias can be used in a NameReference. It implies the attribute @aliasNameSpace must be set and match the Alias attribute @nameSpace to the alias it references.

#### 4.3.4.8.9 ArrayParameterRefEntry

##### 4.3.4.8.9.1 General

An ArrayParameterRefEntry specifies both the array parameter to include in the container and the size of its dimensions.

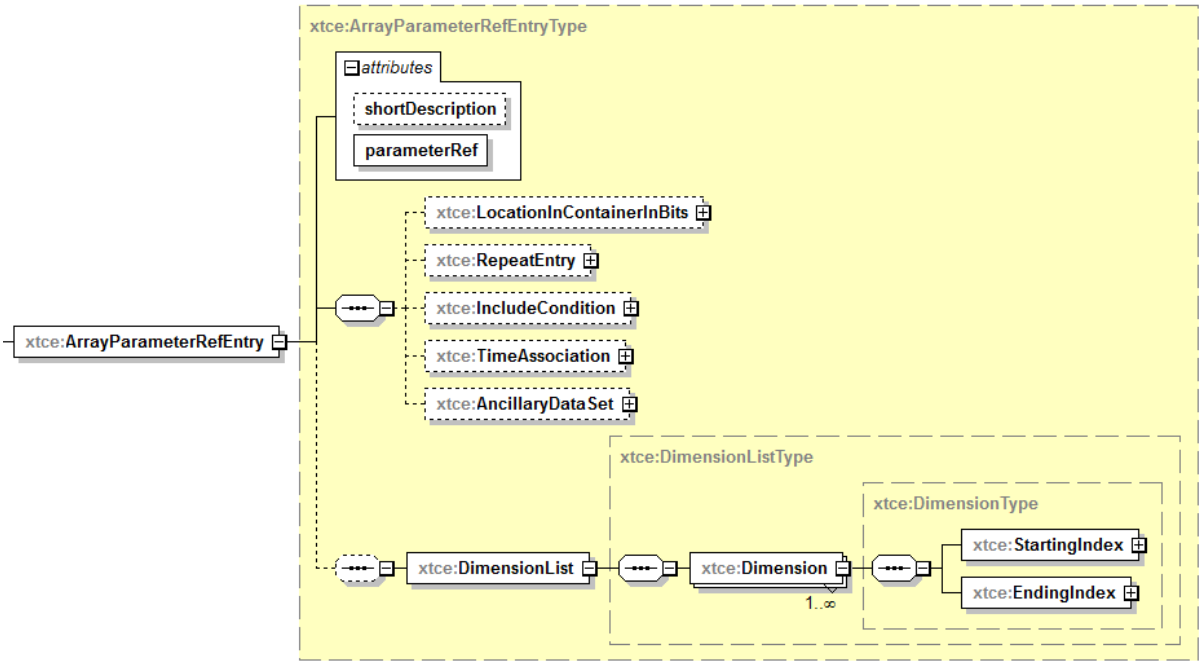



**Figure 4-85: ArrayParameterRefEntry**

The dimension list form is: Array[1stDim][2ndDim][lastDim]. The last dimension is assumed to be the least significant. The order must ascend, or the array will need to be broken out entry by entry.

Partial dimensions are supported. Usage will be similar to the following example that defines the dimension from zero to three (four items):

```
<xtce:ArrayParameterRefEntry parameterRef="OneDimArray">
    <xtce:DimensionList>
        <xtce:Dimension>
            <xtce:StartingIndex>
                <xtce:FixedValue>0</xtce:FixedValue>
            </xtce:StartingIndex>
            <xtce:EndingIndex>
                <xtce:FixedValue>3</xtce:FixedValue>
            </xtce:EndingIndex>
        </xtce:Dimension>
    </xtce:DimensionList>
</xtce:ArrayParameterRefEntry>
```

It is up to the implementations that exchange the data to designate whether the definitions refer to row major order or column major order.

The dimension start and end can also be supplied by a ParameterInstance. This makes them dynamic.

#### 4.3.4.8.10 Modifying Entries Elements

##### 4.3.4.8.10.1 General

There are three child elements that can be used to modify most entries: LocationInContainerInBits, RepeatEntry, and IncludeCondition.

##### 4.3.4.8.10.2 LocationInContainerInBits

##### 4.3.4.8.10.2.1 General

LocationInContainerInBits allows one to modify the address entry. There are several modifiers associated with this element; each is a form of relative addressing or absolute addressing relative to the container in which it is defined. (See 5.3 for a fuller discussion of calculating EntryList addresses.)

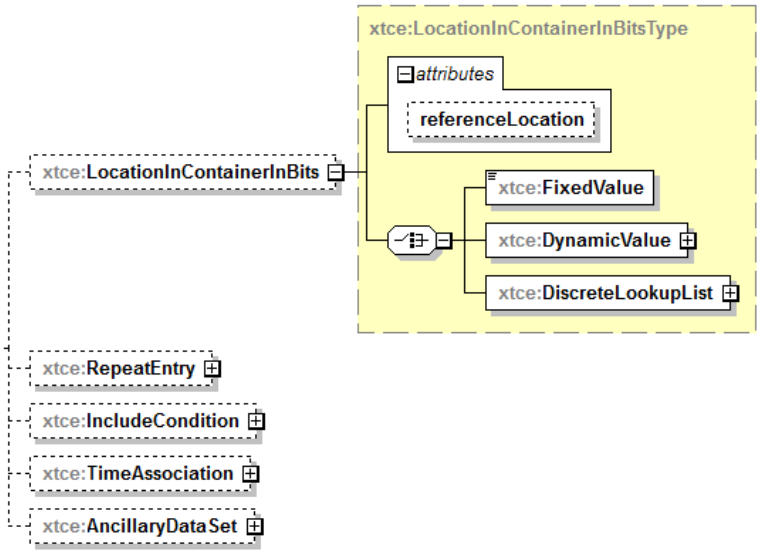



**Figure 4-86: The LocationInContainerInBits Element**

##### 4.3.4.8.10.2.2 referenceLocation Attribute

Two of these are forms of absolute addressing, and two are forms of relative addressing:

- containerStart: offset from the start of container;
- containerEnd: offset from the end of the container;
- prevEntry: offset from prevEntry (default if not specified);
- nextEntry: offset from the nextEntry.

The containerStart and prevEntry are intuitive and easy to implement. The containerStart address is calculated from zero. The prevEntry address is from the entry ahead of it. The default (unspecified) treats all entries as addressing from a prevEntry of zero. They are all placed 'back to back' without gap.

The use of both containerEnd and nextEntry depends on a known fixed size or range. It can be the end of the container or the location of a particular entry in that container.

#### 4.3.4.8.10.3 RepeatEntry—Super-Sampling/Super-Commutated

##### 4.3.4.8.10.3.1 General

RepeatEntries are super-sampled or 'super-commutated' values. They are samples of the same memory location over time.

In the EntryList area, every entry has the child element RepeatEntry, which may be used to describe super-sampled parameters or groups of parameters.

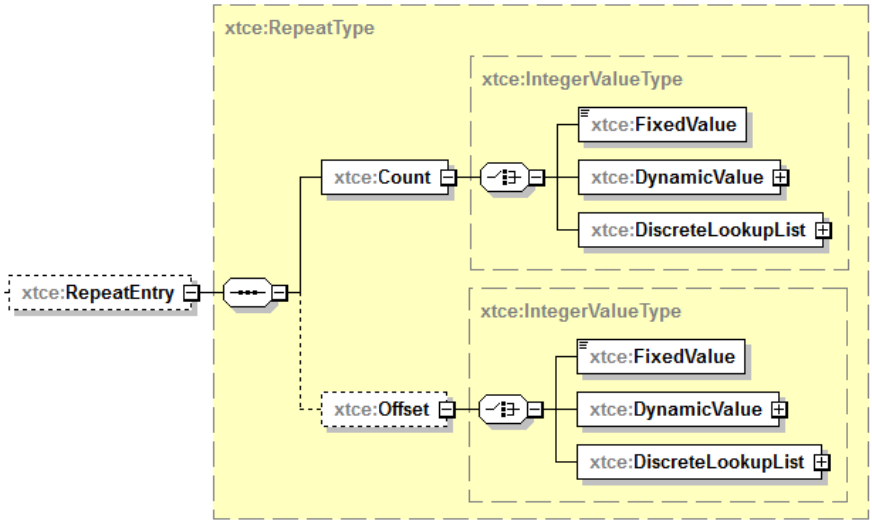



**Figure 4-87: RepeatEntry Element**

The RepeatEntry element specifies the count (or the number of repeats) of an item. The optional offsetSizeInBits specifies the offset between repeats (defaults to one).

##### 4.3.4.8.10.3.2 Count Element

The Count element specifies the number of repeats. If the count is one, the entry repeats one time, resulting in two entries. A count of two means the entry repeats two additional times, resulting in three entries.

#### 4.3.4.8.10.3.3 Offset Element

##### 4.3.4.8.10.3.3.1 General

The Offset element specifies a gap between the entries.

#### 4.3.4.8.10.4 IncludeCondition Element

The IncludeCondition conditionally includes the entry. It can be a parameter, container, or stream based on comparisons. The IncludeCondition uses MatchCriteria (see 3.4.3.6).

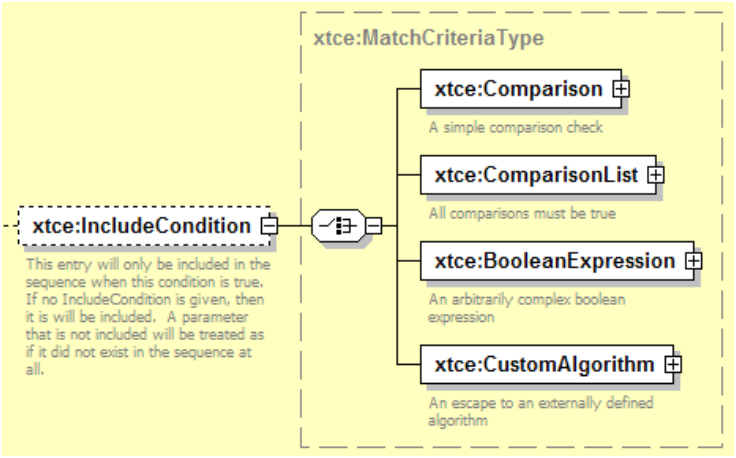



**Figure 4-88: IncludeCondition Element**

### 4.3.4.9 BaseContainer Element—Container Inheritance

#### 4.3.4.9.1 General

A container may extend another container in an object-oriented manner. The child container receives the parent's EntryList, and that EntryList is exactly prefixed to the front of the child's EntryList.

The main purpose of container inheritance is to build up a single EntryList from a set of containers.

Included with BaseContainer is a mechanism for specifying constraints. These are defined in the element RestrictionCriteria, which are conditional expressions for telemetry that identify specific locations in the raw packets or minor frames and their expected values. If the values are found, then the rest of the description can be used to fully decommutate the packet.

The RestrictionCriteria element is used to name the identification areas and their expected values in a telemetry format (such as certain fields in a header). The specific XTCE container description can then be matched and associated with it. Subsection 5.2 contains a full discussion of XTCE container inheritance. The basic rules are presented below.

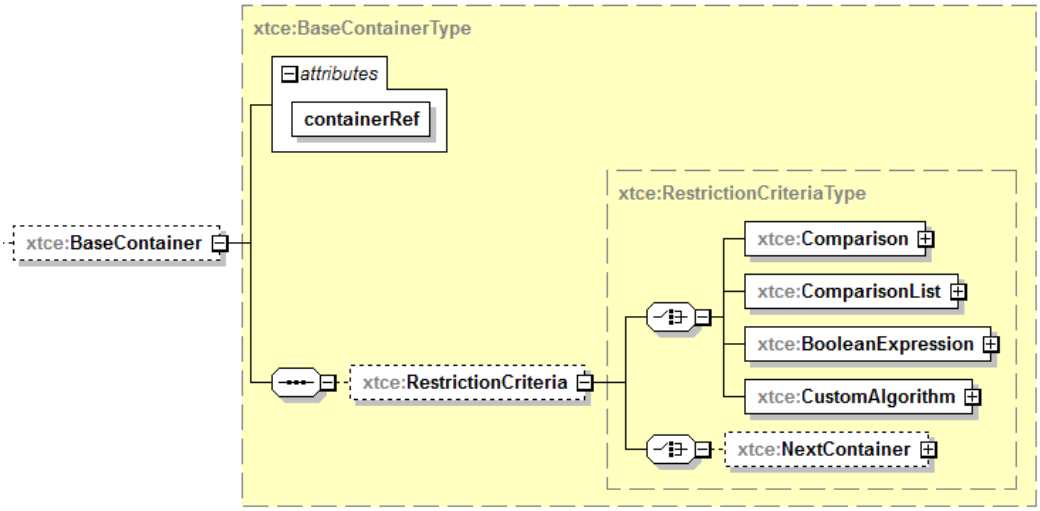



**Figure 4-89: BaseContainer Element**

a) A TelemetryMetaData/ContainerSet/SequenceContainer may extend another TelemetryMetaData/ContainerSet/SequenceContainer. Figure 4-90 shows how a ContainerSet/SequenceContainer may be extended; in this case, a SequenceContainer extends another SequenceContainer.

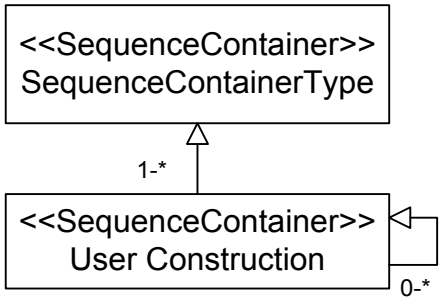



**Figure 4-90: Extending SequenceContainers**

b) User-defined SequenceContainers may be extended any number of times to create new constructions as needed. It is often confusing to extend outside of a SequenceContainer, but this may be necessary for shared constructs such as headers.

#### 4.3.4.9.2 containerRef Attribute

The containerRef attribute is the NameReference of a container that is extended by this container.

#### 4.3.4.9.3 RestrictionCriteria Element

The RestrictionCriteria element is a MatchCriteriaType with the addition of NextContainer. (See 3.4.3.6 for discussion of MatchCriteria.) RestrictionCriteria is used to supply the identifying packet information for a CCSDS packet. The identifying information will be part of the comparison.

The parameters referenced in the RestrictionCriteria are ParameterInstanceRefs (see 3.4.4). In the example below, the following construction is legal, and ID is visible to the RestrictionCriteria comparisons.

```
<xtce:SequenceContainer name="SuperContainer" abstract="true">
    <xtce:EntryList/>
    </xtce:SequenceContainer>
<xtce:SequenceContainer name="DerivedContainer">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="ID"/>
        <xtce:ParameterRefEntry parameterRef="BodyParameterP1"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="SuperContainer">
        <xtce:RestrictionCriteria>
            <xtce:Comparison parameterRef="ID" value="1"/>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

Here DerivedContainer is a SuperContainer, which is an abstraction in this example. All the items of interest are supplied in the DerivedContainer. The constraints refer to these items as well.

It is recommended to put only the primary identifying keys into the constraints to keep the expression simple. Any other items should be moved out of the constraints and into the AncillaryData.

#### 4.3.4.9.4 NextContainer Element

The container that holds this NextContainer element (i.e., the SequenceContainer) is not processed unless the container specified in this element's containerRef attribute is matched. It is used in conjunction with the other comparisons.

```
<xtce:SequenceContainer name="ValuesMayNotBeGoodUntilLatch">
    <xtce:EntryList/>
        <xtce:BaseContainer containerRef="CCSDSPacket">
            <xtce:RestrictionCriteria>
                <xtce:ComparisonList>
                    <xtce:Comparison parameterRef="APID" value="16"/>
                    <xtce:NextContainer containerRef="Latch"/>
                </xtce:ComparisonList>
```

```
            </xtce:RestrictionCriteria>
        </xtce:BaseContainer>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="Latch">
        <xtce:EntryList/>
        <xtce:BaseContainer containerRef="CCSDSPacket">
            <xtce:RestrictionCriteria>
                <xtce:Comparison parameterRef="APID" value="32"/>
            </xtce:RestrictionCriteria>
        </xtce:BaseContainer>
</xtce:SequenceContainer>
```

#### 4.3.4.9.5 Container Inheritance as an Operation

##### 4.3.4.9.5.1 General

Treating container inheritance as an operation means that the child acquires certain elements, attributes, and content from a parent. In some cases, items may be overridden by the child if the information is set in the parent. The result is a new entity that has features of both.

##### 4.3.4.9.5.2 Inheritance Rules

a) A container may extend another using the BaseContainer element.

b) The parent's EntryList is copied to the beginning of the child's EntryList.

c) If multiple levels of inheritance exist, all RestrictionCriteria must evaluate to 'true'.

d) BaseContainers that form loops are illegal.

e) Abstract containers do not create concrete instances or new entities themselves.

f) Besides the basic rules above, various other elements and attributes form part of the inheritance process as shown in table 4-19.

**Table 4-19: Container Inheritance Rules**

| SequenceContainer Element or Attribute | Inheritance Rule |
|---|---|
| @name | Not inherited by child. |
| @abstract | Not inherited by child. |
| @idlePattern | Not inherited by child. |
| @shortDescription | Not inherited by child. |
| LongDescription | Not inherited by child. |
| AliasSet | Not inherited by child. |
| AncillaryDataSet | Child's content prefixed to parent's content if present. |

| SequenceContainer Element or Attribute | Inheritance Rule |
|---|---|
| DefaultRateInStream | Child's content will override parent's content if present; otherwise, child gets parent's content if it is specified. |
| RateInStreamSet | Child's content prefixed to parent's content if present, if RateInStream/@streamRef matches, the child's RateInStream will override. |
| BinaryEncoding/SizeInContainer | Child's content will override parent's content if present; otherwise, child gets parent's content if specified. |
| EntryList | Parent's content prefixed to child's, if present. |
| BaseContainer/RestrictionCriteria | Restrictions are scoped to their constructions. This means that a child's restrictions apply to both its EntryList and any inherited entries. |

#### 4.3.4.9.5.3 Inheritance Example

This simple example shows the basic concept of the inheritance mechanism using a 'MyPacket1' that requires ID==10.

```
<xtce:SequenceContainer name="AllMyPackets" abstract="true">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="ID"/>
        <xtce:ParameterRefEntry parameterRef="Length"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="MyPacket1" abstract="true">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="Volt"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="AllMyPackets">
        <xtce:RestrictionCriteria>
            <xtce:Comparison parameterRef="ID" value="10"/>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

This may also be represented in Unified Modelling Language (UML) as follows:

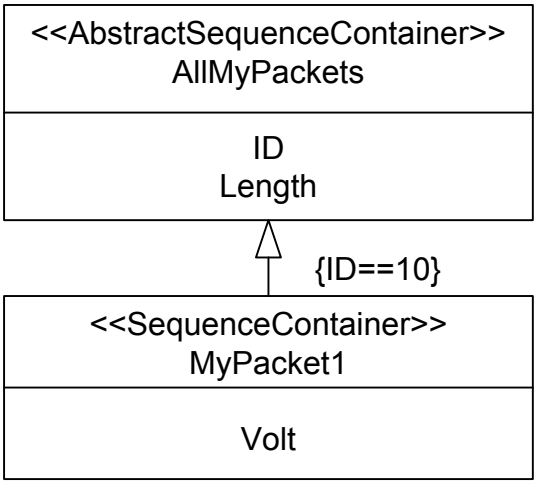



**Figure 4-91: UML Representation of Example Containers**

The parameters in RestrictionCriteria are parameter instances, and they are not required by the specification to refer to parameters in the container being inherited. The default instance is ‘zero’ (the most recent instance), and the default operator is ‘==’ (‘equals’).

The construction above builds one EntryList called the final EntryList. It constructs a packet with the three entries: ID, Length, and Volt, when ID is equal to 10.

A more detailed example is given in 5.4.2.3.

### 4.3.5 MESSAGESET

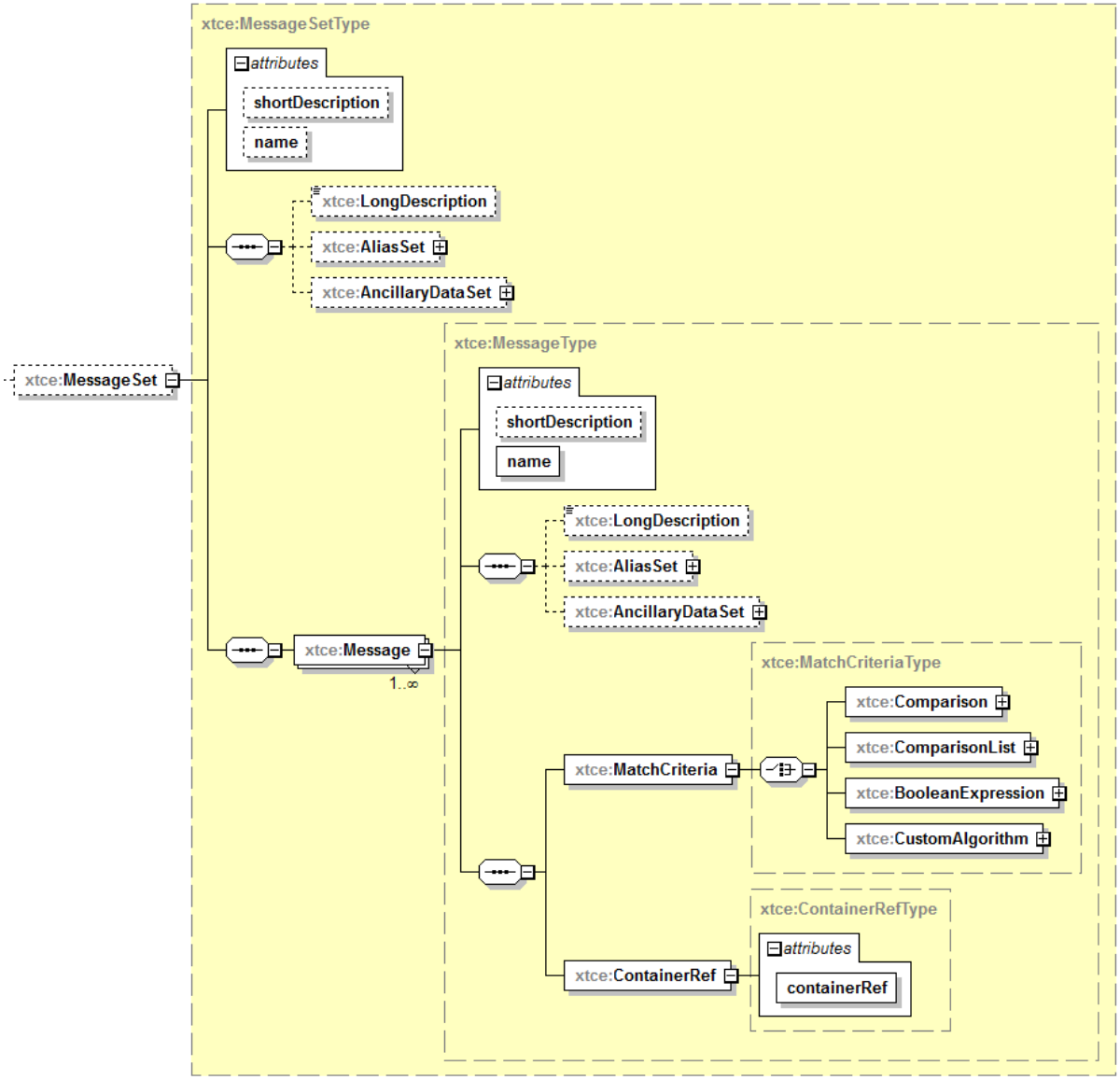



**Figure 4-92: MessageSet**

MessageSet/Messages are an association-based way of defining packets or minor frames.

The @name attribute associated with MessageSet is a typo and should be ignored.

The purpose of Message is to supply a mechanism to describe packets or minor frames without using inheritance. Message refers to a container, and that container is used to build up the final EntryList for a packet or minor frame.

MatchCriteria is used to identify the items of interest in a BaseContainer. The MatchCriteria supplies the identification areas for that particular packet or minor frame. It does not matter if the container is abstract or has a BaseContainer; those items should be ignored for Message.

To point to a container, the ContainRef and @containRef attributes are used.

### 4.3.6 STREAMSET

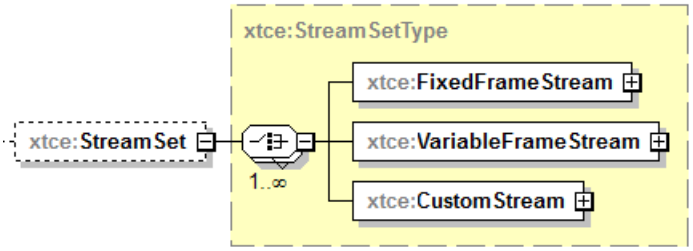



**Figure 4-93: StreamSet**

StreamSet allows definition of certain aspects of frames or bit streams (i.e., telemetry). It can also be used to define aspects of a 'sub-stream' within packets, such as video or voice stream.

Once the frame information is defined, the items in the frame can be referenced using the child elements ContainerRef or ServiceRef:

- If the ContainerRef is the root container of the container hierarchy (has derived child containers), then all its child containers are included.
- If ServiceRef is used, the stream refers to a defined service that itself contains a set of containers (see 4.5).

```
<xtce:StreamSet>
    <xtce:FixedFrameStream name="CCSDS" frameLengthInBits="8196" pcmType="NRZM">
        <xtce:ServiceRef serviceRef="CCSDSServices"/>
        <xtce:SyncStrategy>
            <xtce:SyncPattern pattern="1ACFFC1D" patternLengthInBits="8"/>
        </xtce:SyncStrategy>
    </xtce:FixedFrameStream>
</xtce:StreamSet>
```

### 4.3.7 ALGORITHMSET

#### 4.3.7.1 General

AlgorithmSet may contain aspects of algorithm descriptions used within the telemetry process system.

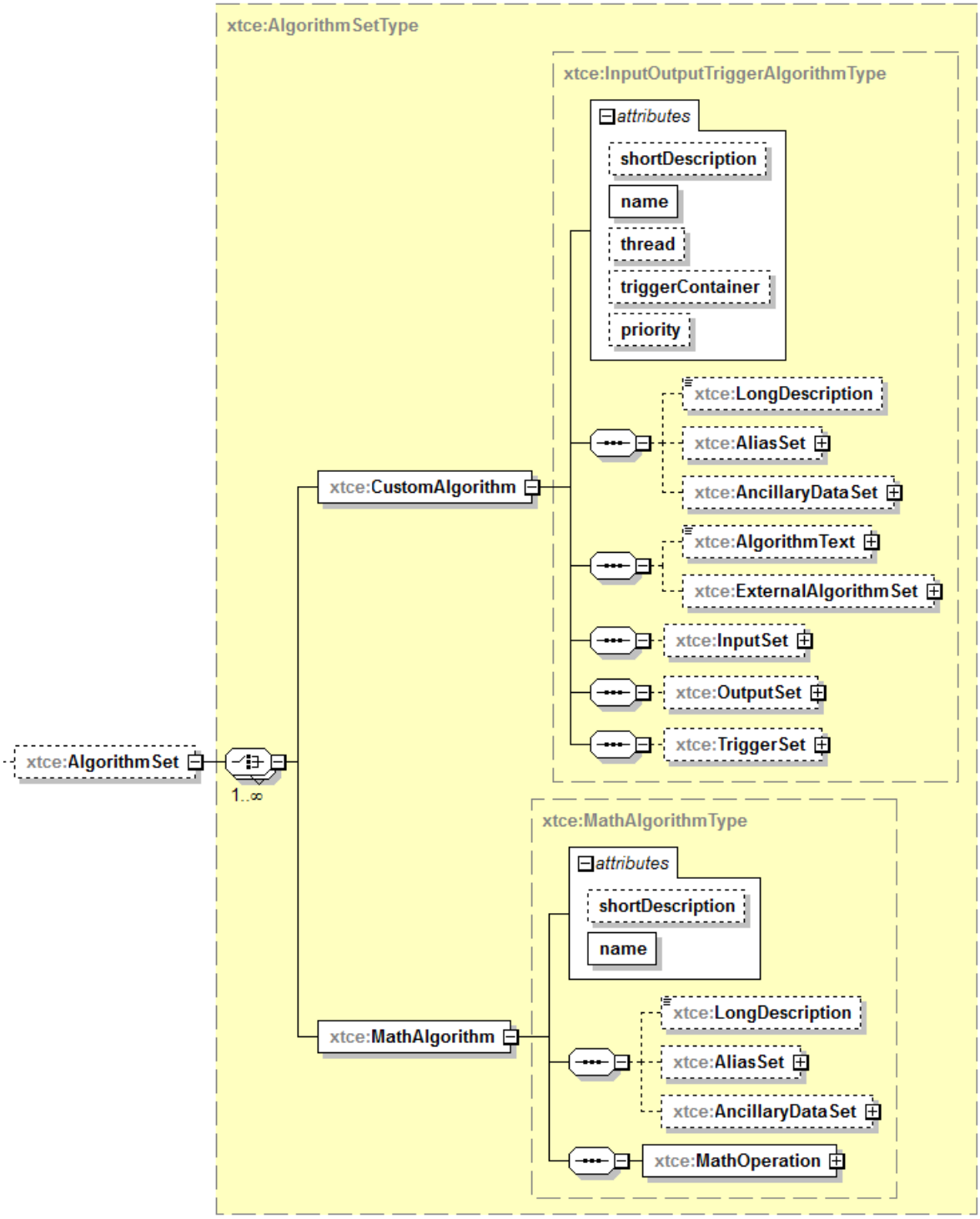



**Figure 4-94: AlgorithmSet Element**

### 4.3.7.2 CustomAlgorithm Element

(See 3.4.3.5.)

#### 4.3.7.3 MathAlgorithm Element

The MathAlgorithm element describes formulas using a postfix notation (‘reverse Polish notation’, i.e., 2 2 +). Prefix (i.e., 2 + 2) notation can be converted to postfix by using the well-known Shunting Yard Algorithm. The various available operands and operations are documented in the annotation of the element. (For an example, see 5.4.4.)

## 4.4 COMMANDMETADATA—COMMANDING

### 4.4.1 GENERAL

The CommandMetaData element captures command and command packet descriptions and other properties associated with commanding.

CommandMetaData shares many elements with TelemetryMetaData, as can be seen in figure 4-95. This subsection focuses on the differences between CommandMetaData and TelemetryMetaData. This subsection will not repeat explanations from the telemetry subsection.

Elements that are only for CommandMetaData are ArgumentTypeSet and MetaCommandSet. CommandContainerSet is derived from SequenceContainers and is identical in construction to TelemetryMetaData/ContainerSet.

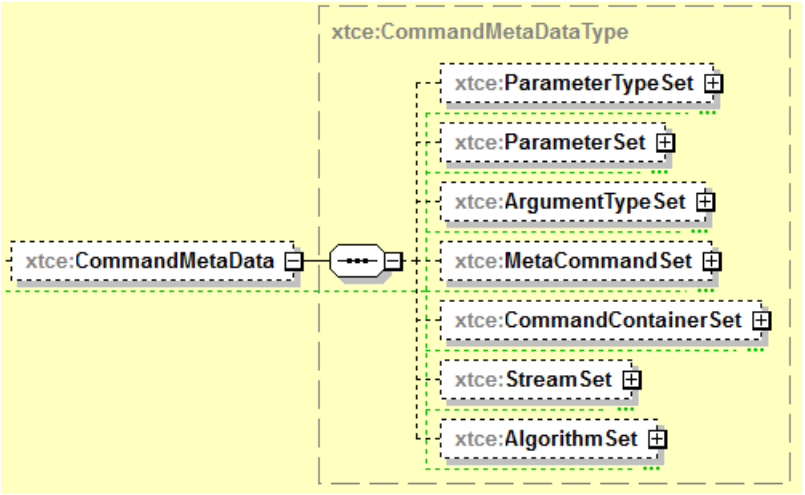



**Figure 4-95: CommandMetaData Element**

Parameters and ParameterTypes are constructed in the same manner as their TelemetryMetaData counterparts. They describe items supplied by the ground side to the fight destination; hence they are interepreted in a “reverse” send from the telemetry cousins. The remaining StreamSet and AlgorithmSet elements are also identical to their TelemetryMetaData counterparts. Again, these are probably best considered in ground to spacecraft direction.

MetaCommandSet contains command description. The term "MetaCommand" is used in XTCE for this because it contains more than just the bit values for a command. Hence MetaCommands are then the individual commands descriptions.

Most organization use the term 'arguments' to mean the operator-supplied values to the command. XTCE commands have arguments which will be discussed below.

However, at the top level of CommandMetaData is ArgumentTypeSe. This area is similar to ParameterTypeSet and is used in similar ways to parameters by the command argument. More about that will be discussed below.

So arguments in XTCE are something an operator provides to a command (MetaCommand), but a command in XTCE may also have parameters.

In XTCE, the dividing line on when to use which one is somewhat subjective, but the parameter is really meant to convey a sense that this part of the command is not something an operator would typically see or adjust but is rather supplied by the system (or through RestrictionCriteria).

Once again, though this may vary somewhat by organizations who may consider all their command's construction to be visible to the operator and in some sense set by them, or the operator is not a person necessarily, but software or a system. And again, the visibility of the exact construction of the command may be entirely in these systems' hands. Even so, XTCE divides the work of command description in this area into arguments and parameters, and the end user of XTCE should consider this when building their command descriptions.

MetaCommands have their own local CommandContainer (MetaCommand/CommandContainer) to describe the actual data block (packet/minor frame) associated with a command. The EntryList is similar the other Containers but has a special entry for arguments, among other differencs. These are discussed in more detail below.

CommandMetaData allows both Container and MetaCommand inheritance which also discussed in more details in the next sections.

### 4.4.2 COMMAND PARAMETERTYPESET—PARAMETERTYPES

#### 4.4.2.1 General

The CommandMetaData ParameterTypeSet holds command ParameterTypes. It makes use of the same schema types as TelemetryMetaData/ParameterTypeSet.

| | | | | |
|---|---|---|---|---|
| source | (optional) | (optional) | destination | (optional) |

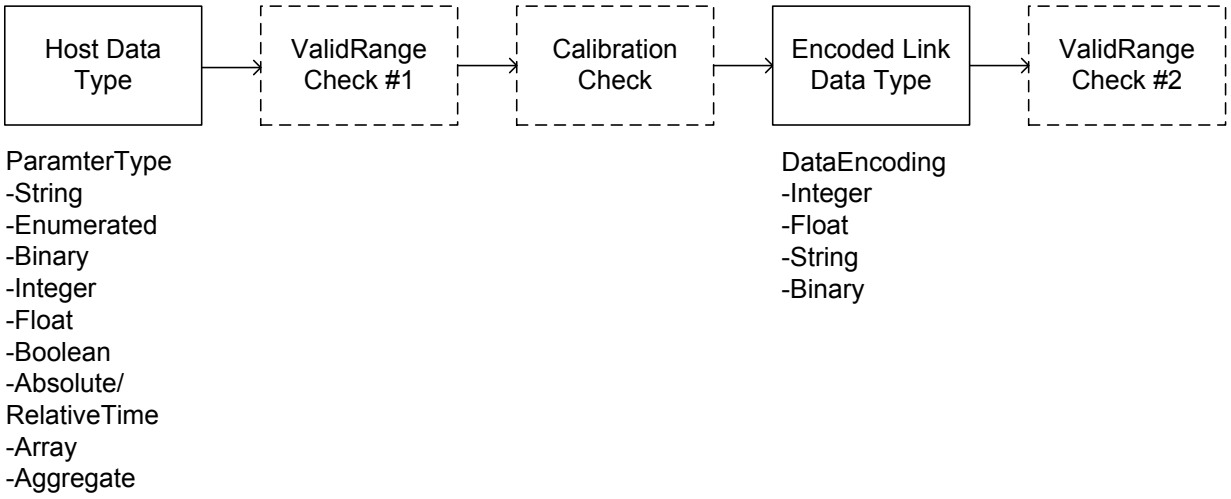



**Figure 4-96: Relationship of CommandMetaData ParameterType Elements (and ArgumentType)**

- ValidRange checks may be applied before calibration (units to raw) to the encoding data type or applied after reverse calibration.
- Command ParameterTypes can have alarm elements.
- Command ParameterType calibrators take values that are already in their proper units and convert them to an encoding for uplink.

#### 4.4.2.2 Command ParameterType and Encoding Table

Tables 4-5 and 4-15 (4.3.2.5) are applicable to Command ParameterTypes; however, the 'source' and 'destination' are swapped: ParameterType represents the source data type, and DataEncoding represents the destination data type.

#### 4.4.2.3 Alarms

The Alarm element is reused from TelemetryMetaData.

### 4.4.3 COMMAND PARAMETERSET—PARAMETERS

Command ParameterSet holds command parameters. The same rules apply for command parameters as telemetry. (See 4.3.2.5.4 for more information.)

Parameters included in a CommandContainerSet/CommandContainer EntryList or a MetaCommand/CommandContainer EntryList are used to construct a command packet.

### 4.4.4 ARGUMENTTYPESET—ARGUMENTTYPES

#### 4.4.4.1 General

ArgumentTypeSet holds ArgumentTypes. Legal ArgumentTypes follow the same rules as both command ParameterTypes and telemetry ParameterTypes. (See 4.4.2 and 4.3.2.3.7.5 for more information.) There are other differences; there are no alarms, for example. In XTCE 1.2, the overall construction of the schema types has diverged from the ParameterTypes as well.

NOTE – Within ArgumentType, some child elements and attributes use the term ‘parameterRef’ or ‘argumentRef’.

#### 4.4.4.2 ArgumentType Inheritance

ArgumentType inheritance behaves in a similar way to ParameterType inheritance (see 4.3.2.2.3). However, there are two minor differences to consider:

– ArgumentTypes do not have alarms.

– FloatArgumentType and IntegerArgument have a ValidRangeSet element. Child items are to be prefixed to any parent content, or the child gets the parent’s content if it is not specified.

#### 4.4.4.3 ArgumentType Tables

The tables in 4.3.2.5 are applicable to command ArgumentTypes; however, the source and destination are swapped: ArgumentType is the source data type, and DataEncoding is the destination data type.

### 4.4.5 METACOMMANDSET—METACOMMANDS, COMMAND DESCRIPTIONS

#### 4.4.5.1 General

MetaCommand is part of the MetaCommandSet.

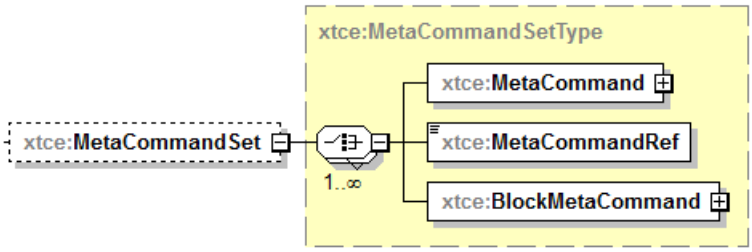



**Figure 4-97: MetaCommandSet Elements**

It is used to describe commands. It has its own local CommandContainer that is different from but similar in name to the CommandContainerSet. The MetaCommand/CommandContainer should be used to describe the bulk of commands related to packets or minor frames.

MetaCommand is a large element, so its discussion is split into two parts.

#### 4.4.5.2 MetaCommand -- Part 1

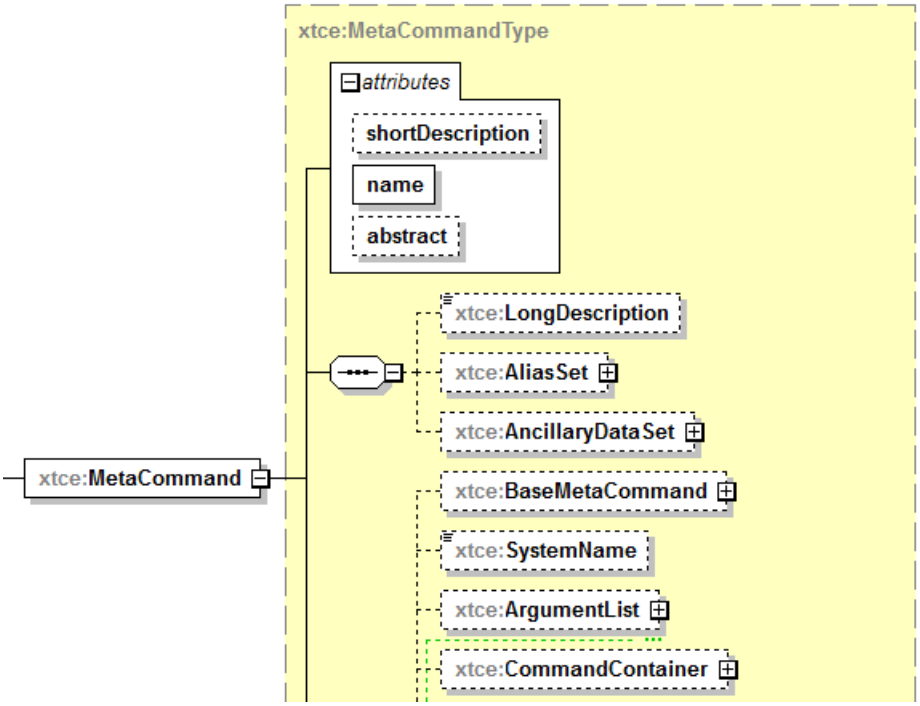



**Figure 4-98: The 1st Part of MetaCommand**

##### 4.4.5.2.1 NameDescription

Subsection 3.4.2 contains a description of the elements and attributes associated with name, shortDescription, LongDescription, AliasSet, and AncillaryDataSet.

##### 4.4.5.2.2 abstract Attribute

A MetaCommand may be set to ‘abstract’. This is interpreted to mean that MetaCommand is not itself a concrete command, but that it is used to build other commands. If the MetaCommand abstract flag is ‘true’, its CommandContainer is assumed to be abstract also.

#### 4.4.5.2.3 BaseMetaCommand Element

##### 4.4.5.2.3.1 General

One MetaCommand may extend another in an object-oriented manner (see 4.4.5.2.3.4).

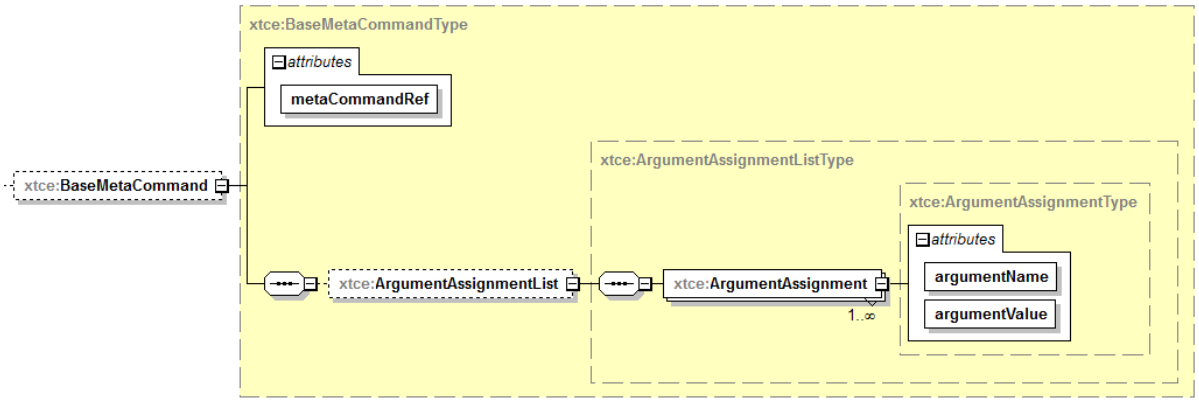



**Figure 4-99: BaseMetaCommand**

##### 4.4.5.2.3.2 metaCommandRef Attribute

The metaCommandRef attribute extends the MetaCommand given as a NameReference.

##### 4.4.5.2.3.3 ArgumentAssignmentList Element

ArgumentAssignmentList elements are named arguments whose values are set during the inheritance process. The arguments specified should be in the inheritance chain to be valid.

```
<xtce:MetaCommandSet>
    <xtce:MetaCommand name="Power">
        <xtce:ArgumentList>
            <xtce:Argument argumentTypeRef="PowerStateType" name="PowerState"/>
        </xtce:ArgumentList>
    </xtce:MetaCommand>
    <xtce:MetaCommand name="PowerON">
        <xtce:BaseMetaCommand metaCommandRef="Power">
            <xtce:ArgumentAssignmentList>
                <xtce:ArgumentAssignment argumentName="PowerState"
                argumentValue="ON"/>
            </xtce:ArgumentAssignmentList>
        </xtce:BaseMetaCommand>
    </xtce:MetaCommand>
</xtce:MetaCommandSet>
```

#### 4.4.5.2.3.4 MetaCommand Inheritance

MetaCommand inheritance is similar in concept to container inheritance (see 4.3.4.9.5), and the following rules apply:

a) A MetaCommand may extend another using the BaseMetaCommand element.

b) BaseMetaCommands that form loops are illegal.

c) Its CommandContainer is only inherited if the BaseContainer is explicitly set between the child and parent. The same rules apply to MetaCommand/CommandContainer inheritance as described in 4.3.4.9.

Besides the basic rules above, the following elements and attributes may form part of the inheritance process as shown in table 4-20.

**Table 4-20: MetaCommand Inheritance Rules**

| MetaCommand Element or Attribute | Inheritance Rule |
|---|---|
| @name | Not inherited by child. |
| @abstract | Not inherited by child. |
| @shortDescription | Not inherited by child. |
| LongDescription | Not inherited by child. |
| AliasSet | Not inherited by child. |
| AncillaryDataSet | Child's content prefixed to parent's content if present. |
| BaseMetaCommand | Not applicable. |
| BaseMetaCommand/ArgumentAssignment | Child's content will override parent's content if present; otherwise, child gets parent's content if it is specified. If argument is the same name, it overrides the parent's ArgumentAssignment. |
| SystemName | Child's content will override parent's content if present; otherwise, child gets parent's content if specified. |
| ArgumentList | Child's content prefixed to parent's content if present. |
| CommandContainer | Special Case: inherited like other containers if CommandContainer/BaseContainer set. Otherwise, it is not inherited. |
| TransmissionConstraintList | Child's content prefixed to parent's content if present. |
| DefaultSignificance | Child's content will override parent's content if present; otherwise, child gets parent's content if specified. |
| ContextSignificanceList | Child's content prefixed to parent's content if present. |
| Interlock | Child's content will override parent's content if present; otherwise, child gets parent's content if specified. |

| MetaCommand Element or Attribute | Inheritance Rule |
|---|---|
| VerifierSet | Child's content prefixed to parent's content if present but:<br>– Same verifiers are overridden by the child.<br>– CommandCompletes are accrued (child elements prefixed to parent's).<br>– If the child's CommandComplete has the same @name as parent's, the child's overrides it. |
| ParameterToSetList | Child's content prefixed to parent's content if present. If the @parameterRef is the same, the child's overrides the parent's. |
| ParameterToSuspendAlarmsOnSet | Child's content prefixed to parent's content if present. If the @parameterRef is the same, the child's overrides the parent's. |

#### 4.4.5.2.4 MetaCommand/CommandContainer—Command Packet

##### 4.4.5.2.4.1 General

MetaCommand/CommandContainer is an inner container area similar to the other container elements. It includes ArgumentRefEntry, ArrayArgumentRefEntry, and FixedValueEntry. It is 'invisible' to the CommandContainerSet and ContainerSet. It is a ContainerRefEntry in the EntryList.

From within a MetaCommand/CommandContainer, a ContainerRefEntry to either a CommandContainerSet or ContainerSet is legal. A MetaCommand/CommandContainer can extend another MetaCommand/CommandContainer, CommandContainerSet, or ContainerSet container. When a MetaCommand extends another MetaCommand, it is required by the XTCE syntax to specify MetaCommand/CommandContainer/BaseContainer to the other MetaCommand/CommandContainer. The example below shows this concept.

```
<xtce:MetaCommandSet>
    <xtce:MetaCommand name="BaseCmd">
        <xtce:CommandContainer name="BasePacket">
            <xtce:EntryList/>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
    <xtce:MetaCommand name="DerivedCmd">
        <xtce:BaseMetaCommand metaCommandRef="BaseCmd"/>
        <xtce:CommandContainer name="DerivedPacket">
            <xtce:EntryList/>
            <xtce:BaseContainer containerRef="BasePacket"/>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
</xtce:MetaCommandSet>
```

The BaseContainer/RestrictionCriteria element is optional in CommandContainer.

The EntryList for CommandContainer shown in figure 4-100 is slightly different from the other containers. (See 4.3.4.)

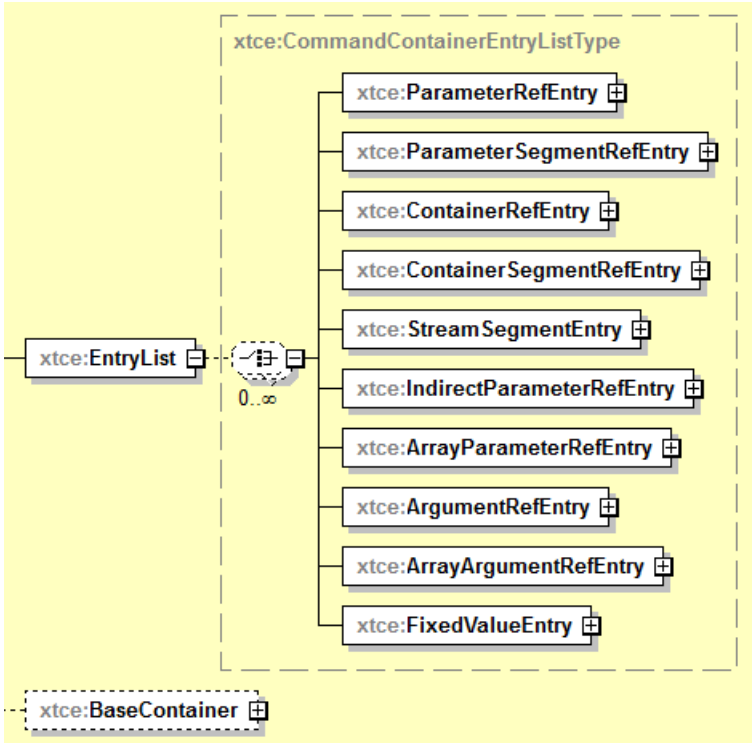



**Figure 4-100: MetaCommand/CommandContainer EntryList**

#### 4.4.5.2.4.2 ParameterRefEntry Element

ParameterRefEntry is the NameReference to a command parameter description or telemetry parameter description as is appropriate. If the reference is to a telemetry parameter, its description is used as if it were defined on the command side, meaning that it would be viewed as a ground side to spacecraft side.

#### 4.4.5.2.4.3 ContainerRefEntry Element

ContainerRefEntry is the NameReference to a CommandContainerSet container or ContainerSet SequenceContainer as is appropriate. If the reference is to a telemetry container, its description is used as if it were defined on the command side. It is probably bad form to refer to another MetaCommand/CommandContainer here.

#### 4.4.5.2.4.4 ArgumentRefEntry Element

ArgumentRefEntry is the NameReference to an Argument. In the final command construction (including inheritance), all the ArgumentRefEntries in the final EntryList construction must be matched to ArgumentList items.

#### 4.4.5.2.4.5 FixedValueEntry Element

The FixedValueEntry element specifies a literal fixed value that specifies with a specific sizeInBits. For XTCE 1.2, an optional name may be added.

An example follows:

```
<xtce:EntryList>
    <xtce:FixedValueEntry binaryValue="5A" sizeInBits="7"/>
</xtce:EntryList>
```

#### 4.4.5.2.4.6 BaseContainer

BaseContainer is used to extend another MetaCommand/CommandContainer. In some cases, another SequenceContainer or CommandContainerSet/CommandContainer may be appropriate.

The containerRef to another MetaCommand/CommandContainer must be present if the MetaCommand is extending another MetaCommand to include the parent's MetaCommand/CommandContainer/EntryList. It should not be included otherwise. (See 4.3.4.9 on Container Inheritance.)

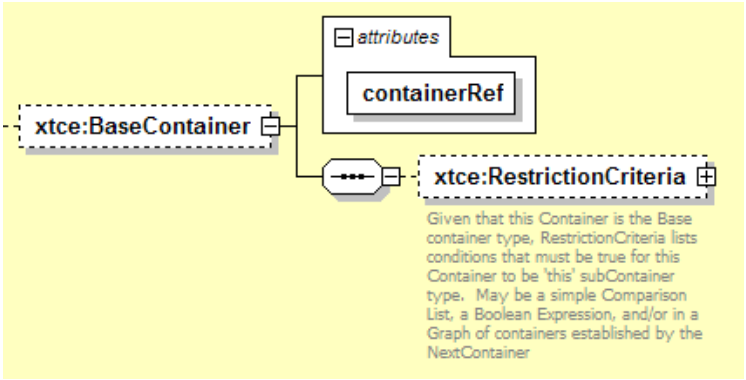



**Figure 4-101: The Base Element from CommandContainer**

Specifying a NameReference to either CommandMetaData/CommandContainerSet containers or TelemetryMetaData/ContainerSet is legal.

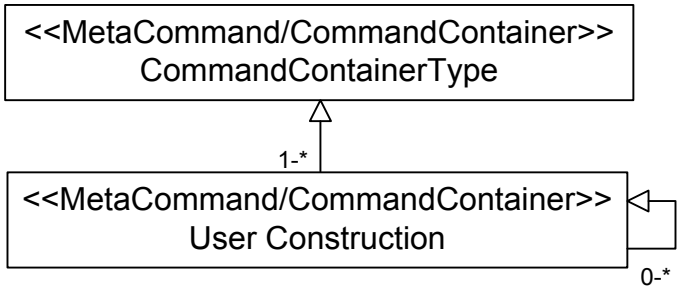



**Figure 4-102: Extending MetaCommand/CommandContainers**

MetaCommand/CommandContainers are very similar to SequenceContainers. It is legal to specify either a CommandContainerSet/CommandContainer or a TelemetryMetaData/ContainerSet/SequenceContainer in the BaseContainer. This is typically used for headers or other shared items.

#### 4.4.5.2.4.7 containerRef Attribute

The containerRef attribute is a NameReference to another MetaCommand/CommandContainer. If one MetaCommand is extending another one, then the NameReference to the parent's MetaCommand/CommandContainer must be included here to get the parent's EntryList.

#### 4.4.5.2.4.8 RestrictionCriteria Element

In CommandMetaData, RestrictionCriteria are assertions. The system processes the expressions to determine which values will make the expressions true. Those values are the values that go with the named parameters in the RestrictionCriteria.

This element is optional in MetaCommand/CommandContainer because many commands may make use of FixedValue to supply the values of interest instead of using parameters and assertions.

The RestrictionCriteria is a MatchCriteriaType. (See 3.4.3.6 for MatchCriteria information.)

#### 4.4.5.2.4.9 Rules for MetaCommand/CommandContainer Inheritance

CommandContainer inheritance follows the same rules as other container inheritance. (See 4.3.4.9 for details.)

#### 4.4.5.2.4.10 ArgumentList Element

Arguments to a specific command are listed in its ArgumentList. The argument references the ArgumentType that is appropriate and matches ArgumentRefs in its CommandContainer/EntryList.

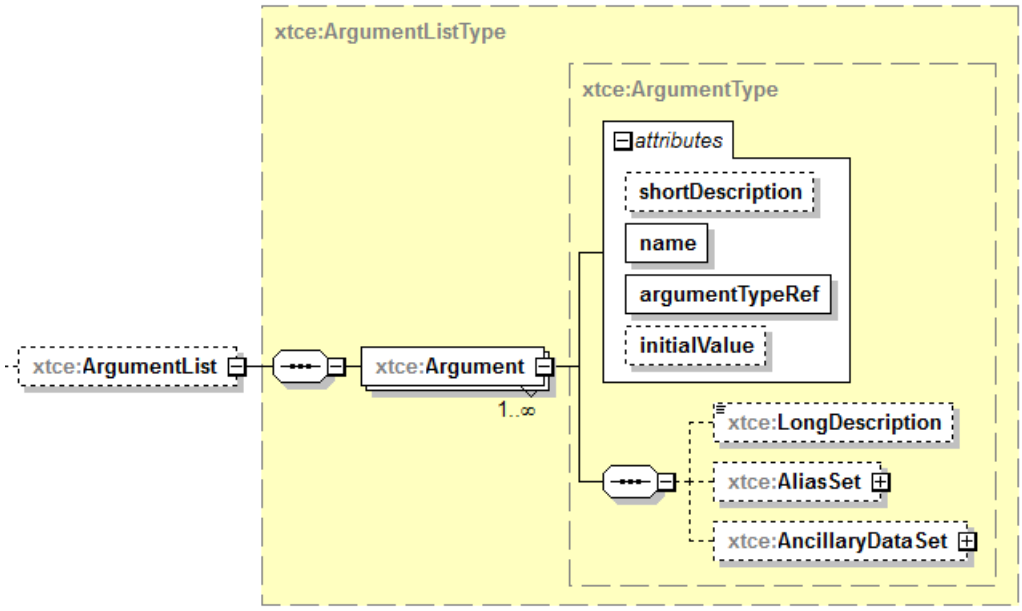



**Figure 4-103: Argument Element**

Arguments may also be in an extended MetaCommand, as the following example shows:

```
<xtce:CommandMetaData>
    <xtce:ArgumentTypeSet>
        <xtce:IntegerArgumentType name="UINT32">
            <xtce:UnitSet/>
            <xtce:IntegerDataEncoding/>
        </xtce:IntegerArgumentType>
    </xtce:ArgumentTypeSet>
    <xtce:MetaCommandSet>
        <xtce:MetaCommand name="Cmd">
            <xtce:ArgumentList>
                <xtce:Argument argumentTypeRef="UINT32" name="MyArg"/>
            </xtce:ArgumentList>
            <xtce:CommandContainer name="CmdPacket">
                <xtce:EntryList>
                    <xtce:ArgumentRefEntry argumentRef="MyArg"/>
                </xtce:EntryList>
            </xtce:CommandContainer>
        </xtce:MetaCommand>
    </xtce:MetaCommandSet>
</xtce:CommandMetaData>
```

#### 4.4.5.3 MetaCommand -- Part 2, Additional Elements

##### 4.4.5.3.1 General

MetaCommand contains a number of additional elements unique to commanding that are used to describe behaviors before or after sending a command. These elements are as follows:

a) TransmissionConstraintList checks that the command may be sent in the current operating mode.

b) DefaultSignificance is used to define a default significance level that must be confirmed before the command is sent.

c) ContextSignifanceLevel specifies significance levels by context.

d) Interlock blocks the next command until this one has reached a certain stage.

e) VerifierSet checks telemetry against the specified command phase.

f) ParameterToSetList captures parameters to set after the command has been set, such as a command counter.

g) ParametersToSuspendAlarms suspends the alarms checks on the telemetry specified for a certain period of time until the command has taken effect.

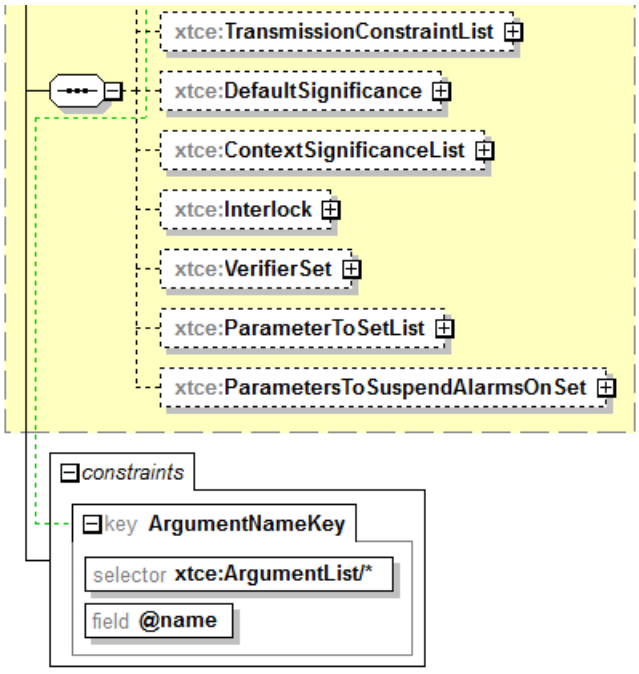



**Figure 4-104: The 2nd Part of MetaCommand**

#### 4.4.5.3.2 TransmissionConstraintList Element

TransmissionConstraintList checks that a command may be sent in the current operating mode. The 'mode' is defined as a MatchCriteria condition (see 3.4.3.6). For example, a mission phase could be defined as an enumerated session variable and used in this location.

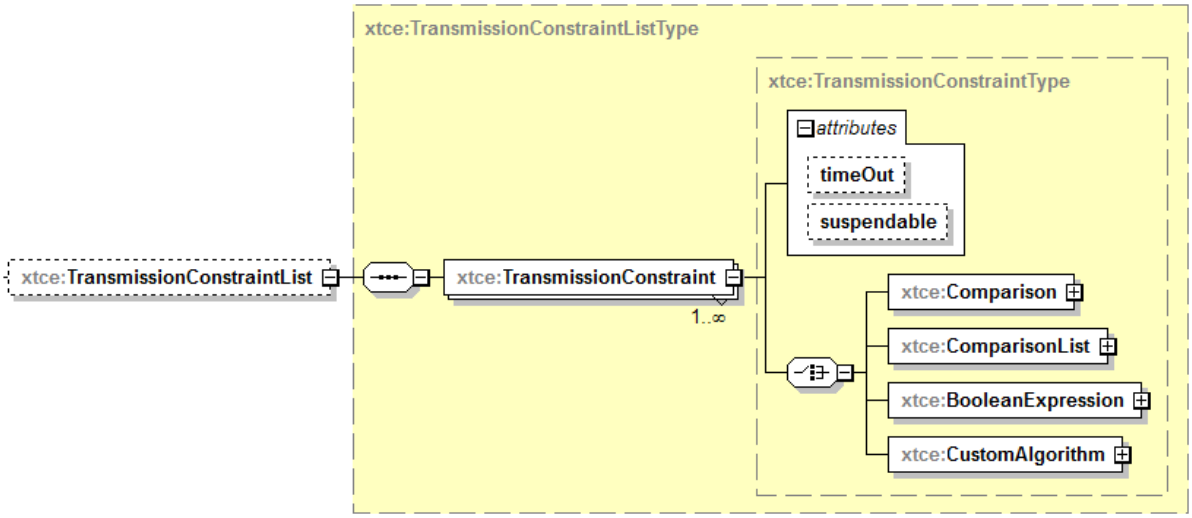



**Figure 4-105: TransmissionConstraintList Element**

The timeOut attribute specifies the relative time the constraint is valid. If it is marked as 'suspendable', then the constraint may be suspended (default value is 'false'). In the example below, the command cannot be sent until operating orbit is achieved.

```
<xtce:MetaCommand name="MyCmd">
    <xtce:TransmissionConstraintList>
        <xtce:TransmissionConstraint>
            <xtce:Comparison parameterRef="MissionPhase" value="OperatingOrbit"/>
        </xtce:TransmissionConstraint>
    </xtce:TransmissionConstraintList>
</xtce:MetaCommand>
```

#### 4.4.5.3.3 DefaultSignificance Element

##### 4.4.5.3.3.1 General

DefaultSignificance is used to define a default significance level that must be confirmed before the command is sent.

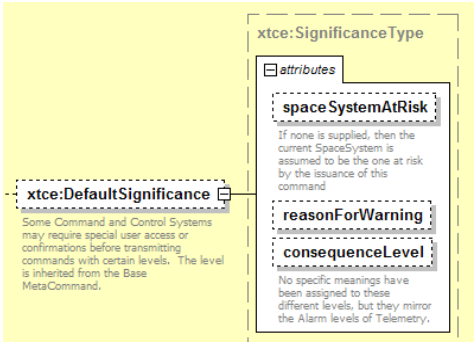



**Figure 4-106: DefaultSignificance Element**

#### 4.4.5.3.3.2 spaceSystemAtRisk Attribute

The spaceSystemAtRisk attribute gives the NameReference of the SpaceSystem within the document that will be affected by this command.

#### 4.4.5.3.3.3 reasonForWarning Attribute

The reasonForWarning attribute is a user-defined string that is appropriate for the issue.

#### 4.4.5.3.3.4 consequenceLevel Attribute

The consequenceLevel attribute is an enumerated list of levels that includes none, watch, warning, distress, critical, and severe. For example:

```
<xtce:MetaCommand name="TestOverPressureCmd">
    <xtce:DefaultSignificance consequenceLevel="watch" spaceSystemAtRisk="CrewModule"
    reasonForWarning="Temporary valve overpressure may occur"/>
</xtce:MetaCommand>
```

### 4.4.5.3.4 ContextSignificanceList Element

The ContextSignificanceList element specifies significances with contexts. The context is a MatchCriteria (see 3.4.3.6) and is user defined.

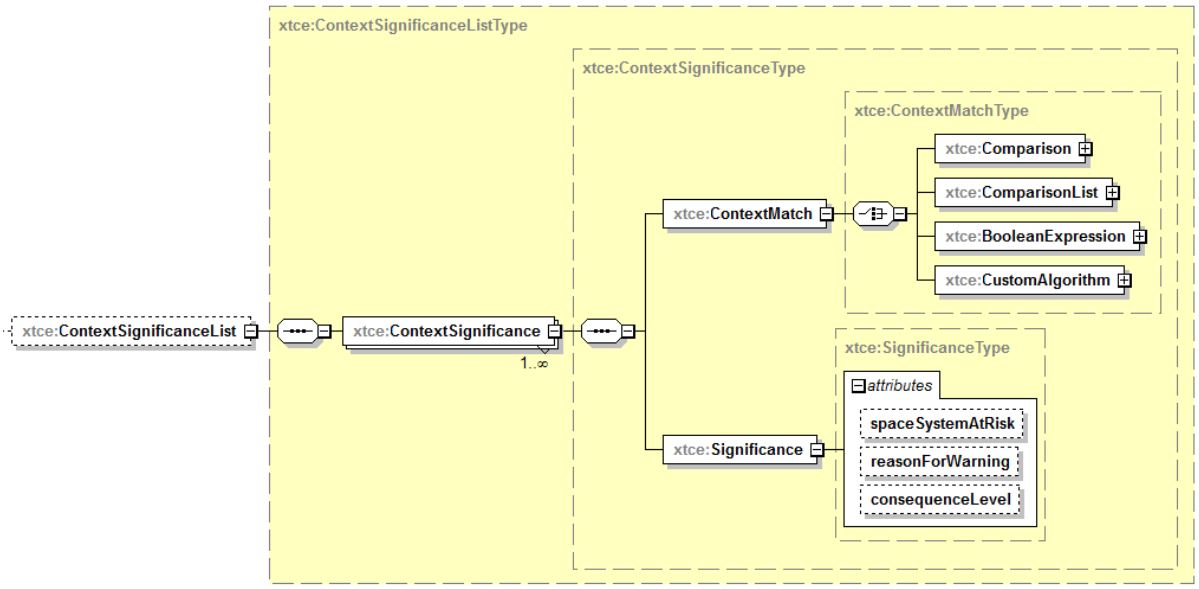



**Figure 4-107: ContextSignificance Element**

### 4.4.5.3.5 Interlock Element

#### 4.4.5.3.5.1 General

The Interlock element prevents the next command from being sent until the current one reaches some verification state.

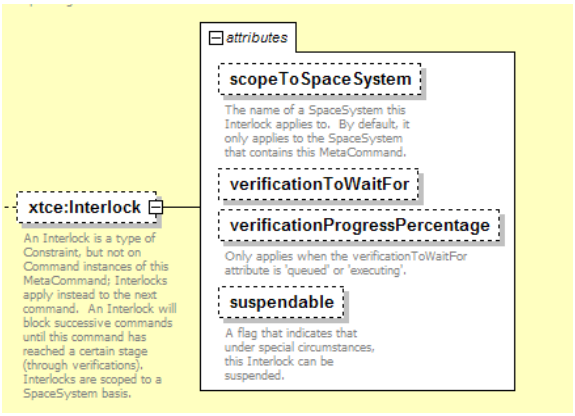



**Figure 4-108: Interlock**

#### 4.4.5.3.5.2 scopeToSpaceSystem Attribute

The scopeToSpaceSystem attribute prevents the next command associated with the named SpaceSystem. It is given as a NameReference. If none is specified, this SpaceSystem is assumed.

#### 4.4.5.3.5.3 verificationToWaitFor Attribute

The verificationToWaitFor attribute specifies the level of verification (see VerificationSet 4.4.5.3.6) to complete before sending the command (default value is ‘complete’).

#### 4.4.5.3.5.4 verificationProgressPercentage Attribute

The verificationProgressPercentage attribute prevents the next command from being sent until this percentage of verification has completed.

#### 4.4.5.3.5.5 suspendable Attribute

The suspendable attribute is used to set the interlock as suspendable (default is ‘false’). In the example below, all commands are blocked until the decouple stage has finished. This element is tied to a verifier.

```
<xtce:MetaCommand name="FireDecouple">
    <xtce:Interlock scopeToSpaceSystem="EngineAssembly" verificationToWaitFor="complete"
    verificationProgressPercentage="100.0"/>
    <xtce:VerifierSet>
        <xtce:CompleteVerifier>
            <xtce:ComparisonList>
                <xtce:Comparison parameterRef="DecoupleTlmStatus" value="0xffff"/>
            </xtce:ComparisonList>
        <xtce:CheckWindow timeToStopChecking="PT100S"/>
        </xtce:CompleteVerifier>
    </xtce:VerifierSet>
</xtce:MetaCommand>
```

### 4.4.5.3.6 VerifierSet Element

#### 4.4.5.3.6.1 General

Various command verification stages may be specified using the VerifierSet element. More than one CompleteVerifier may be specified.


**Figure 4-109: Verifiers**

Each verifier follows a similar pattern. Two have slight additions: the ExecutionVerifier adds PercentageComplete; the CompleteVerifier adds a return for a ParameterRef that should be interpreted as the last recorded value for that Parameter.

Each verifier has the following elements:

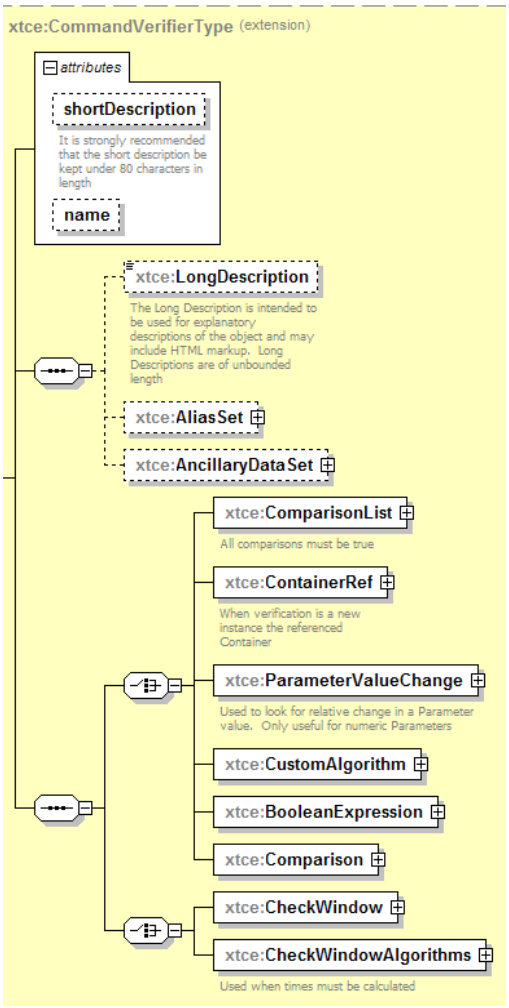



**Figure 4-110: CommandVerifiers**

#### 4.4.5.3.6.2 NameDescription

Subsection 3.4.2 contains a description of the elements and attributes associated with name, shortDescription, LongDescription, AliasSet, and AncillaryDataSet.

#### 4.4.5.3.6.3 ComparisonList Element

The verifier is a ComparisonList (see MatchCriteria 3.4.3.6).

#### 4.4.5.3.6.4 ContainerRef Element

ContainerRef indicates the stage of verification that has been reached when the referenced container appears in the input stream. For CCSDS missions, it would be the last container in a definition making up an entire CCSDS packet container hierarchy.

#### 4.4.5.3.6.5 ParameterValueChange Element

ParameterValueChange is used to compare the difference between the last recorded value of the specified parameter and the new one. The difference must be greater than or equal to the value stored in Change.

#### 4.4.5.3.6.6 CustomAlgorithm Element

CustomAlgorithm is used to specify a custom algorithm as the verifier.

#### 4.4.5.3.6.7 BooleanExpression Element

The BooleanExpression verifier is a more complex Boolean expression (see MatchCriteria 3.4.3.6).

#### 4.4.5.3.6.8 Comparison Element

The Comparison verifier is a single comparison (see MatchCriteria 3.4.3.6).

#### 4.4.5.3.6.9 CheckWindow Element

The CheckWindow verifier is tied to a time window. There are the following three attributes:

- timeToStartChecking: optional, xsd:duration (xtce:RelativeTimeType);
- timeToStopChecking: required, xsd:duration;
- timeWindowIsRelativeTo: optional, default: timeLastVerifierPassed, or commandReleased.

```
<xtce:CheckWindow timeToStartChecking="PT1S" timeToStopChecking="PT10S"
timeWindowIsRelativeTo="commandRelease"/>
```

In the example above, the CheckWindow verifier starts to check 1 second after the command is released and continues checking for 10 seconds.

#### 4.4.5.3.6.10 CheckWindowAlgorithm Element

The verifier is tied to a more complex algorithm-based time window.

```
<xtce:CompleteVerifier>
    <xtce:ParameterValueChange>
        <xtce:ParameterRef parameterRef="StrainGauge"/>
        <xtce:Change value="15"/>
    </xtce:ParameterValueChange>
    <xtce:CheckWindow timeToStopChecking="PT10S"/>
</xtce:CompleteVerifier>
```

#### 4.4.5.3.7 ParameterToSetList Element

The ParameterToSetList element captures parameters that will have their values adjusted once the setOnVerification command verification level has been reached for a MetaCommand. The default value for setOnVerification is ‘complete’. If one or more CompleteVerifiers is specified, then they should have evaluated to ‘true’; otherwise, the command is assumed to have completed successfully. The same would be true for any other specified setOnVerification value.

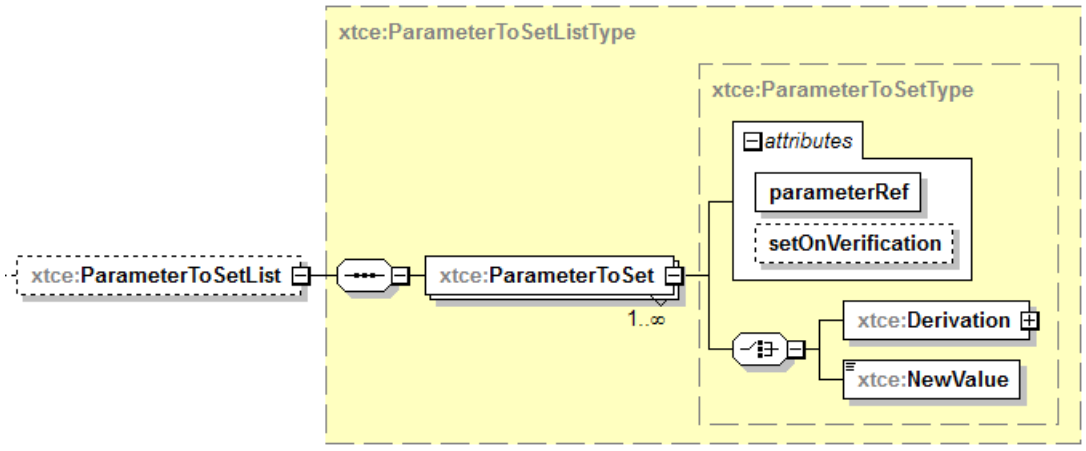



**Figure 4-111: ParameterToSet Element**

Once setOnVerification has been reached, either a fixed value from NewValue is set for the parameter in parameterRef, or a derived value is set using the MathOperations element in Derivation.

```
<xtce:ParameterToSetList>
    <xtce:ParameterToSet parameterRef="CommandCounter">
        <xtce:Derivation>
        <xtce:ParameterInstanceRefOperand parameterRef="CommandCounter"/>
        <xtce:ValueOperand>1</xtce:ValueOperand>
        <xtce:Operator>+</xtce:Operator>
        </xtce:Derivation>
    </xtce:ParameterToSet>
</xtce:ParameterToSetList>
```

The example above increments the command counter once the command is completed.

#### 4.4.5.3.8 ParametersToSuspendAlarmsOnSet Element

The ParametersToSuspendAlarmsOnSet element describes telemetry parameters whose alarms will be suspended for the given suspenseTime once the MetaCommand has reached a certain command verification level as specified in verifierToTriggerOn.

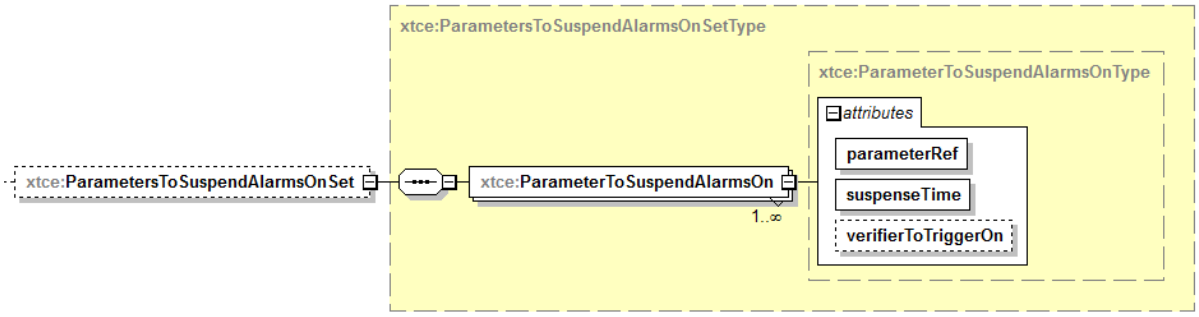



**Figure 4-112: ParametersToSuspendAlarmsOnSet Element**

The default is to start the suspension when the verification release level returns 'true'. If no verifiers are set, it would be assumed that the command released. In the example below, once the command is released, alarm checking on telemetry parameter 'P1LevelsThatAreNoisy' is suspended.

```
<xtce:ParametersToSuspendAlarmsOnSet>
    <xtce:ParameterToSuspendAlarmsOn parameterRef="P1LevelsThatAreNoisy"
    suspenseTime="PT30S" verifierToTriggerOn="release"/>
</xtce:ParametersToSuspendAlarmsOnSet>
```

### 4.4.5.4 MetaCommandSet – MetaCommandRef and BlockMetaCommand

#### 4.4.5.4.1 General

There are two more sibling elements of MetaCommand to consider: MetaCommandRef and BlockMetaCommand.

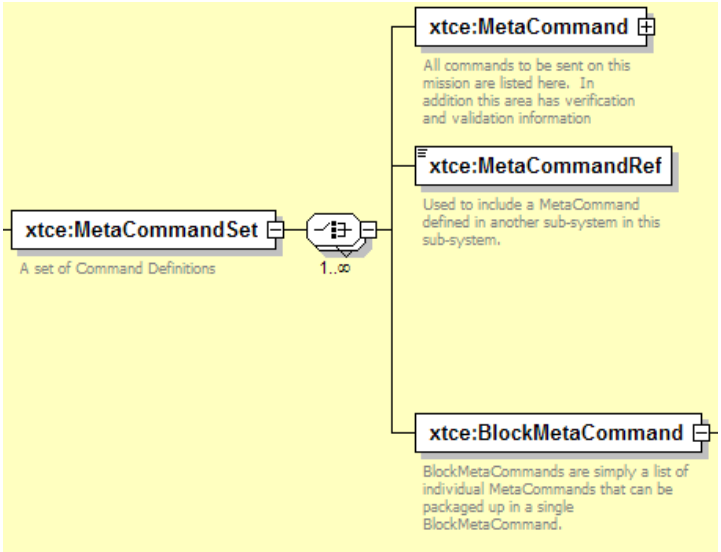



**Figure 4-113: MetaCommandSet Element**

#### 4.4.5.4.2 MetaCommandRef Element

MetaCommandRef allows for the importation of a MetaCommand defined in another SpaceSystem. The SpaceSystem containing the MetaCommandRef acts as if the MetaCommand being referenced is in the MetaCommandSet.

#### 4.4.5.4.3 BlockMetaCommand Element

BlockMetaCommand is used to capture the notion of command groupings.

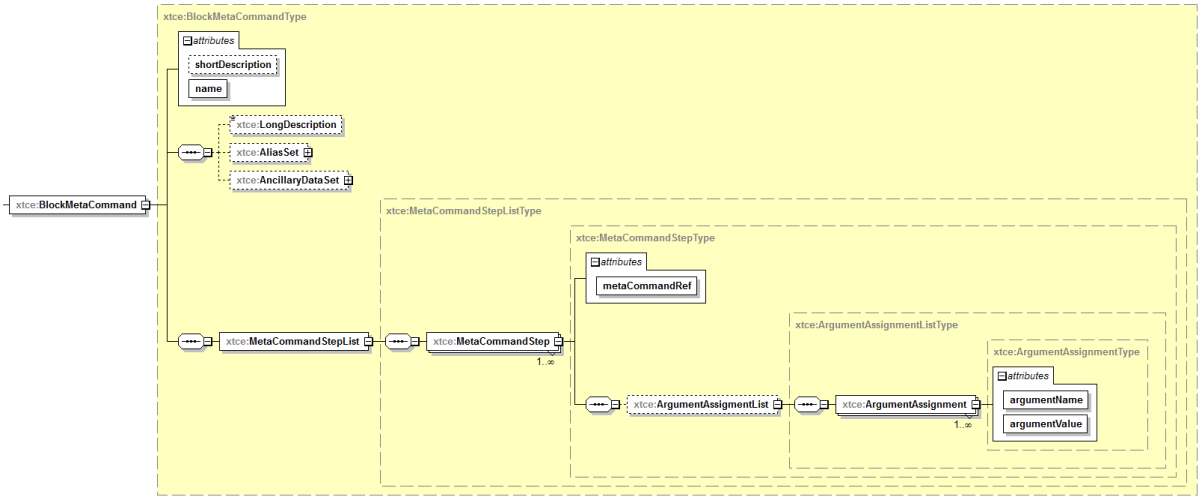



**Figure 4-114: BlockMetaCommand**

When specifying that a particular MetaCommand is part of a block, the argument list and the necessary argument values may be specified as well.

Often, block commands are sent as one unit. In the example below, 'NOOP' commands are known to take approximately 1ms to process. This block command then is constructed to open and close a valve after a 5ms delay time period.

```
<xtce:BlockMetaCommand name="TimedValveCmdFixed">
        <xtce:MetaCommandStepList>
            <xtce:MetaCommandStep metaCommandRef="ValveCmd">
                <xtce:ArgumentList>
                    <xtce:Argument value="ON" name="ValveState"/>
                </xtce:ArgumentList>
        </xtce:MetaCommandStep>
        <xtce:MetaCommandStep metaCommandRef="NOOP"/>
        <xtce:MetaCommandStep metaCommandRef="NOOP"/>
        <xtce:MetaCommandStep metaCommandRef="NOOP"/>
        <xtce:MetaCommandStep metaCommandRef="NOOP"/>
        <xtce:MetaCommandStep metaCommandRef="NOOP"/>
        <xtce:MetaCommandStep metaCommandRef="ValveCmd">
```

```
            <xtce:ArgumentList>
                <xtce:Argument value="OFF" name="ValveState"/>
            </xtce:ArgumentList>
        </xtce:MetaCommandStep>
    </xtce:MetaCommandStepList>
</xtce:BlockMetaCommand>
```

### 4.4.6 COMMANDCONTAINERSET — COMMANDCONTAINER

#### 4.4.6.1 General

CommandContainerSet holds CommandContainers. It is identical to the ContainerSet area within the TelemetryMetaData (see 4.3.4).

#### 4.4.6.2 CommandContainerSet Inheritance Rules

CommandContainerSet/CommandContainers may extend another CommandContainerSet/CommandContainer. This follows the same rules already established for container inheritance (see 4.3.4.9).

It is legal to use NameReference to extend either a TelemetryMetaData/ContainerSet container or a MetaCommand/CommandContainer Container.

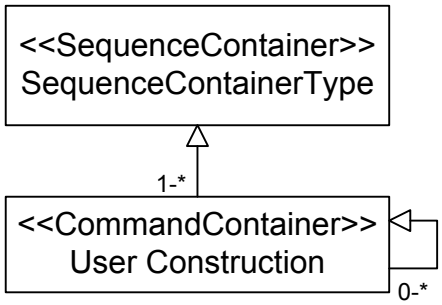



**Figure 4-115: Extending CommandContainers**

### 4.4.7 STREAMSET

(See 4.3.6.)

### 4.4.8 ALGORITHMSET

(See 4.3.7.)

## 4.5 SERVICESET ELEMENT — SERVICES

### 4.5.1 GENERAL

Services are logical groupings of containers or messages under a common name. The meaning of such groupings is implementation dependent.

When using containerRefs, the container being referred to may be an abstract container. If it is, all its child containers are included in the service. If this is not the case, then only that specific container is included, and any other containers it has up the inheritance chain are not.

This is not an issue when using MessageRefSet since messages ignore BaseContainer.

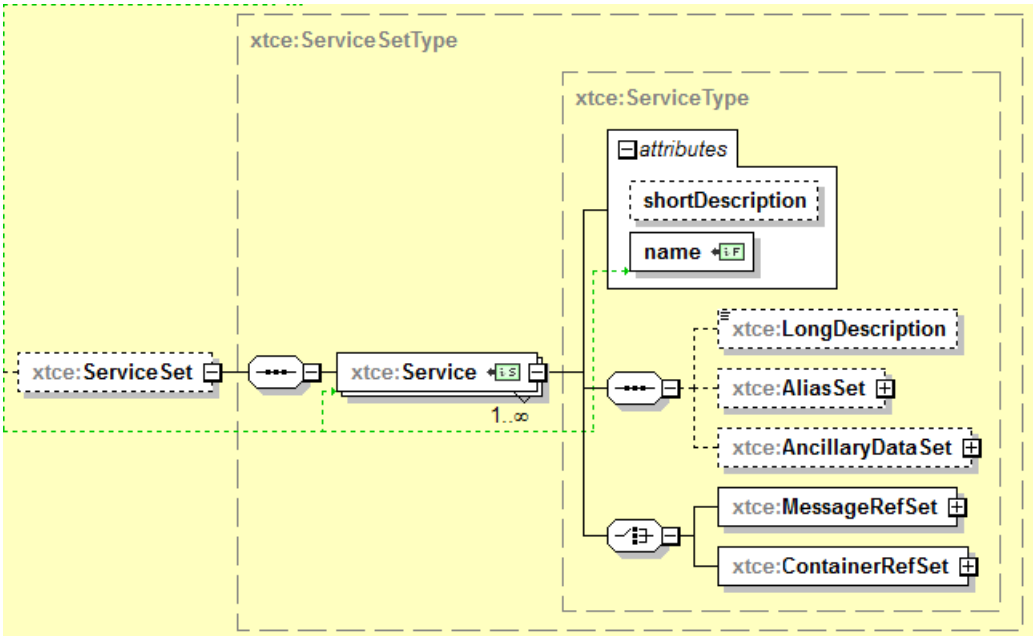



**Figure 4-116: Services**

How containers or messages are organized is left to the end user.

```
<xtce:ServiceSet>
    <xtce:Service name="Imagers">
        <xtce:ContainerRefSet>
            <xtce:ContainerRef containerRef="ImagerID"/>
            <xtce:ContainerRef containerRef="ResX"/>
            <xtce:ContainerRef containerRef="ResY"/>
            <xtce:ContainerRef containerRef="FOV"/>
            <xtce:ContainerRef containerRef="Segments"/>
            <xtce:ContainerRef containerRef="Status"/>
        </xtce:ContainerRefSet>
    </xtce:Service>
</xtce:ServiceSet>
```

### 4.5.2 MESSAGEREFSET ELEMENT

MessageRefSet refers to a message rather than the container that will then be included in the service.

### 4.5.3 CONTAINERREFSET ELEMENT

This element contains a list of containers that are part of this service. Referring to an abstract container that has concrete child containers will include all of them in the service. Individual representations of things like packets should refer to the leaf container (which is nonabstract and has a baseContainer). Leaf containers are covered in section 5.

# 5 ADVANCED TOPICS

## 5.1 OVERVIEW

This section describes various concepts in XTCE and uses examples to reinforce them.

## 5.2 ABSTRACT, CONCRETE, AND PLAIN CONTAINERS

### 5.2.1 GENERAL

Containers are defined in either the TelemetryMetaData/ContainerSet area, the CommandMetaData/CommandContainerSet area, or the MetaCommand/CommandContainer area.

A container can take three forms:

– abstract (sometimes called a generic container);

– plain; and

– concrete (sometimes called an instance container or container instances).

### 5.2.2 THE ABSTRACT CONTAINER

A container marked as abstract is often used to represent a more generic concept in the format being described.

Usually, abstract containers are used by other containers to build a more complete definition that represents packet or minor frames. Often they are at the root of the container hierarchy; however, they may be at each level of a multi-level hierarchy until the last container. For example, an abstract container could be used as a header in a packet description.

It is legal for an abstract container to extend another container.

```
<xtce:SequenceContainer name="AbstractHeader" abstract="true">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="Version"/>
        <xtce:ParameterRefEntry parameterRef="ID"/>
        <xtce:ParameterRefEntry parameterRef="Length"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
```

### 5.2.3 THE PLAIN CONTAINER

A plain container is not marked abstract and has no BaseContainer. Plain container definitions will be used in a ContainerRefEntry. A plain container is part of another construction and a complete entity such as a packet or minor frame. Plain containers are defined in either ContainerSet or CommandContainerSet.

A MetaCommand/CommandContainer is not a plain container; however, it is used to specify how to build a command packet. It may be an entity such as a packet or minor frame even if it has no BaseContainer.

```
<xtce:SequenceContainer name="Plain">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="P1"/>
        <xtce:ParameterRefEntry parameterRef="P2"/>
        <xtce:ParameterRefEntry parameterRef="P3"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
```

### 5.2.4 THE CONCRETE CONTAINER

A concrete container or instance container is not abstract and has a BaseContainer element. Concrete containers extend other containers (which are probably abstract), and they represent a complete entity such as a packet or minor frame.

```
<xtce:SequenceContainer name="MyConcretePacket">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="P1"/>
        <xtce:ParameterRefEntry parameterRef="P2"/>
        <xtce:ParameterRefEntry parameterRef="P3"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="AbstractHeader">
    <xtce:RestrictionCriteria>
        <xtce:ComparisonList>
            <xtce:Comparison parameterRef="Version" value="1"/>
            <xtce:Comparison parameterRef="ID" value="100"/>
        </xtce:ComparisonList>
    </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

## 5.3 MODIFYING AN ENTRY

### 5.3.1 GENERAL

Each entry has several child elements that can be modified in a variety of ways:

a) LocationInContainerInBits can change the entry’s address.

b) RepeatEntry can cause the entry to repeat more than one time.

c) IncludeCondition can cause the entry to be left out or included in the EntryList based on the evaluation of a set of expressions.

### 5.3.2 ADDRESSING USING LOCATIONINCONTAINERINBITS

#### 5.3.2.1 General

Subsection 4.3.4.8.10 contains a discussion of the element LocationInContainerBits, its attributes, and child elements. The following diagram graphically shows the details of container addressing.

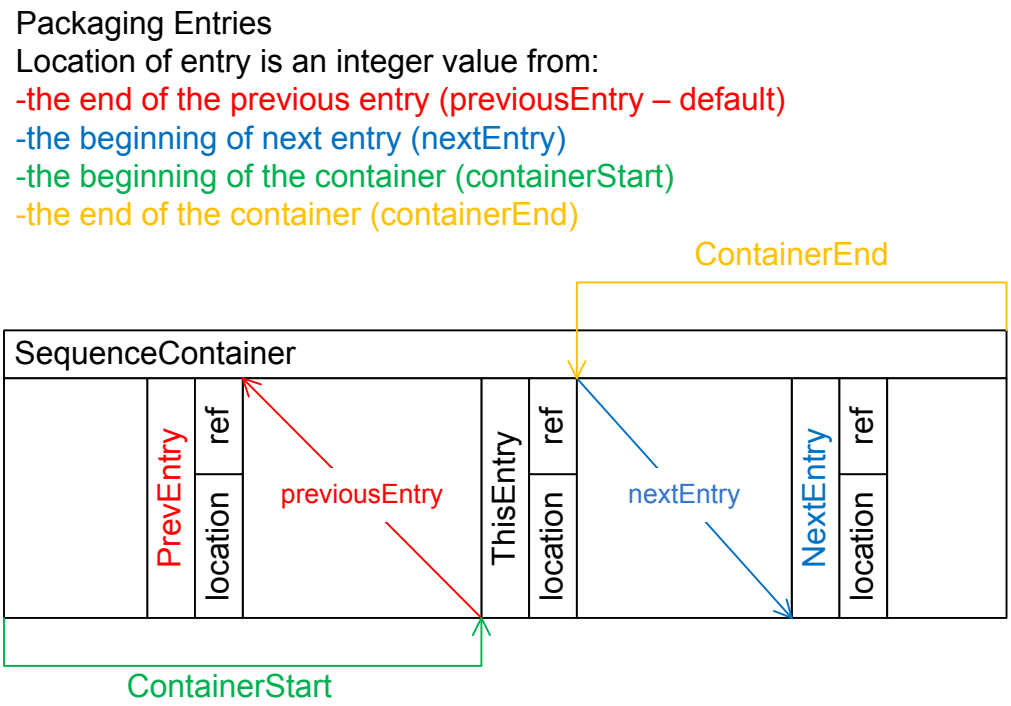



**Figure 5-1: EntryList Addressing**

If DynamicValue or DiscreteLookupList are used, the offset of the parameter and exact size of the container cannot be determined statically.

A range may be determined by looking at the:

– ValidRange;

– sizeInBits of the parameter DataEncoding;

– maximum allowed size of container for the format; and

– container's own sizeInBits.

#### 5.3.2.2 With a ParameterRefEntry

##### 5.3.2.2.1 General

The ParameterEntry's address is adjusted by the offset, which is given in bits based on the ReferenceLocation.

##### 5.3.2.2.2 Example #1—PrevEntry

PrevEntry with a value of zero is the default. This example shows a gap created using it explicitly.

```
<xtce:EntryList>
    <xtce:ParameterRefEntry parameterRef="P1"/>
    <xtce:ParameterRefEntry parameterRef="P2">
        <xtce:LocationInContainerInBits referenceLocation="previousEntry">
            <xtce:FixedValue>44</xtce:FixedValue>
        </xtce:LocationInContainerInBits>
    </xtce:ParameterRefEntry>
</xtce:EntryList>
```

Parameter P2 should be placed 44 bits past the end of P1. If P1 is 16-bits wide, then P2's address is 50, while P1's address is 0.

**Table 5-1: PrevEntry Example**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P1 | 16 | 0 | 0 |
| P2 | 16 | 44 from prevEntry | 50 = 44 + 16 |

##### 5.3.2.2.3 Example #2—ContainerStart

ContainerStart is an absolute addressing from zero from the start of the container.

```
<xtce:EntryList>
    <xtce:ParameterRefEntry parameterRef="P1"/>
    <xtce:ParameterRefEntry parameterRef="P2">
        <xtce:LocationInContainerInBits referenceLocation="containerStart">
            <xtce:FixedValue>44</xtce:FixedValue>
        </xtce:LocationInContainerInBits>
    </xtce:ParameterRefEntry>
</xtce:EntryList>
```

In this example, P1's address is again at zero, and P2 is at address 44.

**Table 5-2: ContainerStart Example**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P1 | 16 | 0 | 0 |
| P2 | 16 | 44 from ContainerStart | 44 |

#### 5.3.2.2.4 Example #3—ContainerEnd

ContainerEnd addresses are given as positive numbers relative to the actual container end.

```
<xtce:SequenceContainer name="ContainerEnd">
    <xtce:BinaryEncoding>
        <xtce:SizeInBits>
            <xtce:FixedValue>100</xtce:FixedValue>
        </xtce:SizeInBits>
    </xtce:BinaryEncoding>
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="P1"/>
        <xtce:ParameterRefEntry parameterRef="P2">
            <xtce:LocationInContainerInBits referenceLocation="containerEnd">
                <xtce:FixedValue>44</xtce:FixedValue>
            </xtce:LocationInContainerInBits>
        </xtce:ParameterRefEntry>
    </xtce:EntryList>
</xtce:SequenceContainer>
```

In this example, P1 is still at zero, but P2 is 44 bits from containerEnd. In fact, the end of P2 is 44 bits from containerEnd. So if P2 is 16 bits in width, then P2's address is at 100−(44 + 16) = 40.

**Table 5-3: ContainerEnd Example**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P1 | 16 | 0 | 0 |
| P2 | 16 | 44 from containerEnd | 40 = 100−(44 + 16) |

#### 5.3.2.2.5 Example #4—NextEntry

NextEntry needs to be used in conjunction with a fixed address. This could be an item in the container's EntryList or the known size of the container (or range).

```
<xtce:EntryList>
    <xtce:ParameterRefEntry parameterRef="P1">
        <xtce:LocationInContainerInBits referenceLocation="nextEntry">
            <xtce:FixedValue>32</xtce:FixedValue>
        </xtce:LocationInContainerInBits>
```

```
        </xtce:ParameterRefEntry>
        <xtce:ParameterRefEntry parameterRef="P2">
            <xtce:LocationInContainerInBits referenceLocation="containerStart">
                <xtce:FixedValue>100</xtce:FixedValue>
            </xtce:LocationInContainerInBits>
        </xtce:ParameterRefEntry>
    </xtce:EntryList>
```

Here P2 is at offset 100, and the end of P1 is 32 bits before this location. If P1 is 16 bits wide, its starting address would be 100-(32 + 16) = 52.

**Table 5-4: NextEntry Example**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P1 | 16 | 32 from nextEntry | 52 = 100−(32 + 16) |
| P2 | 16 | 100 from ContainerStart | 100 |

### 5.3.2.3 With a ContainerRefEntry

A ContainerRefEntry behaves in a similar way to a ParameterRefEntry (the container can be viewed as a large ParameterRefEntry). First, the entire referenced container's EntryList should be treated as one cohesive unit, and its EntryList addresses should be computed. The result should then be inserted as a block into the referencing container. This block may itself then be modified by its own LocationInContainerInBits and repeat. Overall, these are similar concepts to those of ParameterRefEntry in 5.3.2.2.

```
    <xtce:SequenceContainer name="Container">
        <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="P1"/>
            <xtce:ContainerRefEntry containerRef="ContainerAbsolute"/>
        </xtce:EntryList>
        </xtce:SequenceContainer>
    <xtce:SequenceContainer name="ContainerAbsolute">
        <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="P2">
                <xtce:LocationInContainerInBits referenceLocation="containerStart">
                    <xtce:FixedValue>64</xtce:FixedValue>
                </xtce:LocationInContainerInBits>
            </xtce:ParameterRefEntry>
            <xtce:ParameterRefEntry parameterRef="P3">
                <xtce:LocationInContainerInBits referenceLocation="containerStart">
                    <xtce:FixedValue>128</xtce:FixedValue>
                </xtce:LocationInContainerInBits>
            </xtce:ParameterRefEntry>
        </xtce:EntryList>
```

```
</xtce:SequenceContainer>
```

In this example, ContainerAbsolute should be resolved first. Its entries will be offset 64 and 128 bits from the beginning of ContainerAbsolute. Then these items will be inserted into the container's EntryList. This will take place relative to the end of P1 as the LocationInContainerInBits of the ContainerRefEntry is set to 'prevEntry', with an address of '0'. Thus when the addresses of P2 and P3 are inserted into the container, they are offset by the end of P1. So the addresses become P1 = 0, P2 = (16 + 64) = 80, and P3 = (16 + 128) = 144.

**Table 5-5: ContainerAbsolute (SizeInBits = 144)**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P2 | 16 | 64 from containerStart | 64 |
| P3 | 16 | 128 from containerStart | 128 |

The container's sizeInBits is calculated as 144 because the last Parameter (P3) is 128 bits from the containerStart and 16 bits long, making a total of 144 bits.

**Table 5-6: Container (SizeInBits = 160)**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P1 | 16 | 0 | 0 |
| ContainerAbsolute | 144 | 16 | 16 |

The container referred to as ContainerAbsolute can be viewed as a single unit for the purposes of determining the overall start-bit location for its entries and the overall size. The containerRef's entries would be further resolved to the locations shown in table 5-7.

**Table 5-7: Container: Resolved (SizeInBits = 160)**

| Name | Bit Width | Offset | Address |
|---|---|---|---|
| P1 | 16 | 0 | 0 |
| ContainerAbsolute:P2 | 16 | 64 + 16 (prevEntry) | 80 |
| ContainerAbsolute:P3 | 16 | 128 + 16 | 144 |

### 5.3.3 REPEATENTRY

RepeatEntry allows one to specify a repeating item. It may be a super-sampled or a super-commutated parameter or container (see 4.3.4.8.10.3).

A RepeatEntry is defined as the original entry + (the original entry × repeat count). A RepeatEntry of '0' is the same as an entry without RepeatEntry, while a RepeatEntry of '1' means the entry is duplicated.

In the following example, the entry repeats four times, meaning there are five entries. The offset between them is zero since they are 'packed' back to back.

```
<xtce:ParameterRefEntry parameterRef="P1">
    <xtce:RepeatEntry>
        <xtce:Count>
            <xtce:FixedValue>4</xtce:FixedValue>
        </xtce:Count>
        <xtce:Offset>
            <xtce:FixedValue>0</xtce:FixedValue>
        </xtce:Offset>
    </xtce:RepeatEntry>
</xtce:ParameterRefEntry>
```

### 5.3.4 INCLUDECONDITION

IncludeCondition allows one to specify the optional inclusion of a parameter or container in an EntryList (see 4.3.4.8.10.4). If the conditions are true, the including container will change size to accommodate the container or parameter being referenced.

```
<xtce:ParameterRefEntry parameterRef="CheckSum">
    <xtce:IncludeCondition>
        <xtce:Comparison value="1" parameterRef="CSFlag"/>
    </xtce:IncludeCondition>
</xtce:ParameterRefEntry>
```

Here, the parameter CheckSum is included if the parameter instance of CSFlag (instance 0) is set to '1'. In this example, the CSFlag is a flag in the header.

### 5.3.5 IN COMBINATION

It is possible that a particular Entry will have some combination of LocationInContainerInBits, RepeatEntry, and IncludeCondition set. In general, IncludeCondition should be processed first, followed by RepeatEntry and finally by LocationInContainerInBits. Any RepeatEntry should be treated as a single unit.

```
<xtce:ParameterRefEntry parameterRef="P1">
    <xtce:LocationInContainerInBits referenceLocation="containerStart">
        <xtce:FixedValue>0</xtce:FixedValue>
    </xtce:LocationInContainerInBits>
    <xtce:RepeatEntry>
        <xtce:Count>
            <xtce:FixedValue>4</xtce:FixedValue>
        </xtce:Count>
        <xtce:Offset>
```

```
            <xtce:FixedValue>0</xtce:FixedValue>
        </xtce:Offset>
    </xtce:RepeatEntry>
    <xtce:IncludeCondition>
        <xtce:Comparison value="1" parameterRef="ID"/>
    </xtce:IncludeCondition>
</xtce:ParameterRefEntry>
```

In the example above, the Parameter P1 is only valid if ID (instance 0) is '1'. It also repeats a total of five times and has an absolute address from containerStart.

The example below shows a ContainerRefEntry and an IncludeCondition.

```
<xtce:SequenceContainer name="Included">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="CP1">
            <xtce:LocationInContainerInBits referenceLocation="containerStart">
                <xtce:FixedValue>100</xtce:FixedValue>
            </xtce:LocationInContainerInBits>
        </xtce:ParameterRefEntry>
        <xtce:ParameterRefEntry parameterRef="CP2">
            <xtce:LocationInContainerInBits referenceLocation="containerStart">
                <xtce:FixedValue>200</xtce:FixedValue>
            </xtce:LocationInContainerInBits>
        </xtce:ParameterRefEntry>
        <xtce:ParameterRefEntry parameterRef="CP3">
            <xtce:LocationInContainerInBits referenceLocation="containerStart">
                <xtce:FixedValue>300</xtce:FixedValue>
            </xtce:LocationInContainerInBits>
        </xtce:ParameterRefEntry>
    </xtce:EntryList>
    </xtce:SequenceContainer>
<xtce:SequenceContainer name="Plain">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="P1"/>
        <xtce:ParameterRefEntry parameterRef="P2"/>
        <xtce:ContainerRefEntry containerRef="Included"/>
        <xtce:ParameterRefEntry parameterRef="P3"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
```

In the above example, the container 'Included' has entries within it that are offset from ContainerStart. Those addresses are relative to the container in which they are defined (i.e., Included), not the container that references Included (i.e., Plain). Thus the container Included would be resolved first, and then its entries would be modified by the LocationInContainerInBits of the containerRef in question. In this example, the default applies, so the LocationInContainerInBits has a value of 'prevEntry' and an offset of '0'. That means each entry, when inserted into container Plain, is offset from the entry P2, not containerStart.

**Table 5-8: Plain Container**

<table>
<tr><th colspan="4">RefEntry</th><th>Start Address</th><th>Width</th></tr>
<tr><td colspan="4">P1</td><td>0</td><td>16</td></tr>
<tr><td colspan="4">P2</td><td>16</td><td>16</td></tr>
<tr><td>Included</td><td>RefEntry</td><td>Start Addr+32</td><td>Width</td><td>32</td><td>316</td></tr>
<tr><td></td><td>CP1</td><td>100 (+32)</td><td>16</td><td></td><td></td></tr>
<tr><td></td><td>CP2</td><td>200 (+32)</td><td>16</td><td></td><td></td></tr>
<tr><td></td><td>CP3</td><td>300 (+32)</td><td>16</td><td></td><td></td></tr>
<tr><td colspan="4">P3</td><td>348</td><td>16</td></tr>
</table>

Subsection 5.4 contains more information on container addressing.

The other EntryList forms are simply variations of either ParameterRefEntry or ContainerRefEntry. All of them except array are segments that simply clip the size of the item being referred to from its inherent size. The ArrayParameterRefEntry is somewhat special because it is here that the dimension sizes are stored.

## 5.4 USING ABSTRACT, PLAIN AND CONCRETE CONTAINERS TO BUILD PACKAGING DEFINITIONS

### 5.4.1 GENERAL

In XTCE, the three container forms can be used to build packaging definitions for a format (such as a packet or minor frame). One simple approach uses a common root abstract container and a set of concrete containers extending it. The appropriate identification information is supplied in the RestrictionCriteria (as well as the entries for the packet bodies).

One level of inheritance may be enough to capture all packet constructions. Additional levels can be used depending on various requirements and the preferences of the designers of the descriptions.

### 5.4.2 CONTAINER INHERITANCE CONCEPTS

#### 5.4.2.1 General

The following subsections use an object-oriented-like mechanism to describe telemetry (and command) packaging (such as packet).

#### 5.4.2.2 Processing a Conceptual Container Inheritance Tree

One way to think of container inheritance is in relation to a system that takes as input a set of telemetry packets and processes those against an inheritance tree of XTCE containers. Such a

system must read the packets and match their contents to the tree of containers, starting at the root container and working down to leaf containers:

If the definitions consist of a root abstract container representing the header, a set of concrete containers is used to extend it and supply uniquely identifying information. In that context, the first packet's byte (or bytes) would start at the root container.

If the container is abstract, no instance of it is created, and processing continues down the container inheritance chain until the leaf containers are processed. In this example, there is only one level of inheritance, so the container leaves are processed.

For each match that occurs, a concrete instance of that packet is created based on the container definitions. Every entry in the EntryList is processed as well as those in the parent container, and all parameters are used to decommutate the packet into the telemetry table.

From an exchange standpoint, it is likely that a given format does not have a container inheritance feature.

#### 5.4.2.3 Treating BaseContainer As an Operation

##### 5.4.2.3.1 General

Treating BaseContainer as an operation processes the container inheritance constructs away, possibly all the way to a single entry list per packet (or minor frame, etc.).

Along with these are the set of restrictions that must be met to uniquely identify those parameters in at least a conceptual set of input packets. Other information is held in the other elements (such as the descriptive information). This simplified description can then be transferred to another format with some ease.

When the BaseContainer is interpreted as an operation, the ContainerSet is looped through and all nonabstract containers with a BaseContainer are processed further. When this occurs, the container's BaseContainer is recursively processed, and a single EntryList is formed by adding the various entries into a single list.

The RestrictionCriteria are collected relative to where they are defined in each inheritance level so that the scope of the parameters used in them is maintained.

More details of this process are given below.

##### 5.4.2.3.2 Container Inheritance: Building Up the EntryList

The container below consists of a single abstract root container that represents a header. A second container is used to represent a secondary header (timestamp). A single concrete container extends them and then adds some entries, which make up the packet body. Taken together, these represent a single packet description.

```
<xtce:SequenceContainer name="MyFormatHeader" abstract="true">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="Version"/>
        <xtce:ParameterRefEntry parameterRef="Type"/>
        <xtce:ParameterRefEntry parameterRef="ID"/>
        <xtce:ParameterRefEntry parameterRef="Length"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="MyTimeStampHeader" abstract="true">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="TimeStamp"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="MyFormatHeader">
        <xtce:RestrictionCriteria>
            <xtce:ComparisonList>
                <xtce:Comparison parameterRef="Version" value="1"/>
                <xtce:Comparison parameterRef="Type" value="1"/>
            </xtce:ComparisonList>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="MyConcretePacket">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="P1"/>
        <xtce:ParameterRefEntry parameterRef="P2"/>
        <xtce:ParameterRefEntry parameterRef="P3"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef=" MyTimeStampHeader ">
        <xtce:RestrictionCriteria>
            <xtce:Comparison parameterRef="ID" value="100"/>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

Because the BaseContainer is treated as an operation, the result will be a singular object or entity. It is shown below as XML.

**Step #1:** If a BaseContainer is present in a nonabstract container, it is recursively processed until the root container is reached.

As the recursion unwinds, each container's entries are added to a final EntryList construct. If its entries refer to other containers, the entries must be resolved to ParameterEntryRefs.

The various other items associated with container inheritance and the original container's description should be processed into one object. For example, the AncillaryData should be accumulated according to the inheritance rules.

It is important to capture the RestrictionCriteria in such a way that each one's scope is maintained relative to where it was defined in each container. In terms of implementation,

one way to do this is to allow the restrictions to be added into the final EntryList itself after each container's entries have been added. This can be done in a variety of ways in different programming languages. The assumption is it is possible to do so.

In the example below, the RestrictionCriteria conditions have been placed at the end of each container's entries, maintaining their place relative to their original container definition. The XML illustrates that this has occurred and then shows the result of the operation.

NOTE – Once again, the resulting processed construction shown below is not itself in XTCE syntax.

```
<ProcessedContainer name="MyConcretePacket">
    <ObjectEntryList>
        <ParameterRefEntry parameterRef="Version"/>
        <ParameterRefEntry parameterRef="Type"/>
        <ParameterRefEntry parameterRef="ID"/>
        <ParameterRefEntry parameterRef="Length"/>
        <ParameterRefEntry parameterRef="TimeStamp"/>
        <RestrictionCriteria>
            <ComparisonList>
                <Comparison parameterRef="Version" value="1"/>
                <Comparison parameterRef="Type" value="1"/>
            </ComparisonList>
        </RestrictionCriteria>
        <ParameterRefEntry parameterRef="P1"/>
        <ParameterRefEntry parameterRef="P2"/>
        <ParameterRefEntry parameterRef="P3"/>
                    <RestrictionCriteria>
            <Comparison parameterRef="ID" value="100"/>
        </RestrictionCriteria>
    </ObjectEntryList>
</ProcessedContainer>
```

**Step #2:** Now that all entries have been resolved into one object, another step might be to derive the addresses of the entries.

Deriving the addresses of the entries also means any Repeat, LocationInContainerInBits, and IncludeCondition must be considered as any of the other entry forms such as ArrayParameterRefEntry.

Each parameter associated with each ParameterRefEntry is found in a ParameterSet, and its ParameterTypes are then found in a ParameterTypeSet. Their DataEncodings provide the size of each, as well as other information such as alarms, calibrators, and so forth. Again, fixed-sized parameters are easiest to deal with, but ranges can be calculated for the other cases.

The following example is a simple case, where each parameter is fixed and 16 bits long.

```
<AddressContainer name="MyConcretePacketAddressesResolved" calculatedSizeInBits="0x80"/>
    <ObjectEntryList>
```

```
        <Parameter name="Version" address="0x00" sizeInBits="16" etc />
        <Parameter name="Type" address="0x10" sizeInBits="16" etc />
        <Parameter name="ID" address="0x20" sizeInBits="16" etc />
        <Parameter name="Length" address="0x30" sizeInBits="16" etc />
            <Parameter name="TimeStamp" address="0x40" sizeInBits="16" etc />
                        <RestrictionCriteria>
            <ComparisonList>
                <Comparison parameterName="Version" value="1"/>
                <Comparison parameterName="Type" value="1"/>
            </ComparisonList>
        </RestrictionCriteria>
        <Parameter name="P1" address="0x50" sizeInBits="16" etc />
        <Parameter name="P2" address="0x60" sizeInBits="16" etc />
        <Parameter name="P3" address="0x70" sizeInBits="16" etc />
                        <RestrictionCriteria>
            <Comparison parameterName="ID" value="100"/>
        </RestrictionCriteria>
    </ObjectEntryList>
</AddressContainer>
```

The above XML is a representation of the final resolved construction (once again, it is not XTCE syntax). It is an address map of all the revolved BaseContainer and EntryList items including the set of conditions that must be true for an incoming packet match to occur.

If such a match occurs (that is, the restrictions are all true), then the remaining information associated with each parameter (such as calibrators and alarms) would be processed to derive instance values for each item. This additional information is not shown here.

NOTES

1 From an exchange standpoint, this type of processed container object removes the inheritance constructs, which should make transfer to another description format easier.

2 The RestrictionCriteria may still be an issue; for example, complex expressions may not transfer easily to a format that expects one identification key per packet definition.

3 The recommended approach at this time is to constrain the form of the expressions so that processing a certain pattern is expected. If possible, the expressions only use the operator '==’.

### 5.4.3 OTHER ITEMS IN CONTAINER INHERITANCE

Many of the other items in SequenceContainer take part in the inheritance process. (See 4.3.4.9.5 for the rules associated with container inheritance.)

Container size is affected by inheritance and BinaryEncoding/SizeInBits. It is easier to simply calculate the size from the description and leave off the BinaryEncoding/SizeInBits.

### 5.4.4 COMMANDCONTAINERSET INHERITANCE

Container inheritance semantics are described in 4.3.4.9. CommandContainerSet is exactly the same as ContainerSet in TelemetryMetaData.

## 5.5 DYNAMIC CONTAINER MATCHING

Dynamic container matching is briefly described in 4.3.4.8.5.2. It occurs when a ContainerRefEntry in a container's EntryList refers to an abstract container, and that abstract container has its own derived concrete child containers.

The following rules define dynamic container matching:

- The ContainerRefEntry refers to a container marked as abstract (i.e., an abstract container).
- The abstract container has one or more derived concrete child containers. If true, zero or more of the child containers (including the inherited EntryList from the abstract parent) may be placed in the original container's EntryList (i.e., at the ContainerRefEntry location). If the expressions in their RestrictionCriteria evaluate to true, the following events occur:
  - If more than one is true, the child containers form a *union* of the containers, and the entire construct is placed in the EntryList.
  - If no child matches occur, the abstract container itself is inserted. (Its EntryList is likely empty for most cases.)
- The ContainerRefEntry itself may have an IncludeCondition. If so, this is evaluated first, and then the dynamic matching conditions (i.e., the RestrictionCriteria) are evaluated. If the IncludeCondition is true, the dynamic match proceeds.
- A union of two or more containers is defined as a block large enough to hold the largest EntryList among them. The EntryLists are not merged, but literally one container overlays the other.
- The order of evaluation of the conditions in each derived child are defined by the implementer.
- If the abstract container has a nonempty EntryList, then its content is included even if none of its derived child containers' conditions are true.
- The derived child containers *inherit* the abstract container's entries.

The construction is a choice of zero or more ContainerRefEntries. The example below illustrates the basic concepts outlined above, given the MainContainer with a ContainerRefEntry to an abstract container called ‘AbstractContainer’:

```
<xtce:SequenceContainer name="MainContainer">
    <xtce:EntryList>
        <xtce:ContainerRefEntry containerRef="AbstractContainer"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
```

The AbstractContainer then begins the dynamic match definition. In this example, its EntryList is empty. This is easy to implement as derived child containers.

```
<xtce:SequenceContainer name="AbstractContainer" abstract="true">
    <xtce:EntryList/>
</xtce:SequenceContainer>
```

Each derived child container has some content that will be included depending on the value of an expression called ‘Condition’ in the two RestrictionCriteria.

```
<xtce:SequenceContainer name="ContainerA">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="A"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="AbstractContainer">
        <xtce:RestrictionCriteria>
            <xtce:Comparison parameterRef="Condition" value="1"/>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
    </xtce:SequenceContainer>
<xtce:SequenceContainer name="ContainerB">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="B"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="AbstractContainer">
        <xtce:RestrictionCriteria>
            <xtce:Comparison parameterRef="Condition" value="2"/>
        </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

The construct then is defined so that only one container (A or B) actually supplies any content into MainContainer. If Condition = ‘1’, then container A is inserted into MainContainer. If Condition = ‘2’, then container B is inserted into MainContainer. If Condition = ‘3’, then only AbstractContainer is inserted, and it has no content.

As stated in the rules, the abstract container is always included, so care should be taken to define the various constructs properly to match the definition desired.

The following made-up construct *ChoiceOfZeroOrMoreDerivedContainers* illustrates the desired functionality. This is made up for the purposes of illustration; it is not XTCE syntax.

```
<SequenceContainer name="MainContainerIllustrated">
    <EntryList>
        <ChoiceOfZeroOrMoreDerivedContainers>
            <ContainerRefEntry containerRef="AbstractContainer" default="alwaysInclude"/>
            <ContainerRefEntry containerRef="ContainerA" condition="Condition" value="1"/>
            <ContainerRefEntry containerRef="ContainerB" condition="Condition" value="2"/>
        </ChoiceOfZeroOrMoreDerivedContainers>
    </EntryList>
</SequenceContainer>
```

The order of the child entries is not explicitly defined by the rules:

```
<SequenceContainer name="MainContainerIllustrated">
    <EntryList>
        <ChoiceOfZeroOrMoreDerivedContainers>
            <ContainerRefEntry containerRef="AbstractContainer" default="alwaysInclude"/>
            <ContainerRefEntry containerRef="ContainerB" condition="Condition" value="1"/>
            <ContainerRefEntry containerRef="ContainerA" condition="Condition" value="2"/>
        </ChoiceOfZeroOrMoreDerivedContainers>
    </EntryList>
</SequenceContainer>
```

The dynamic match mechanism can be used where holes in a container definition exist. For example, it may be that under normal conditions a telemetry packet has no timestamp, but in some cases, a 32-bit timestamp is present, and in other cases, a higher-resolution, 48-bit timestamp is present.

One possible solution to this construction in XTCE would be to define a packet container with two consecutive ContainerRefEntries. They would have IncludeConditions for the two TimeStamp containers for each resolution time stamp.

The conditions need to be defined in such way as to ensure that the three cases desired are created, that is, no timestamp, a 32-bit timestamp, or the 48-bit timestamp. Using this technique, such a construction is a sequence of optional entries in the packet container.

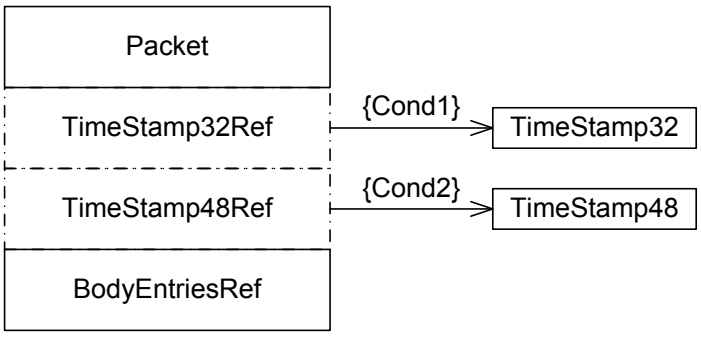



**Figure 5-2: Using IncludeConditions**

The entries are optional in the sense shown above, as 'Cond1' or 'Cond2' are IncludeConditions controlling their inclusion into the original container.

In the dynamic match variation, the packet container would have a ContainerRefEntry to an abstract *TimeStamp* container with no entries. Two derived containers form the TimeStamps, one for the 32-bit time stamp, and the other for the 48-bit variation. The conditions should evaluate so that either:

– no time stamp is included (in the example below, assume *TimeStamp* has an empty EntryList, and Cond1 and Cond2 are false);

– a 32-bit TimeStamp32 is included (Cond2 is true); or

– a 48-bit TimeStamp48 is included (Cond1 is true).

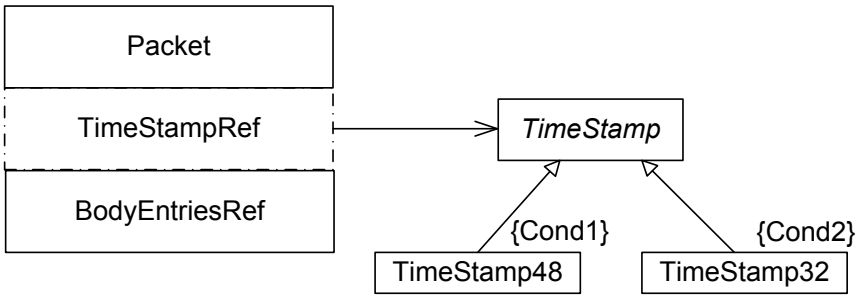



**Figure 5-3: Using Dynamic Container Matching**

### Example

The following example uses the one outlined above. A system defines telemetry with three possible secondary headers depending on a certain condition: none, a regular time stamp, or a hi-res time stamp. Dynamic container matching is used to describe this scenario.

```
<xtce:SequenceContainer name="Header" abstract="true">
    <xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="Version"/>
        <xtce:ParameterRefEntry parameterRef="SecondaryHeaderFlag"/>
        <xtce:ParameterRefEntry parameterRef="ID"/>
        <xtce:ParameterRefEntry parameterRef="Length"/>
    </xtce:EntryList>
    </xtce:SequenceContainer>
<xtce:SequenceContainer name="MyPacket1">
    <xtce:EntryList>
        <xtce:ContainerRefEntry containerRef="MySecondaryHeader"/>
        <xtce:ParameterRefEntry parameterRef="P1"/>
        <xtce:ParameterRefEntry parameterRef="P2"/>
        <xtce:ParameterRefEntry parameterRef="P3"/>
    </xtce:EntryList>
    </xtce:SequenceContainer>
<xtce:SequenceContainer name="MySecondaryHeader" abstract="true">
```

```
        <xtce:EntryList/>
        </xtce:SequenceContainer>
    <xtce:SequenceContainer name="TimeStamp1">
        <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="Seconds"/>
            <xtce:ParameterRefEntry parameterRef="Milliseconds"/>
        </xtce:EntryList>
        <xtce:BaseContainer containerRef="MySecondaryHeader">
        <xtce:RestrictionCriteria>
            <xtce:ComparisonList>
            <xtce:Comparison parameterRef="SecondaryHeaderFlag" value="1"/>
            <xtce:Comparison parameterRef="ID" comparisonOperator="&lt;" value="100"/>
            </xtce:ComparisonList>
        </xtce:RestrictionCriteria>
        </xtce:BaseContainer>
        </xtce:SequenceContainer>
    <xtce:SequenceContainer name="TimeStamp2">
        <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="Seconds"/>
            <xtce:ParameterRefEntry parameterRef="Milliseconds"/>
            <xtce:ParameterRefEntry parameterRef="Microseconds"/>
        </xtce:EntryList>
        <xtce:BaseContainer containerRef="MySecondaryHeader">
        <xtce:RestrictionCriteria>
        <xtce:ComparisonList>
            <xtce:Comparison parameterRef="SecondaryHeaderFlag" value="1"/>
            <xtce:Comparison parameterRef="ID" comparisonOperator="&gt;=" value="100"/>
        </xtce:ComparisonList>
        </xtce:RestrictionCriteria>
        </xtce:BaseContainer>
    </xtce:SequenceContainer>
```

NOTES

1 In the example above, the *MySecondaryHeader* container is only included if SecondaryHeaderFlag is ‘1’.

2 If the IncludeCondition is true (SecondaryHeaderFlag == ‘1’), then dynamic container matching can be processed.

3 In the first case, if the ID of the packet is below 100, then TimeStamp1 will be included. But if ID is 100 or larger, then TimeStamp2 will be included.

4 If neither matches, no descriptive content is added.

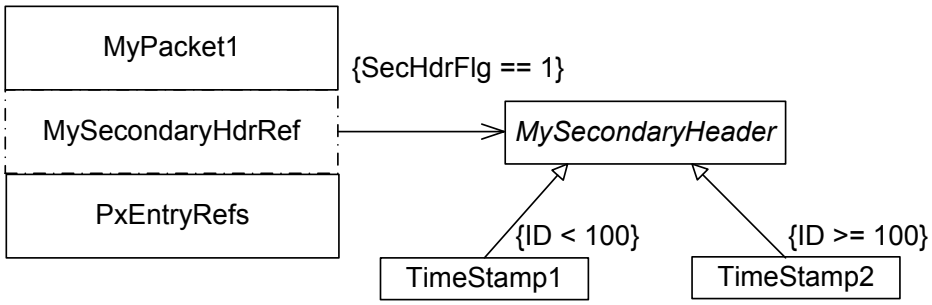



**Figure 5-4: Dynamic Container Match Example**

## 5.6 METACOMMAND INHERITANCE

### 5.6.1 GENERAL

While MetaCommand inheritance semantics are described in 4.4.5.2.3, this subsection focuses on the construction of concrete MetaCommands and their CommandContainers.

MetaCommand inheritance takes place using the BaseMetaCommand element. Through this inheritance mechanism, the child MetaCommand may inherit certain aspects of the parent MetaCommand. It can also optionally supply argument values to parent arguments.

### 5.6.2 METACOMMAND/COMMANDCONTAINER INHERITANCE

The MetaCommand/CommandContainer/BaseContainer must be supplied in order for the child command to inherit the EntryList of the parent.

The MetaCommand/CommandContainer/BaseContainer element behaves in a similar way to the other container forms of inheritance (see 4.3.4.9).

**Examples**

In this example a command is defined with MetaCommand. It takes one argument. Two commands are then defined from it. Both set the argument to ‘lock in’ the user input.

```
<xtce:MetaCommandSet>
    <xtce:MetaCommand name="Power">
        <xtce:ArgumentList>
        <xtce:Argument argumentTypeRef="ONorOFFEnum" name="PowerState"/>
        </xtce:ArgumentList>
        <xtce:CommandContainer name="PowerPacket">
            <xtce:EntryList>
                <xtce:FixedValueEntry binaryValue="AA"/>
                <xtce:ArgumentRefEntry argumentRef="PowerState"/>
            </xtce:EntryList>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
```

```
    <xtce:MetaCommand name="PowerON">
        <xtce:BaseMetaCommand metaCommandRef="Power">
        <xtce:ArgumentAssignmentList>
        <xtce:ArgumentAssignment argumentName="PowerState" argumentValue="1"/>
        </xtce:ArgumentAssignmentList>
        </xtce:BaseMetaCommand>
        <xtce:CommandContainer name="PowerONPacket">
            <xtce:EntryList/>
            <xtce:BaseContainer containerRef="PowerPacket"/>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
    <xtce:MetaCommand name="PowerOFF">
        <xtce:BaseMetaCommand metaCommandRef="Power">
        <xtce:ArgumentAssignmentList>
        <xtce:ArgumentAssignment argumentName="PowerState" argumentValue="0"/>
        </xtce:ArgumentAssignmentList>
        </xtce:BaseMetaCommand>
        <xtce:CommandContainer name="PowerOFFPacket">
            <xtce:EntryList/>
            <xtce:BaseContainer containerRef="PowerPacket"/>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
</xtce:MetaCommandSet>
```

In the above example, a simple ‘Power’ command is defined. It has an ‘opcode’ of ‘0xAA’ and takes one argument. It does not care if the power is ‘OFF’ or ‘ON’.

Two additional commands are defined from Power; each sets the argument to either PowerON or PowerOFF.

In both cases, the MetaCommand/CommandContainer of each derives from the Power/PowerPacket MetaCommand/CommandContainer. This reference must be set explicitly in XTCE. The EntryList is empty in each since it provides no additional items to the main command.

### 5.6.3 TREATING BASEMETACOMMAND AS AN OPERATION

#### 5.6.3.1 General

This subsection is similar to 5.4.2.3, which describes treating BaseContainer as operational.

This subsection presents an approach base around the idea of treating the inheritance as a processable operation. The result is an object or entity that has accumulated the various items defined into one entity.

In the case of MetaCommands, there is a ‘dual’ inheritance mechanism. First, the MetaCommand/BaseMetaCommand inheritance chain can be processed. Once complete, its CommandContainer/BaseContainer can be processed.

To determine which MetaCommands to process, the abstract attribute is used as the guide. Any nonabstract MetaCommand represents a real command instance. These can be processed and resolved to aggregated objects.

#### 5.6.3.2 Extended Command Example

The first item shown is the CCSDS header container. This container is defined in ContainerSet, and in this particular implementation, it is shared by both 'sides', telemetry and commanding.

```
<xtce:SequenceContainer abstract="true" name="CCSDSPacket">
    <xtce:LongDescription>
            Super-container for all CCSDS telemetry and command packets
            </xtce:LongDescription>
        <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="CCSDSVersion"/>
            <xtce:ParameterRefEntry parameterRef="CCSDSType"/>
            <xtce:ParameterRefEntry parameterRef="CCSDSSecH"/>
            <xtce:ParameterRefEntry parameterRef="CCSDSAPID"/>
            <xtce:ParameterRefEntry parameterRef="CCSDSGroupFlags"/>
            <xtce:ParameterRefEntry parameterRef="CCSDSSourceSequenceCount"/>
            <xtce:ParameterRefEntry parameterRef="CCSDSPacketLength"/>
        </xtce:EntryList>
</xtce:SequenceContainer>
```

On the commanding side, first an ArgumentType is shown. It is used by all the commands being presented and is an enumeration that has three values: 'On', 'Off', or 'NoChange'.

```
<xtce:ArgumentTypeSet>
    <xtce:EnumeratedArgumentType name="RelayStateType">
        <xtce:UnitSet/>
        <xtce:IntegerDataEncoding sizeInBits="3"/>
        <xtce:EnumerationList>
            <xtce:Enumeration value="0" label="On"/>
            <xtce:Enumeration value="1" label="Off"/>
            <xtce:Enumeration value="2" label="NoChange"/>
        </xtce:EnumerationList>
        </xtce:EnumeratedArgumentType>
</xtce:ArgumentTypeSet>
```

In MetaCommandSet, a *super* or *root* command is described. This is an abstraction, and it conveys little real information (the EntryList is empty) except for certain values associated with some packet header fields that are true for all commands (in this conceptual system). It extends CCSDSPacket.

```
<xtce:MetaCommand abstract="true" name="CCSDSCommand">
    <xtce:LongDescription>
            Super-Command for all CCSDS commands.
            </xtce:LongDescription>
    <xtce:CommandContainer name="CCSDSCommandPacket">
```

```
            <xtce:LongDescription>
                Super-container for all CCSDS command packets.
                </xtce:LongDescription>
            <xtce:EntryList/>
            <xtce:BaseContainer containerRef="CCSDSPacket">
                <xtce:RestrictionCriteria>
                    <xtce:ComparisonList>
                    <xtce:Comparison value="0" parameterRef="CCSDSVersion"/>
                    <xtce:Comparison value="1" parameterRef="CCSDSType"/>
                    <xtce:Comparison value="0" parameterRef="CCSDSSecH"/>
                    <xtce:Comparison value="3" parameterRef="CCSDSGroupFlags"/>
                    </xtce:ComparisonList>
                </xtce:RestrictionCriteria>
            </xtce:BaseContainer>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
```

After this, a SetRelay command is defined. Since it is not marked abstract, it is itself a command instance. The general form of the command is:

```
    <xtce:MetaCommand abstract="false" name="SetRelays">
        <xtce:BaseMetaCommand metaCommandRef="CCSDSCommand"/>
        <xtce:SystemName>HRD</xtce:SystemName>
        <xtce:ArgumentList>
            <xtce:Argument argumentTypeRef="RelayStateType" name="RelayState1"/>
            <xtce:Argument argumentTypeRef="RelayStateType" name="RelayState2"/>
            <xtce:Argument argumentTypeRef="RelayStateType" name="RelayState3"/>
            <xtce:Argument argumentTypeRef="RelayStateType" name="RelayState4"/>
        </xtce:ArgumentList>
        <xtce:CommandContainer name="SetRelaysPacket">
            <xtce:AncillaryDataSet>
                <xtce:AncillaryData name="VCID">0</xtce:AncillaryData>
            </xtce:AncillaryDataSet>
            <xtce:EntryList>
                <xtce:ArgumentRefEntry argumentRef="RelayState1"/>
                <xtce:ArgumentRefEntry argumentRef="RelayState2"/>
                <xtce:ArgumentRefEntry argumentRef="RelayState3"/>
                <xtce:ArgumentRefEntry argumentRef="RelayState4"/>
            </xtce:EntryList>
            <xtce:BaseContainer containerRef="CCSDSCommandPacket">
                <xtce:RestrictionCriteria>
                    <xtce:Comparison value="179" parameterRef="CCSDSAPID"/>
                </xtce:RestrictionCriteria>
            </xtce:BaseContainer>
        </xtce:CommandContainer>
    </xtce:MetaCommand>
```

The command takes four arguments. Along with the header, it inherits through CCSDSCommandPacket. These arguments are used to make up the items in the actual packet.

After this, two derived commands are based on it. The first sets all the relays on. It does this by extending the SetRelay command and providing argument assignments to the various arguments as follows:

```
<xtce:MetaCommand name="SetAllRelaysOn">
    <xtce:BaseMetaCommand metaCommandRef="SetRelays">
        <xtce:ArgumentAssignmentList>
            <xtce:ArgumentAssignment argumentValue="On" argumentName="RelayState1"/>
            <xtce:ArgumentAssignment argumentValue="On" argumentName="RelayState2"/>
            <xtce:ArgumentAssignment argumentValue="On" argumentName="RelayState3"/>
            <xtce:ArgumentAssignment argumentValue="On" argumentName="RelayState4"/>
        </xtce:ArgumentAssignmentList>
    </xtce:BaseMetaCommand>
    <xtce:SystemName>HRD</xtce:SystemName>
    <xtce:CommandContainer name="SetAllRelaysOnPacket">
        <xtce:EntryList/>
        <xtce:BaseContainer containerRef="SetRelaysPacket"/>
    </xtce:CommandContainer>
</xtce:MetaCommand>
```

The purpose of the construction is to completely reuse the general relay command packet. A new command is created by specifying and setting arguments to certain values.

In a real system, the SetRelay command itself may be set to ‘abstract’, and only the various derived command base on it would be accessible to the users. In this example, they are all nonabstract.

The second derived command simply turns off all the relays by supplying the appropriate ‘Off’ flag to ArgumentAssignment.

```
<xtce:MetaCommand name="SetAllRelaysOff">
    <xtce:BaseMetaCommand metaCommandRef="SetRelays">
        <xtce:ArgumentAssignmentList>
            <xtce:ArgumentAssignment argumentValue="Off" argumentName="RelayState1"/>
            <xtce:ArgumentAssignment argumentValue="Off" argumentName="RelayState2"/>
            <xtce:ArgumentAssignment argumentValue="Off" argumentName="RelayState3"/>
            <xtce:ArgumentAssignment argumentValue="Off" argumentName="RelayState4"/>
        </xtce:ArgumentAssignmentList>
    </xtce:BaseMetaCommand>
    <xtce:SystemName>HRD</xtce:SystemName>
    <xtce:CommandContainer name="SetAllRelaysOffPacket">
        <xtce:EntryList/>
        <xtce:BaseContainer containerRef="SetRelaysPacket"/>
    </xtce:CommandContainer>
</xtce:MetaCommand>
```

Three commands are created by using the inheritance mechanism.

#### 5.6.3.3 Processing the Example

Given the example above and the three commands, if the inheritance mechanisms are treated as operations, the result is as follows:

First, the SetRelays command becomes the following:

```
<ProcessedMetaCommand name="SetRelays">
    <SystemName>HRD</SystemName>
    <ArgumentList>
        <Argument argumentTypeRef="RelayStateType" name="RelayState1"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState2"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState3"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState4"/>
    </ArgumentList>
    <ProcessedMetaCommandCommandContainer name="SetRelaysPacket">
        <AncillaryDataSet>
            <AncillaryData name="VCID">0</AncillaryData>
        </AncillaryDataSet>
        <EntryList>
            <ParameterRefEntry parameterRef="CCSDSVersion"/>
            <ParameterRefEntry parameterRef="CCSDSType"/>
            <ParameterRefEntry parameterRef="CCSDSSecH"/>
            <ParameterRefEntry parameterRef="CCSDSAPID"/>
            <ParameterRefEntry parameterRef="CCSDSGroupFlags"/>
            <ParameterRefEntry parameterRef="CCSDSSourceSequenceCount"/>
            <ParameterRefEntry parameterRef="CCSDSPacketLength"/>
            <Restriction>
                <ComparisonList>
                    <Comparison value="0" parameterRef="CCSDSVersion"/>
                    <Comparison value="1" parameterRef="CCSDSType"/>
                    <Comparison value="0" parameterRef="CCSDSSecH"/>
                    <Comparison value="3" parameterRef="CCSDSGroupFlags"/>
                </ComparisonList>
            </Restriction>
            <ArgumentRefEntry argumentRef="RelayState1"/>
            <ArgumentRefEntry argumentRef="RelayState2"/>
            <ArgumentRefEntry argumentRef="RelayState3"/>
            <ArgumentRefEntry argumentRef="RelayState4"/>
            <Restriction>
                <Comparison value="179" parameterRef="CCSDSAPID"/>
            </Restriction>
        </EntryList>
    </ProcessedMetaCommandCommandContainer>
</ProcessedMetaCommand>
```

NOTE – The Restrictions are placed relative to the now-processed-away containers in which they were defined. This helps to maintain their proper scope.

Next, the processed SetRelaysOn command has the following information in it:

```
<ProcessedMetaCommand name="SetAllRelaysOn">
    <ArgumentAssignmentList>
        <ArgumentAssignment argumentValue="On" argumentName="RelayState1"/>
        <ArgumentAssignment argumentValue="On" argumentName="RelayState2"/>
        <ArgumentAssignment argumentValue="On" argumentName="RelayState3"/>
        <ArgumentAssignment argumentValue="On" argumentName="RelayState4"/>
    </ArgumentAssignmentList>
    <SystemName>HRD</SystemName>
    <ArgumentList>
        <Argument argumentTypeRef="RelayStateType" name="RelayState1"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState2"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState3"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState4"/>
    </ArgumentList>
    <ProcessedMetaCommandCommandContainer name="SetAllRelaysOnPacket">
        <AncillaryDataSet>
            <AncillaryData name="VCID">0</AncillaryData>
        </AncillaryDataSet>
        <EntryList>
            <ParameterRefEntry parameterRef="CCSDSVersion"/>
            <ParameterRefEntry parameterRef="CCSDSType"/>
            <ParameterRefEntry parameterRef="CCSDSSecH"/>
            <ParameterRefEntry parameterRef="CCSDSAPID"/>
            <ParameterRefEntry parameterRef="CCSDSGroupFlags"/>
            <ParameterRefEntry parameterRef="CCSDSSourceSequenceCount"/>
            <ParameterRefEntry parameterRef="CCSDSPacketLength"/>
            <Restriction>
                <ComparisonList>
                    <Comparison value="0" parameterRef="CCSDSVersion"/>
                    <Comparison value="1" parameterRef="CCSDSType"/>
                    <Comparison value="0" parameterRef="CCSDSSecH"/>
                    <Comparison value="3" parameterRef="CCSDSGroupFlags"/>
                </ComparisonList>
            </Restriction>
            <ArgumentRefEntry argumentRef="RelayState1"/>
            <ArgumentRefEntry argumentRef="RelayState2"/>
            <ArgumentRefEntry argumentRef="RelayState3"/>
            <ArgumentRefEntry argumentRef="RelayState4"/>
            <Restriction>
                <Comparison value="179" parameterRef="CCSDSAPID"/>
            </Restriction>
        </EntryList>
    </ProcessedMetaCommandCommandContainer>
</ProcessedMetaCommand>
```

It should be the same as the basic SetRelays command except for the ArgumentAssignments.

The same should be true for the SetRelaysOff command, which is as follows:

```
<ProcessedMetaCommand name="SetAllRelaysOff">
    <ArgumentAssignmentList>
        <ArgumentAssignment argumentValue="Off" argumentName="RelayState1"/>
        <ArgumentAssignment argumentValue="Off" argumentName="RelayState2"/>
        <ArgumentAssignment argumentValue="Off" argumentName="RelayState3"/>
        <ArgumentAssignment argumentValue="Off" argumentName="RelayState4"/>
    </ArgumentAssignmentList>
    <SystemName>HRD</SystemName>
    <ArgumentList>
        <Argument argumentTypeRef="RelayStateType" name="RelayState1"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState2"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState3"/>
        <Argument argumentTypeRef="RelayStateType" name="RelayState4"/>
    </ArgumentList>
    <ProcessedMetaCommandCommandContainer name="SetAllRelaysOffPacket">
        <AncillaryDataSet>
            <AncillaryData name="VCID">0</AncillaryData>
        </AncillaryDataSet>
        <EntryList>
            <ParameterRefEntry parameterRef="CCSDSVersion"/>
            <ParameterRefEntry parameterRef="CCSDSType"/>
            <ParameterRefEntry parameterRef="CCSDSSecH"/>
            <ParameterRefEntry parameterRef="CCSDSAPID"/>
            <ParameterRefEntry parameterRef="CCSDSGroupFlags"/>
            <ParameterRefEntry parameterRef="CCSDSSourceSequenceCount"/>
            <ParameterRefEntry parameterRef="CCSDSPacketLength"/>
            <Restriction>
                <ComparisonList>
                    <Comparison value="0" parameterRef="CCSDSVersion"/>
                    <Comparison value="1" parameterRef="CCSDSType"/>
                    <Comparison value="0" parameterRef="CCSDSSecH"/>
                    <Comparison value="3" parameterRef="CCSDSGroupFlags"/>
                </ComparisonList>
            </Restriction>
            <ArgumentRefEntry argumentRef="RelayState1"/>
            <ArgumentRefEntry argumentRef="RelayState2"/>
            <ArgumentRefEntry argumentRef="RelayState3"/>
            <ArgumentRefEntry argumentRef="RelayState4"/>
            <Restriction>
                <Comparison value="179" parameterRef="CCSDSAPID"/>
            </Restriction>
        </EntryList>
            </ProcessedMetaCommandCommandContainer>
</ProcessedMetaCommand>
```

Further processing could take place to fully resolve all the references and gather all the information into one aggregate object. This may then change to some other format as is necessary.

## 5.7 REFERENCING THAT CROSSES SIDES

Items defined on one 'side' of XTCE (TelemetryMetaData or CommandMetaData) can be referenced to items on the other side. In this case, only the construction is used on the referring side. The values represented by those constructions are not copied.

The exception is ParameterInstanceRefs, which refer to both constructions and values.

## 5.8 REFERENCE SCOPE

### 5.8.1 GENERAL

The scope of items associated with NameReferences is important in a variety of locations in XTCE. The subsections below explain the details.

### 5.8.2 IN SPACESYSTEM HIERARCHIES

SpaceSystem hierarchies have already been discussed in 4.2; however, one area needs further consideration. If an item is defined in a particular SpaceSystem and refers to another item in another SpaceSystem, care should be taken that any NameReferences it has are resolved relative to the second SpaceSystem. This is particularly true for NameReferences that are either unqualified or have a relative path.

For example, if a ParameterType called SizeType is defined in SpaceSystem A, then in SpaceSystem B, Size Parameter, SizeType, and ParameterType are defined. SizeType is then an unqualified NameReference.

Care should be taken to find SpaceSystem B's Size parameter's SizeType in B, not in A by mistake.

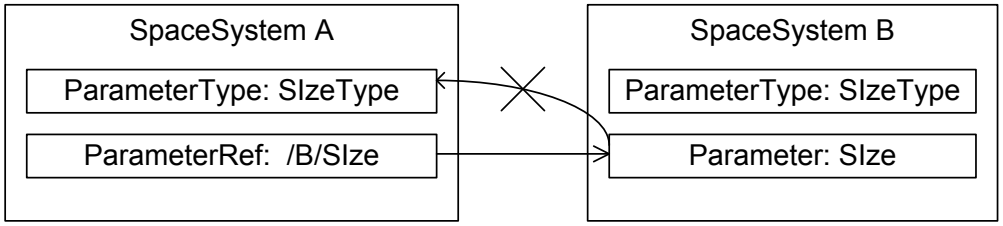



**Figure 5-5: NameReferences Should Not 'Move' Inadvertently through Processing**

Items are associated with the SpaceSystem in which they are defined, not with the SpaceSystem in which they are used.

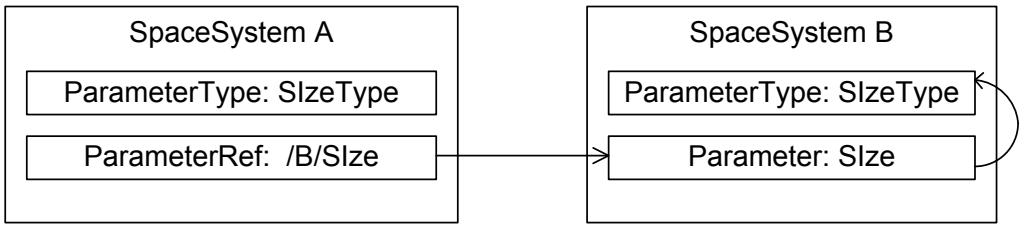



**Figure 5-6: Definitions Are Sticky**

### 5.8.3 IN RESTRICTIONCRITERIA

ParameterInstanceRefs that are unqualified in a container's RestrictionCriteria and have an instance of zero first refer to the entries in that container. If no matching parameter is found, then it refers to a conceptual received telemetry table. It is also legal for the parameter to be a session variable.

ParameterInstanceRefs in RestrictionCriteria cannot refer to items in child containers.

### 5.8.4 IN INCLUDECONDITION

The scope of instances defined in IncludeCondition is similar to RestrictionCriteria. The instance cannot refer 'forward' in the container past the point in the EntryList in which the IncludeCondition has been defined. For example:

```
<xtce:SequenceContainer name="IncludeScope">
    xtce:EntryList>
        <xtce:ParameterRefEntry parameterRef="P0"/>
        <xtce:ParameterRefEntry parameterRef="P1">
            <xtce:IncludeCondition>
                <xtce:ComparisonList>
                    <xtce:Comparison parameterRef="P0" value="1"/>
                    <xtce:Comparison parameterRef="P2" value="1"/>
                </xtce:ComparisonList>
            </xtce:IncludeCondition>
        </xtce:ParameterRefEntry>
        <xtce:ParameterRefEntry parameterRef="P2"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
```

## 5.9 ARRAY PARAMETERS

An ArrayParameterType defines the number of dimensions in an array, and for XTCE1.2, the size of dimension directly. This differs from XTCE 1.2, in which the size of each dimension is defined at the 'point of use' in a container.

ArrayParameterType can refer to any other ParameterType, and a member of an Aggregate can refer to an ArrayParameterType in XTCE 1.2 now.

## 5.10 DEFINING SESSION VARIABLES

Session variables or parameters are parameters with a ParameterType that has no DataEncoding element. They are used to represent items supplied by the processing system. The method of delivery is not specified in XTCE; however, some session variables may be associated in an algorithm, as shown below.

```
<xtce:TelemetryMetaData>
    <xtce:ParameterTypeSet>
        <xtce:IntegerParameterType name="VCIDType">
            <xtce:UnitSet/>
        </xtce:IntegerParameterType>
    </xtce:ParameterTypeSet>
    <xtce:ParameterSet>
        <xtce:Parameter parameterTypeRef="VCIDType" name="VCID">
            <xtce:ParameterProperties dataSource="local"/>
        </xtce:Parameter>
    </xtce:ParameterSet>
</xtce:TelemetryMetaData>
```

In the example above, the VCID Parameter has an IntegerParameterType with no encoding. Since there is no encoding, the VCID Parameter is a session variable.

## 5.11 DEFINING PSEUDO-PARAMETERS

Pseudo-parameters (also known as derived parameters) may be defined using parameters, ParameterTypes, and algorithms. Pseudo-parameters are session parameters (variables) associated with an algorithm used to produce them. MathOperation and CustomAlgorithm are available to describe the implementations; for example:

```
<xtce:TelemetryMetaData>
    <xtce:ParameterTypeSet>
        <xtce:FloatParameterType name="FahrenheitType">
            <xtce:UnitSet>
                <xtce:Unit>Fahrenheit</xtce:Unit>
            </xtce:UnitSet>
            <xtce:FloatDataEncoding encoding="IEEE754_1985" sizeInBits="32"/>
        </xtce:FloatParameterType>
        <xtce:FloatParameterType name="CelsiusType">
            <xtce:UnitSet>
                <xtce:Unit>Celsius</xtce:Unit>
            </xtce:UnitSet>
        </xtce:FloatParameterType>
    </xtce:ParameterTypeSet>
    <xtce:ParameterSet>
        <xtce:Parameter parameterTypeRef="FahrenheitType" name="Fahrenheit"/>
            <xtce:Parameter parameterTypeRef="CelsiusType" name="Celsius">
```

```
            <xtce:ParameterProperties dataSource="derived"/>
        </xtce:Parameter>
    </xtce:ParameterSet>
    <xtce:AlgorithmSet>
        <xtce:MathAlgorithm name="FahrenheitToCelsius">
            <xtce:MathOperation outputParameterRef="Celsius">
                <xtce:ParameterInstanceRefOperand parameterRef="Fahrenheit"/>
                <xtce:ValueOperand>32</xtce:ValueOperand>
                    <xtce:Operator>-</xtce:Operator>
                        <xtce:ValueOperand>5</xtce:ValueOperand>
                <xtce:Operator>*</xtce:Operator>
                        <xtce:ValueOperand>9</xtce:ValueOperand>
                <xtce:Operator>/</xtce:Operator>
                <xtce:TriggerSet>
                    <xtce:OnParameterUpdateTrigger parameterRef="Fahrenheit"/>
                </xtce:TriggerSet>
            </xtce:MathOperation>
        </xtce:MathAlgorithm>
    </xtce:AlgorithmSet>
</xtce:TelemetryMetaData>
```

The example describes how the parameter Celsius is derived from the parameter Fahrenheit. The algorithm used for the conversion is given as a MathOperation. It uses postfix notation, that is, ‘reverse Polish notation’.

The above example specifies the conversion formula for Fahrenheit to Celsius that is given in infix notation as:

$$\text{Celsius} = (\text{Fahrenheit} - 32) * 5/9$$

To convert infix to postfix notation, the Shunting Yard Algorithm is used and produces this result:

$$\text{Celsius} = \text{Fahrenheit } 32 - 5 * 9 /$$

The postfix notation eliminates the need for parentheses and therefore simplifies the XML description.

# 6 TELEMETRY CONTAINER PATTERNS

## 6.1 OVERVIEW

As can be seen from the proceeding sections, there is flexibility in using containers to describe packaging. This section provides some basic guidance for telemetry containers.

## 6.2 TELEMETRY PACKET PATTERNS

### 6.2.1 GENERAL

The patterns outlined here use UML-like class diagrams. The labels are given as '<<stereotypes>>', which describe the XTCE element being diagrammed.

### 6.2.2 BASIC PATTERN

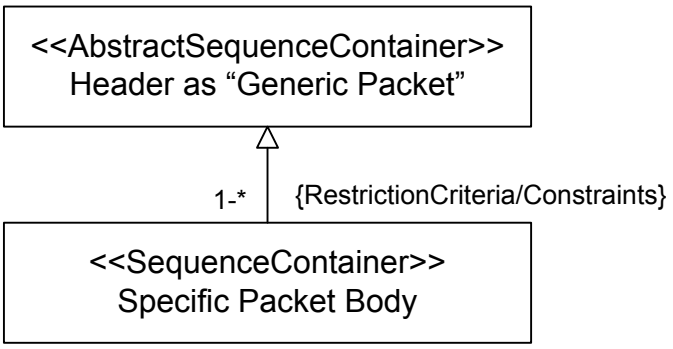



**Figure 6-1: Basic Container Inheritance Pattern**

The basic pattern defines an abstract container for a given format and then derives all telemetry packets from it. For many missions, one level of inheritance is all that is needed to describe every package element in the system.

### 6.2.3 PATTERN WITH AN OPTIONAL SECONDARY HEADER

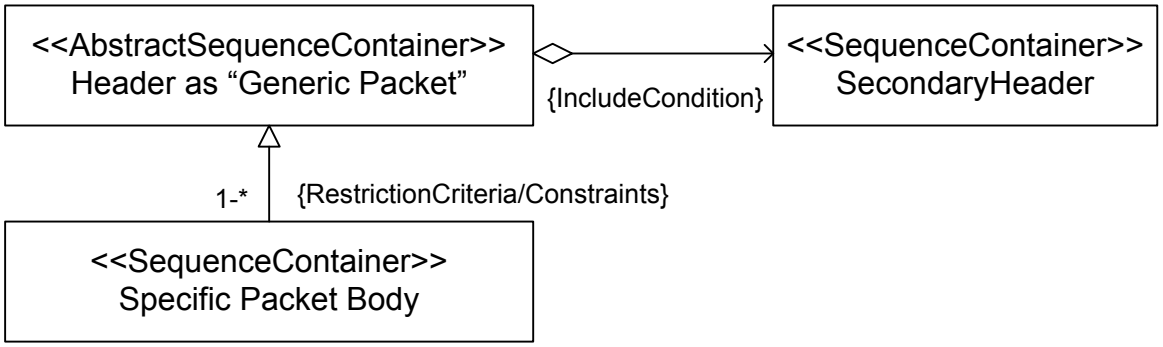



**Figure 6-2: Including a Secondary Header**

Here an optional SecondaryHeader is using a ContainerEntryRef with an IncludeCondition in the Generic Packet's EntryList. A condition would check a flag to indicate if it is present.

It may be easier to simply hard code the secondary header in the packet body containers that have them.

### 6.2.4 A MORE COMPLEX PATTERN

Additional levels of inheritance might be desired. Perhaps each telemetry packet has a common root, or represents a common service. In the pattern below, the telemetry packets have a common container.

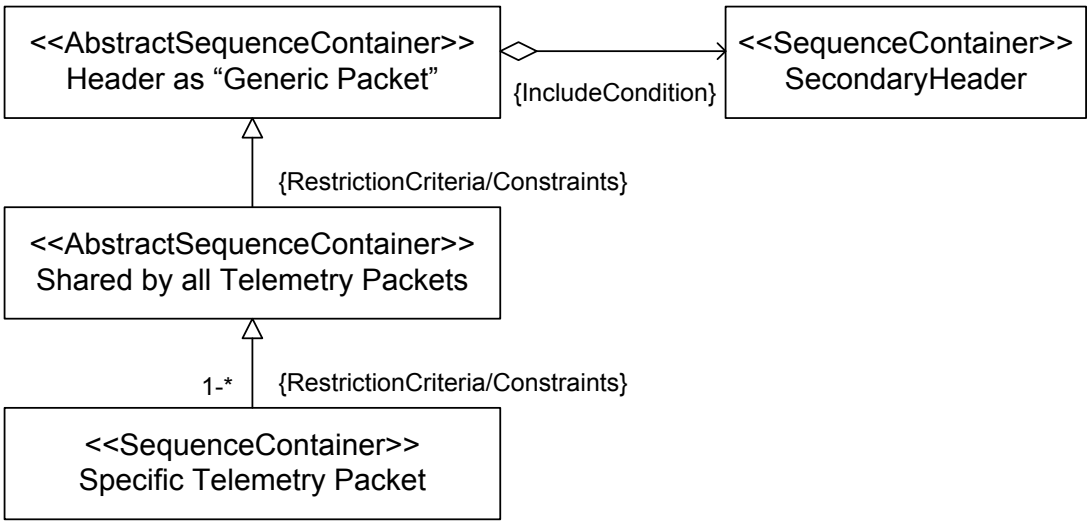



**Figure 6-3: Common Telemetry Root Container**

### 6.2.5 PATTERN WITH DYNAMIC SECONDARYHEADER MATCHING

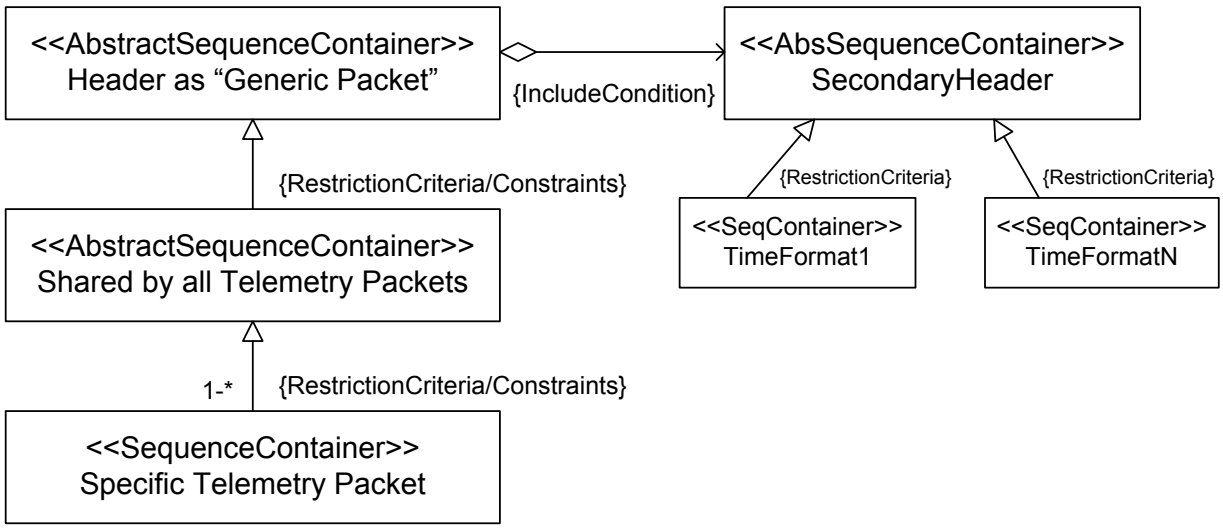



**Figure 6-4: Dynamic Container Matching of Secondary Headers**

Dynamic container matching could be used for the secondary header. In this example, the time stamp format changes depending on local condition.

# 7 COMMANDS AND COMMAND CONTAINER PATTERNS

## 7.1 GENERAL

There are several ways to build commands in XTCE depending on end-user needs and requirements. This section shows three basic patterns for building commands.

## 7.2 COMMAND PATTERNS

### 7.2.1 GENERAL

UML-like class diagrams are again used to show the relationships for both the MetaCommands and their CommandContainers.

### 7.2.2 SIMPLE PATTERN

The simplest form of a command uses individual MetaCommands that define their entire packet in their CommandContainer only (MetaCommand/CommandContainer). The form uses FixedValueEntry to supply hard-coded information such as opcodes. Arguments easily can be added to this simple construction.

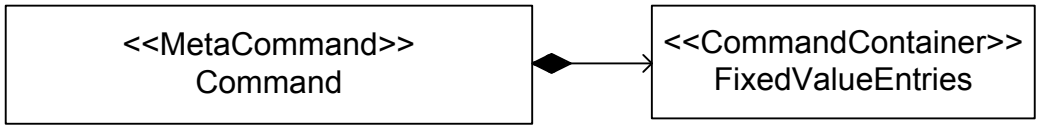



**Figure 7-1: Simple Command Pattern**

### 7.2.3 COMPLEX PATTERN

The next level of complexity starts with an abstract MetaCommand that supplies some common portions of all mission commands in its CommandContainer. This 'generic command' is then derived into concrete commands.

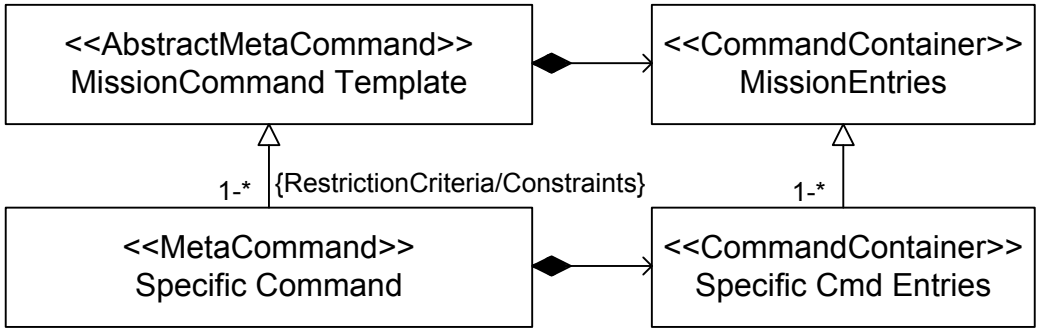



**Figure 7-2: Basic Command Inheritance Pattern**

What is not shown is that MissionEntries may share a common header defined in TelemetryMetaData.

### 7.2.4 GENERIC PATTERN

This version defines a root Mission MetaCommand and associated Mission CommandContainer for a given mission. It derives them from a common set of constructions.

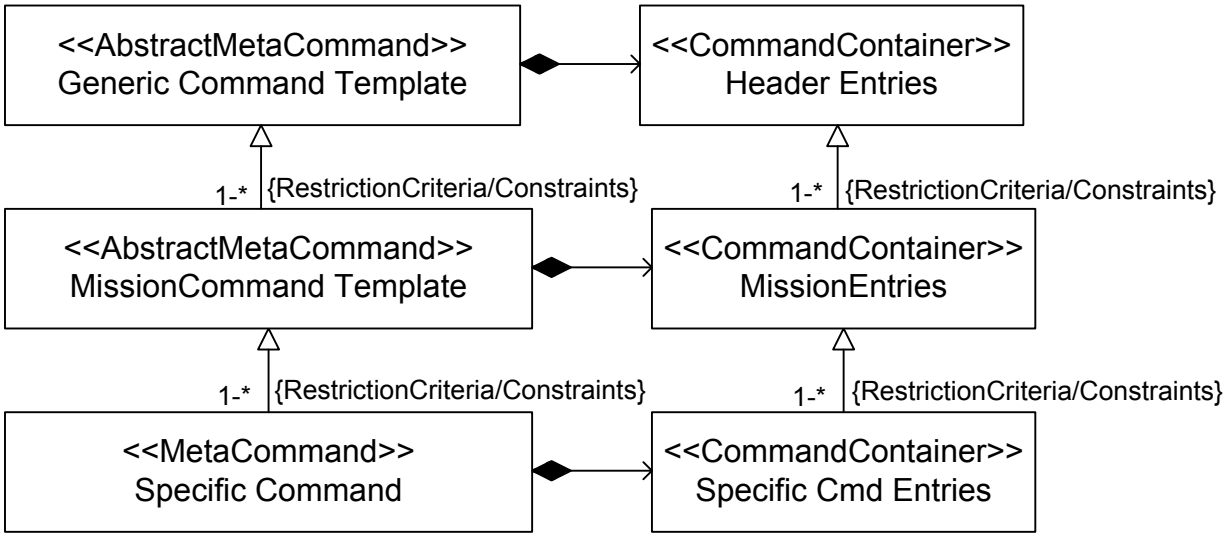



**Figure 7-3: Mission MetaCommand**

In some systems, derived commands may be further defined. These can be added by deriving them from the Specific Command and its CommandContainer.

# 8 SUGGESTIONS FOR CREATING AND PARSING XTCE DOCUMENTS

## 8.1 GENERAL

XTCE documents can be created in a variety of ways: manually with a text editor or tool such as XMLSpy, or programmatically.

The following tools and technologies have proven helpful in this regard:

a) XML Technologies:
   - XPath 1.0 and 2.0: look up nodes in an XML file;
   - XSLT: script-based transformations;
   - XQuery: query a file somewhat similar to SQL query concepts;

b) Low-Level Parsers (programming libraries):
   - SAX: event-driven parsing;
   - DOM: tree-based parsing;

c) Schema-Type Mappings to Programming Languages:
   - XMLSpy or other commercial products: C++, Java, C#, etc.;
   - Eclipse IDE EMF: Java;
   - XMLBeans: Java;
   - JAXB: Java (part of Java distribution);
   - Castor and others.

Schema-type mapping to programming languages deserves additional discussion. These mapping programs (sometimes called XML data binding) read in any XML Schema and then produce a program mapping of the schema types used to create it as classes in the programming language. These classes can then be used to parse or create XML files in the XML Schema language. The mapping created is convenient because artifacts in the programming language match familiar schema types.

## 8.2 JAXB CASE STUDY

### 8.2.1 OVERVIEW

The following subsection briefly looks at a JAXB XTCE mapping. Many of the other XML data-binding tools listed above will produce similar results.

All of these tools work in a similar fashion. They allow one to read in an XML Schema (such as XTCE) and then in a programming language (such as Java) produce a set of classes that represent the simple types, the complex types, the elements, and the attributes in that XML Schema.

One can then read an XML file into these objects created from the classes; this is referred to as 'unmarshalling'. By creating objects from these and supplying the information of interest, one can produce XML files by serializing the objects to XML; this is known as 'marshalling'.

For example, JAXB maps the XML Schema xsd:string type to a java.lang.String or the SpaceSystem in XTCE to a class called SpaceSystemType. The result is a high-level API that varies from something like DOM, which offers a lower level of access to XML files using a generic API.

JAXB is currently included as part of the official Java distribution.

### 8.2.2 XJC

The *xjc* tool is the JAXB supplied to read in the XML Schema and map it into Java classes. JAXB follows some basic rules and maps the XML simple data types to certain included Java classes (i.e., the simple data type string to String). It then implements classes for items that have been defined in the XML Schema. This process is highly configurable to improve the mapping in various ways.

### 8.2.3 XJC CONFIGURATION FILE

A configuration file can be used to manipulate *xjc* in numerous ways. However, it is recommendated to use the default mappings.

### 8.2.4 MARSHALLING AND UNMARSHALLING

The basic usage of JAXB from the programmer standpoint is reading in an existing XML file and binding that into the proper JAXB-generated Java class objects representing the information in the file. For XTCE, this means reading in an XTCE file and returning a set of objects representing the XTCE elements and attributes in the file starting with an object for the SpaceSystem root.

In JAXB, this is called unmarshalling, and the converse is called marshalling. Marshalling produces XML output from the objects that have been created from the classes generated representing the schema (such as XTCE).

Simple examples of these two processes are given below.

### 8.2.5 UNMARSHALLING EXAMPLE

```
import java.io.File;
import java.util.List;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

import org.omg.spec.xtce._20180204.SpaceSystemType;
import org.omg.spec.xtce._20180204.ParameterType;

public class ReadXTCEDoc {

   public static void main(String[] args) {

      try {
         JAXBContext jc =
                           JAXBContext.newInstance("org.omg.spec.xtce._2018
                           0204");
         Unmarshaller unmarshaller = jc.createUnmarshaller();
         Object temp = unmarshaller.unmarshal(new File( args[0] ));
         JAXBElement jaxbElement = ((JAXBElement) temp);
         SpaceSystemType spaceSystem =
                                 (SpaceSystemType)jaxbElement.getValue();
         System.out.println("Read: " + spaceSystem.getName());

         List<Object> params =
                              spaceSystem.
                                       getTelemetryMetaData().
                                       getParameterSet().
                                       getParameterOrParameterRef();
         for (Object p : params) {
            if (p instanceof ParameterType)
                System.out.println("p: " + ((ParameterType)p).getName());
         }

      } catch (JAXBException e) {
         e.printStackTrace();
      }
   }
}
```

As can be seen, the program unmarshalls a pre-existing XTCE XML file as given in the command-line arguments. From there, it simply gets the first SpaceSystem and prints out the name of any Parameters found.

The imports show that the JAXB-generated classes are in org.omg.space.xtce. This was done by the *xjc* program but can be overridden by the user.

### 8.2.6 MARSHALLING EXAMPLE

```
import java.math.BigInteger;
```

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import org.omg.spec.xtce._20180204.IntegerDataEncodingType;
import org.omg.spec.xtce._20180204.NameDescriptionType;
import org.omg.spec.xtce._20180204.ObjectFactory;
import org.omg.spec.xtce._20180204.ParameterSetType;
import org.omg.spec.xtce._20180204.SpaceSystemType;
import org.omg.spec.xtce._20180204.IntegerParameterType;

public class WriteXTCEDoc {
      JAXBElement<SpaceSystemType> spaceRoot;
      JAXBContext jc;
      Marshaller marshaller;
      ObjectFactory factory;

      public WriteXTCEDoc() throws JAXBException {
            jc = JAXBContext.newInstance("org.omg.spec.xtce._20180204");
            marshaller = jc.createMarshaller();

      marshaller.setProperty(
          "com.sun.xml.internal.bind.namespacePrefixMapper",
             new XTCEPrefixMapper());
            marshaller.setProperty(
           Marshaller.JAXB_SCHEMA_LOCATION,"
http://www.omg.org/spec/XTCE/20180204
                     XTCE12.xsd");
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
            new Boolean(true));

            factory = new ObjectFactory();
      }
      public void buildCxSpaceSystem(String name) {
            SpaceSystemType space = factory.createSpaceSystemType();
            space.setName(name);
            spaceRoot = factory.createSpaceSystem(space);
      }
      public void buildCxTelemetry() {
            SpaceSystemType space = spaceRoot.getValue();


      space.setTelemetryMetaData(factory.createTelemetryMetaDataType());
            space.getTelemetryMetaData().setParameterSet(
               factory.createParameterSetType());
            space.getTelemetryMetaData().setParameterTypeSet(
                factory.createParameterTypeSetType());

      }
      public void addParameter(String name, NameDescriptionType type) {
            SpaceSystemType space = spaceRoot.getValue();

            ParameterSetType parameterSet = space.getTelemetryMetaData()
                      .getParameterSet();

            ParameterSetType.Parameter parameter = factory
                       .createParameterSetTypeParameter();
```

```
            parameter.setName(name);
            parameter.setParameterTypeRef(type.getName());

            parameterSet.getParameters().add(parameter);

            space.getTelemetryMetaData().getParameterTypeSet()
                    .getParameterTypes().add(type);
      }

      public void addParameter(String name, String typeName) {
            SpaceSystemType space = spaceRoot.getValue();

            ParameterSetType parameterSet = space.getTelemetryMetaData()
                        .getParameterSet();

            ParameterSetType.Parameter parameter = factory
                    .createParameterSetTypeParameter();

            parameter.setName(name);
            parameter.setParameterTypeRef(typeName);
            parameterSet.getParameters().add(parameter);

      }

      public NameDescriptionType int16(String name) {
            IntegerParameterType i = factory
                .createParameterTypeSetTypeIntegerParameterType();
            i.setName(name);
            IntegerDataEncodingType idenc = factory
                .createIntegerDataEncodingType();

            idenc.setSizeInBits(BigInteger.valueOf(16));
            i.setIntegerDataEncoding(idenc);
            return i;

      }
      public void out() throws JAXBException {
            marshaller.marshal(spaceRoot, System.out);
      }
      public static void main(String[] args) {

            try {
                  WriteXTCEDoc XTCEDoc = new WriteXTCEDoc();

                  // the basic boilerplace - placeholder now
                  XTCEDoc.buildCxSpaceSystem("Test");
                  XTCEDoc.buildCxTelemetry();

                  //build some parameters
                  XTCEDoc.addParameter("P1", XTCEDoc.int16("Int16Type"));
                  XTCEDoc.addParameter("P2", "Int16Type");
                  XTCEDoc.addParameter("P3", "Int16Type");

                  XTCEDoc.out();
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

```
import com.sun.xml.internal.bind.marshaller.NamespacePrefixMapper;

public class XTCEPrefixMapper extends NamespacePrefixMapper {

      @Override
      public String getPreferredPrefix(String namespaceUri,
             String suggestion, boolean requiredPrefix) {
            if ("http://www.w3.org/2001/XMLSchema-
instance".equals(namespaceUri))
                  return "xsi";

            if
("http://www.omg.org/spec/XTCE/20180204".equals(namespaceUri))
                  return "xtce";

            return suggestion;
      }

      @Override
      public String[] getPreDeclaredNamespaceUris2() {
            return new String[] {"xsi", "http://www.w3.org/2001/XMLSchema-
instance"};
      }

}
```

### 8.2.7 MARSHALLER XML OUTPUT EXAMPLE

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xtce:SpaceSystem                                          name="SpaceVehicle"
xsi:schemaLocation="http://www.omg.org/spec/XTCE/20180204       XTCE12.xsd"
xmlns:xtce="http://www.omg.org/spec/XTCE/20180204"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xtce:TelemetryMetaData>
    <xtce:ParameterTypeSet>
      <xtce:IntegerParameterType name="Int16Type">
        <xtce:IntegerDataEncoding sizeInBits="16"/>
      </xtce:IntegerParameterType>
    </xtce:ParameterTypeSet>
    <xtce:ParameterSet>
      <xtce:Parameter parameterTypeRef="Int16Type" name="P1"/>
      <xtce:Parameter parameterTypeRef="Int16Type" name="P2"/>
      <xtce:Parameter parameterTypeRef="Int16Type" name="P3"/>
    </xtce:ParameterSet>
  </xtce:TelemetryMetaData>
</xtce:SpaceSystem>
```

### 8.2.8 OTHER DATA-BINDING ISSUES

Two issues seem to come up consistently with all of the various XML data-binding tools. The first is the Condition element, which has two elements in it with the same name (ParameterInstanceRef). The second is the MetaCommand/CommandContainer, which has both an ArrayParameterRefEntry and ArrayArgumentRefEntry constructed using the same schema type.

The way to solve these problems in terms of reading the information out or creating it is to interact more directly with the XML. So far, the various data-binding tools used have all provided this mechanism in some manner, and these two more difficult cases can be handled.

## 8.3 A BASIC ROUNDTRIP CONVERSION PROCESS

Many new users of XTCE will have a pre-existing source of telemetry and command description information that they wish to map to XTCE. Part of that process is to verify the result by converting the created XTCE file back to its original description format and then comparing the original files to the converted XTCE files. This process is called roundtrip conversion.

These convertors or translators seem to follow a rather basic process that is outlined here and can be tuned for local needs:

a) The parser or software needed to read the information in the original telemetry and command descriptions is acquired.

   – In some cases it may not be possible to acquire such software. If not, one option is to convert the files to XML. Depending on the format, this can be a fairly easy process.

   – Once this process has been completed, the new XML version of the original file can be read into a program like XMLSpy. XMLSpy will derive an XML Schema from it.

b) The original format is taken and its keywords are listed in a column. This can be done using a program such as Excel.

c) A second column is created, and next to each item is placed the XTCE elements and attributes that most correspond in concept to them. There are several suggestions to take into account at this step:

   1) An attempt should be made to map like concepts to like concepts.

   2) AncillaryData should not be overused.

   3) An element should not be ‘misused’ to hold something it was never intended to hold.

   4) Items that simply do not fit anywhere need to be described.

5) Areas that may require a more creative interpretation of an element or attribute need to be marked.

These suggestions are subjective. Typically, the mapping will be direct, although some will require updates (minor, major).

d) A container and command-inheritance patterns should be chosen. The way the format describes telemetry packaging and commanding needs to be considered. Diagrams to map out a basic pattern for telemetry and commands should be used.

e) The mapping is produced. The mapping table and the container and command patterns are used to produce software that will read in the original files and convert them to XTCE. The XTCE files need to be validated.

f) The mapping is then reversed. The XTCE files are brought back to the original format and the results compared to the original.

g) The mapping is iteratively improved.

h) The mapping should be documented. Since it is desirable to share the mapping with others, the mapping table and the container and command inheritance patterns should be captured in documentation for distribution.

# 9 COMPLETE EXAMPLE

## 9.1 OVERVIEW

The following is a full example of a single telemetry packet. To save space, pattern #1 is used from 6.2.2.

## 9.2 TELEMETRY PACKET EXAMPLE

**Table 9-1: Telemetry Packet Example**

| Name: HealthSafety1 | Byte Offset | Bit Length | Mnemonic | Comments |
|---|---|---|---|---|
| ***Channel*** | N/A | 8 | Channel | 0x05—packets are associated with channel. The channel is supplied by the system and not held in the packet itself. It is not part of header. |
| | | | | |
| **HEADER** | | | | 3 bytes |
| ***ID*** | 0 | 6 | ID | 0x18 (24) |
| ***Type*** | | 1 | Type | 0—command, 1—telemetry |
| ***SecondaryHeader*** | | 1 | SecH | none, 1—yes |
| ***Length*** | 2 | 16 | Length | Bytes—does not include header<br>– no secondary = 8<br>– w/secondary = 14 |
| **Secondary Header** | | | | 6 bytes, secs/millis |
| ***timestamp(seconds)*** | 4 | 32 | Seconds | Combined with below to form MissionTime - |
| ***timestamp(milliseconds)*** | 8 | 16 | MilliSeconds | Part of MissionTime |
| **battery module temp** | 10 | 16 | PBATMTMP | Convert to Float—12 LSB's raw A/D read value<br>deg C = 5th order polynomial, coefficients:<br>(-7459.23273708, 8.23643519148, -3.02185061876e-3, 2.33422429056e-7,<br>5.67189556173e-11) |
| **wheel timer flag** | 12 | 16 | PSWHLTIMFLG | 0—default<br>0 = Timer OFF<br>1 = Timer ON<br>2 = Timer Completed |
| **PseudoTelemetry** | | | | **Also known as derived; not part of header but part of parameter set** |
| **MissionTime** | N/A | 48 | MissionTime | Combined STIME and MSTIME |

### 9.2.1 XTCE REPRESENTATION

#### 9.2.1.1 Overview

In this example, assume each packet is mapped to a singular ID.

#### 9.2.1.2 The Parameters

```
<xtce:Parameter name="SecH" parameterTypeRef="SecHType"/>
<xtce:Parameter name="Type" parameterTypeRef="TypeType"/>
<xtce:Parameter name="ID" parameterTypeRef="IDType"/>
<xtce:Parameter name="Length" parameterTypeRef="LengthType"/>
<xtce:Parameter name="Seconds" parameterTypeRef="SecondsType"/>
<xtce:Parameter name="MilliSeconds" parameterTypeRef="MilliSecondsType"/>
<xtce:Parameter name="PBATMTEMP" parameterTypeRef="PBATMTEMPType"/>
<xtce:Parameter name="PSWHLTIMFLG" parameterTypeRef="PSWHLTIMFLGType"/>
<xtce:Parameter name="MissionTime" parameterTypeRef="MissionTimeType">
    <xtce:ParameterProperties dataSource="derived"/>
</xtce:Parameter>
```

The MissionTime Parameter is an AbsoluteTimeParameterType composed of the two telemetry time values: Seconds and MilliSeconds. It is a derived Parameter.

#### 9.2.1.3 The ParameterTypes

```
<xtce:ParameterTypeSet>
    <xtce:IntegerParameterType signed="false" name="ChannelType">
            <xtce:UnitSet/>
    </xtce:IntegerParameterType>
    <xtce:IntegerParameterType signed="false" name="IDType">
            <xtce:UnitSet/>
            <xtce:IntegerDataEncoding sizeInBits="8"/>
    </xtce:IntegerParameterType>
    <xtce:IntegerParameterType signed="false" name="SecHType">
            <xtce:UnitSet/>
            <xtce:IntegerDataEncoding sizeInBits="1"/>
    </xtce:IntegerParameterType>
    <xtce:IntegerParameterType signed="false" name="TypeType">
            <xtce:UnitSet/>
            <xtce:IntegerDataEncoding sizeInBits="1"/>
    </xtce:IntegerParameterType>
    <xtce:IntegerParameterType signed="false" name="LengthType">
            <xtce:UnitSet/>
            <xtce:IntegerDataEncoding sizeInBits="16"/>
    </xtce:IntegerParameterType>
    <xtce:EnumeratedParameterType name="PSWHLTIMFLGType"
initialValue="TIMER_OFF">
            <xtce:UnitSet/>
            <xtce:IntegerDataEncoding sizeInBits="16"/>
            <xtce:EnumerationList>
```

```
                <xtce:Enumeration label="TIMER_OFF" value="0"/>
                <xtce:Enumeration label="TIMER_ON" value="1"/>
                <xtce:Enumeration label="TIMER_COMPLETED" value="2"/>
            </xtce:EnumerationList>
        </xtce:EnumeratedParameterType>
        <xtce:FloatParameterType sizeInBits="64" name="PBATMTEMPType">
            <xtce:UnitSet>
                <xtce:Unit description="Bq">units:Becquerel</xtce:Unit>
            </xtce:UnitSet>
        <xtce:IntegerDataEncoding sizeInBits="16" encoding="twosComplement">
            <xtce:DefaultCalibrator>
                <xtce:PolynomialCalibrator>
                    <xtce:Term coefficient="-7459.23273708" exponent="0"/>
                    <xtce:Term coefficient="8.23643519148" exponent="1"/>
                    <xtce:Term coefficient="-3.02185061876e-3" exponent="2"/>
                    <xtce:Term coefficient="2.33422429056e-7" exponent="3"/>
                    <xtce:Term coefficient="5.67189556173e-11" exponent="4"/>
                </xtce:PolynomialCalibrator>
            </xtce:DefaultCalibrator>
        </xtce:IntegerDataEncoding>
        </xtce:FloatParameterType>
        <xtce:AbsoluteTimeParameterType name="MissionTimeType">
            <xtce:ReferenceTime>
                <xtce:OffsetFrom parameterRef="Seconds"/>
            </xtce:ReferenceTime>
        </xtce:AbsoluteTimeParameterType>
        <xtce:AbsoluteTimeParameterType name="SecondsType">
            <xtce:Encoding units="seconds">
                <xtce:IntegerDataEncoding sizeInBits="32"/>
            </xtce:Encoding>
            <xtce:ReferenceTime>
                <xtce:OffsetFrom parameterRef="MilliSeconds"/>
            </xtce:ReferenceTime>
        </xtce:AbsoluteTimeParameterType>
        <xtce:AbsoluteTimeParameterType name="MilliSecondsType">
            <xtce:Encoding scale="0.001" units="seconds">
                <xtce:IntegerDataEncoding sizeInBits="16"/>
            </xtce:Encoding>
            <xtce:ReferenceTime>
                <xtce:Epoch>TAI</xtce:Epoch>
            </xtce:ReferenceTime>
        </xtce:AbsoluteTimeParameterType>
```

The header-based ParameterTypes are IntegerParameterTypes with unsigned IntegerDataEncodings. These are either flags or small integer values like ID or Length. The battery temperature-related parameter is a ‘count’ to a float polynomial conversion. The calibration produces a 64-bit or double-precision type. The timer-related value is coded as an enumeration. The remaining telemetry items are time related. The two time types, SecondsType and MilliSecondsType, are in the SecondaryHeader and are combined to create a pseudo-telemetry type called MissionTime.

#### 9.2.1.4 The Containers

```
<xtce:SequenceContainer abstract="true" name="Header">
    <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="ID"/>
            <xtce:ParameterRefEntry parameterRef="SecH"/>
            <xtce:ParameterRefEntry parameterRef="Type"/>
            <xtce:ParameterRefEntry parameterRef="Length"/>
            <xtce:ContainerRefEntry containerRef="SecondaryHeader">
                    <xtce:IncludeCondition>
                            <xtce:Comparison parameterRef="SecH" value="1"/>
                    </xtce:IncludeCondition>
            </xtce:ContainerRefEntry>
    </xtce:EntryList>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="SecondaryHeader">
    <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="Seconds"/>
            <xtce:ParameterRefEntry parameterRef="MilliSeconds"/>
    </xtce:EntryList>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="HealthSafety1">
    <xtce:EntryList>
            <xtce:ParameterRefEntry parameterRef="PBATMTEMP"/>
            <xtce:ParameterRefEntry parameterRef="PSWHLTIMFLG"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="Header">
            <xtce:RestrictionCriteria>
                    <xtce:ComparisonList>
                            <xtce:Comparison value="1" parameterRef="Type"/>
                            <xtce:Comparison value="24" parameterRef="ID"/>
                    </xtce:ComparisonList>
            </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
</xtce:SequenceContainer>
```

The above construction describes the HealthSafety1 packet. It derives from a Header that itself optionally includes the SecondaryHeader that has the time stamp. The conditions supplied indicate for HealthSafety1 to be present in the incoming byte stream; it must be telemetry (Type=1) and its ID must be 24. The length of the packet is determined using a discrete lookup that returns a different value depending on whether the SecondaryHeader is included. Most telemetry systems time stamp their packets.

#### 9.2.1.5 Entire Example

The entire example is in annex A.

## 9.3 COMMAND EXAMPLE

### 9.3.1 GENERAL

**Table 9-2: Command and Command Packet Example**

| | | | |
|---|---|---|---|
| **Command Name** | Wheel Timer | | |
| **Mnemonic** | PWHTMR | | |
| **ID** | 0x100 | **CHANNEL** | 9 |
| **Op Code** | 0x1e | | |
| **Data Field Length** | 4 bytes | | |
| **Packet Format** | [Header] 1eCS aaaa | **Packet Name** | PWHTMRPacket |
| **Argument Fields** | | | |
| **Offset from Header** | **Name** | **Data Type** | **Data Range** |
| 2 | TimerStartStop | Unsigned 16-bit integer, enumeration | 0 = stops the wheel SW timer (STOP)<br>1 = starts the wheel SW timer (START) |
| **Parameter Fields** | | | |
| **Offset from Header** | **Name** | **Data Type** | **Data Range** |
| **0** | OpCode | Unsigned 8-bit integer | 0x1e |
| **1** | Checksum | Unsigned 8-bit integer | 0x00-0xff |
| **Operation** | This command starts or stops the SW timer that turns on the reaction wheels at wheel_timer_alarm seconds. | | |
| **Criticality** | Yes. | | |
| **Telemetry Verification** | Command counter gets incremented.<br>***The wheel_timer_flag in the slow HealthSafety1*** will be set to data field (a). | | |
| **Error Conditions** | None reported.<br>Return value: 0 if command did not execute; 0x0055 otherwise. | | |

### 9.3.2 XTCE REPRESENTATION

#### 9.3.2.1 Overview

In the following example, it is assumed the ID is unique for each command packet.

#### 9.3.2.2 The Parameters

```
<xtce:Parameter name="CommandCounter" parameterTypeRef="CommandCounterType">
    <xtce:ParameterProperties dataSource="local"/>
</xtce:Parameter>
<xtce:Parameter name="CheckSum" parameterTypeRef="CheckSumType">
    <xtce:ParameterProperties dataSource="derived"/>
</xtce:Parameter>
<xtce:Parameter name="CommandReturn" parameterTypeRef="CommandReturnType">
    <xtce:ParameterProperties dataSource="local"/>
</xtce:Parameter>
```

The CheckSum is calculated by the system and inserted into the EntryList. The CommandCounter is maintained by the system. The parameters associated with the Header remain defined in the telemetry area.

#### 9.3.2.3 The ParameterTypes

```
<xtce:IntegerParameterType name="CommandReturnType" signed="false">
    <xtce:UnitSet/>
</xtce:IntegerParameterType>
<xtce:IntegerParameterType name="CommandCounterType" signed="false">
    <xtce:UnitSet/>
</xtce:IntegerParameterType>
<xtce:IntegerParameterType name="CheckSumType" signed="false">
    <xtce:UnitSet/>
        <xtce:IntegerDataEncoding/>
</xtce:IntegerParameterType>
```

The default IntegerDataEncoding is 8 bits.

#### 9.3.2.4 The ArgumentTypes

```
<xtce:EnumeratedArgumentType name="TimerStartStopType">
    <xtce:UnitSet/>
    <xtce:IntegerDataEncoding sizeInBits="16"/>
    <xtce:EnumerationList>
            <xtce:Enumeration label="TIMER_STOP" value="0"/>
            <xtce:Enumeration label="TIMER_START" value="1"/>
    </xtce:EnumerationList>
</xtce:EnumeratedArgumentType>
```

The argument is very similar to its telemetry cousin.

### 9.3.2.5 MetaCommand

```
<xtce:MetaCommand name="PWHTMR">
    <xtce:ArgumentList>
            <xtce:Argument name="TimerStartStop"
argumentTypeRef="TimerStartStopType"/>
    </xtce:ArgumentList>
    <xtce:CommandContainer name="PWHTMRPacket">
            <xtce:BinaryEncoding>
                    <xtce:SizeInBits>
                            <xtce:FixedValue>32</xtce:FixedValue>
                    </xtce:SizeInBits>
            </xtce:BinaryEncoding>
            <xtce:EntryList>
                    <xtce:FixedValueEntry binaryValue="1e"/>
                    <xtce:ParameterRefEntry parameterRef="CheckSum"/>
                    <xtce:ArgumentRefEntry argumentRef="TimerStartStop"/>
            </xtce:EntryList>
            <xtce:BaseContainer containerRef="Header">
                    <xtce:RestrictionCriteria>
                            <xtce:ComparisonList>
                                    <xtce:Comparison parameterRef="ID" value="256"/>
                                    <xtce:Comparison parameterRef="Type" value="0"/>
                                    <xtce:Comparison parameterRef="SecH" value="0"/>
                            </xtce:ComparisonList>
                    </xtce:RestrictionCriteria>
            </xtce:BaseContainer>
    </xtce:CommandContainer>
        <!-- Verifier and others shown below -->
</xtce:MetaCommand>
```

The MetaCommand (command description) has one argument called opcode. The opcode is supplied as a FixedEntryValue, and the rest are ParameterRefEntries.

### 9.3.2.6 Significance

```
<xtce:DefaultSignificance consequenceLevel="critical"/>
```

The command is marked as ‘critical’. This matches the description that uses the term in the table: ‘Criticality—yes’. One interpretation is that the mission must be in a ‘critical’ state in order for this command to be sent.

### 9.3.2.7 Verifier—Command Complete Test

```
<xtce:CompleteVerifier>
    <xtce:ContainerRef containerRef="HealthSafety1"/>
    <xtce:CheckWindow timeToStopChecking="PT10M"/>
    <xtce:ReturnParmRef parameterRef="CommandReturn"/>
</xtce:CompleteVerifier>
```

The command CompleteVerifier starts its check for 10 minutes. The verifier is 'kicked off' by the receipt of the telemetry packet HealthSafety1. The ReturnParmRef is defined as a local parameter supplied by the system.

#### 9.3.2.8 Verifier—Failed Verifier Test

```
<xtce:FailedVerifier>
    <xtce:Comparison parameterRef="CommandReturn" value="0"/>
    <xtce:CheckWindow timeToStopChecking="PT10M"/>
</xtce:FailedVerifier>
```

The failed verifier tests the value of CommandReturn. If zero, the command has failed.

#### 9.3.2.9 ParameterToSet

```
<xtce:ParameterToSet parameterRef="CommandReturn" setOnVerification="release">
    <xtce:NewValue>0x00</xtce:NewValue>
</xtce:ParameterToSet>
```

This code initializes the CommandReturn value before the command is sent.

```
<xtce:ParameterToSet parameterRef="CommandReturn">
    <xtce:NewValue>0x55</xtce:NewValue>
</xtce:ParameterToSet>
```

Here, upon VerifierComplete, the CommandReturn is updated with the success value 0x55.

#### 9.3.2.10 Full Example

The complete example is given in annex A.

# ANNEX A

# FULL EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2011 rel. 3 sp1 (x64) (http://www.altova.com)-->
<xtce:SpaceSystem name="SpaceVehicle"
xsi:schemaLocation="http://www.omg.org/spec/XTCE/20180204 XTCE12.xsd"
xmlns:xtce="http://www.omg.org/spec/XTCE/20180204"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
	<xtce:TelemetryMetaData>
		<xtce:ParameterTypeSet>
			<xtce:IntegerParameterType signed="false" name="IDType">
				<xtce:UnitSet/>
				<xtce:IntegerDataEncoding sizeInBits="8"/>
			</xtce:IntegerParameterType>
			<xtce:IntegerParameterType signed="false" name="SecHType">
				<xtce:UnitSet/>
				<xtce:IntegerDataEncoding sizeInBits="1"/>
			</xtce:IntegerParameterType>
			<xtce:IntegerParameterType signed="false" name="TypeType">
				<xtce:UnitSet/>
				<xtce:IntegerDataEncoding sizeInBits="1"/>
			</xtce:IntegerParameterType>
			<xtce:IntegerParameterType signed="false" name="LengthType">
				<xtce:UnitSet/>
				<xtce:IntegerDataEncoding sizeInBits="16"/>
			</xtce:IntegerParameterType>
			<xtce:EnumeratedParameterType name="PSWHLTIMFLGType"
initialValue="TIMER_OFF">
				<xtce:UnitSet/>
				<xtce:IntegerDataEncoding sizeInBits="16"/>
				<xtce:EnumerationList>
					<xtce:Enumeration label="TIMER OFF" value="0"/>
					<xtce:Enumeration label="TIMER ON" value="1"/>
					<xtce:Enumeration label="TIMER_COMPLETED" value="2"/>
				</xtce:EnumerationList>
			</xtce:EnumeratedParameterType>
			<xtce:FloatParameterType sizeInBits="64" name="PBATMTEMPType">
				<xtce:UnitSet>
					<xtce:Unit description="Bq">units:Becquerel</xtce:Unit>
				</xtce:UnitSet>
				<xtce:IntegerDataEncoding sizeInBits="16"
encoding="twosComplement">
					<xtce:DefaultCalibrator>
						<xtce:PolynomialCalibrator>
							<xtce:Term coefficient="-7459.23273708"
exponent="0"/>
							<xtce:Term coefficient="8.23643519148"
exponent="1"/>
							<xtce:Term coefficient="-3.02185061876e-
3" exponent="2"/>
							<xtce:Term coefficient="2.33422429056e-7"
exponent="3"/>
							<xtce:Term coefficient="5.67189556173e-
11" exponent="4"/>
						</xtce:PolynomialCalibrator>
					</xtce:DefaultCalibrator>
				</xtce:IntegerDataEncoding>
			</xtce:FloatParameterType>
			<xtce:AbsoluteTimeParameterType name="MissionTimeType">
				<xtce:ReferenceTime>
					<xtce:OffsetFrom parameterRef="Seconds"/>
				</xtce:ReferenceTime>
			</xtce:AbsoluteTimeParameterType>
			<xtce:AbsoluteTimeParameterType name="SecondsType">
				<xtce:Encoding units="seconds">
					<xtce:IntegerDataEncoding sizeInBits="32"/>
				</xtce:Encoding>
				<xtce:ReferenceTime>
					<xtce:OffsetFrom parameterRef="MilliSeconds"/>
				</xtce:ReferenceTime>
			</xtce:AbsoluteTimeParameterType>
```

```
                        <xtce:AbsoluteTimeParameterType name="MilliSecondsType">
                                <xtce:Encoding scale="0.001" units="seconds">
                                        <xtce:IntegerDataEncoding sizeInBits="16"/>
                                </xtce:Encoding>
                                <xtce:ReferenceTime>
                                        <xtce:Epoch>TAI</xtce:Epoch>
                                </xtce:ReferenceTime>
                        </xtce:AbsoluteTimeParameterType>
                </xtce:ParameterTypeSet>
                <xtce:ParameterSet>
                        <xtce:Parameter name="SecH" parameterTypeRef="SecHType"/>
                        <xtce:Parameter name="Type" parameterTypeRef="TypeType"/>
                        <xtce:Parameter name="ID" parameterTypeRef="IDType"/>
                        <xtce:Parameter name="Length" parameterTypeRef="LengthType"/>
                        <xtce:Parameter name="Seconds" parameterTypeRef="SecondsType"/>
                        <xtce:Parameter name="MilliSeconds"
parameterTypeRef="MilliSecondsType"/>
                        <xtce:Parameter name="PBATMTEMP" parameterTypeRef="PBATMTEMPType"/>
                        <xtce:Parameter name="PSWHLTIMFLG" parameterTypeRef="PSWHLTIMFLGType"/>
                        <xtce:Parameter name="MissionTime" parameterTypeRef="MissionTimeType">
                                <xtce:ParameterProperties dataSource="derived"/>
                        </xtce:Parameter>
                </xtce:ParameterSet>
                <xtce:ContainerSet>
                        <xtce:SequenceContainer abstract="true" name="Header">
                                <xtce:EntryList>
                                        <xtce:ParameterRefEntry parameterRef="ID"/>
                                        <xtce:ParameterRefEntry parameterRef="SecH"/>
                                        <xtce:ParameterRefEntry parameterRef="Type"/>
                                        <xtce:ParameterRefEntry parameterRef="Length"/>
                                        <xtce:ContainerRefEntry containerRef="SecondaryHeader">
                                                <xtce:IncludeCondition>
                                                        <xtce:Comparison parameterRef="SecH"
value="1"/>
                                                </xtce:IncludeCondition>
                                        </xtce:ContainerRefEntry>
                                </xtce:EntryList>
                        </xtce:SequenceContainer>
                        <xtce:SequenceContainer name="SecondaryHeader">
                                <xtce:EntryList>
                                        <xtce:ParameterRefEntry parameterRef="Seconds"/>
                                        <xtce:ParameterRefEntry parameterRef="MilliSeconds"/>
                                </xtce:EntryList>
                        </xtce:SequenceContainer>
                        <xtce:SequenceContainer name="HealthSafety1">
                                <xtce:EntryList>
                                        <xtce:ParameterRefEntry parameterRef="PBATMTEMP"/>
                                        <xtce:ParameterRefEntry parameterRef="PSWHLTIMFLG"/>
                                </xtce:EntryList>
                                <xtce:BaseContainer containerRef="Header">
                                        <xtce:RestrictionCriteria>
                                                <xtce:ComparisonList>
                                                        <xtce:Comparison value="1"
parameterRef="Type"/>
                                                        <xtce:Comparison value="24"
parameterRef="ID"/>
                                                </xtce:ComparisonList>
                                        </xtce:RestrictionCriteria>
                                </xtce:BaseContainer>
                        </xtce:SequenceContainer>
                </xtce:ContainerSet>
        </xtce:TelemetryMetaData>
        <xtce:CommandMetaData>
                <xtce:ParameterTypeSet>
                        <xtce:IntegerParameterType name="CommandReturnType" signed="false">
                                <xtce:UnitSet/>
                        </xtce:IntegerParameterType>
                        <xtce:IntegerParameterType name="CommandCounterType" signed="false">
                                <xtce:UnitSet/>
                        </xtce:IntegerParameterType>
                        <xtce:IntegerParameterType name="CheckSumType" signed="false">
                                <xtce:UnitSet/>
                                <xtce:IntegerDataEncoding/>
                        </xtce:IntegerParameterType>
                </xtce:ParameterTypeSet>
```

```
                <xtce:ParameterSet>
                        <xtce:Parameter name="CommandCounter"
parameterTypeRef="CommandCounterType">
                                <xtce:ParameterProperties dataSource="local"/>
                        </xtce:Parameter>
                        <xtce:Parameter name="CheckSum" parameterTypeRef="CheckSumType">
                                <xtce:ParameterProperties dataSource="derived"/>
                        </xtce:Parameter>
                        <xtce:Parameter name="CommandReturn"
parameterTypeRef="CommandReturnType">
                                <xtce:ParameterProperties dataSource="local"/>
                        </xtce:Parameter>
                </xtce:ParameterSet>
                <xtce:ArgumentTypeSet>
                        <xtce:EnumeratedArgumentType name="TimerStartStopType">
                                <xtce:UnitSet/>
                                <xtce:IntegerDataEncoding sizeInBits="16"/>
                                <xtce:EnumerationList>
                                        <xtce:Enumeration label="TIMER_STOP" value="0"/>
                                        <xtce:Enumeration label="TIMER_START" value="1"/>
                                </xtce:EnumerationList>
                        </xtce:EnumeratedArgumentType>
                </xtce:ArgumentTypeSet>
                <xtce:MetaCommandSet>
                        <xtce:MetaCommand name="PWHTMR">
                                <xtce:ArgumentList>
                                        <xtce:Argument name="TimerStartStop"
argumentTypeRef="TimerStartStopType"/>
                                </xtce:ArgumentList>
                                <xtce:CommandContainer name="PWHTMRPacket">
                                        <xtce:BinaryEncoding>
                                                <xtce:SizeInBits>
                                                        <xtce:FixedValue>32</xtce:FixedValue>
                                                </xtce:SizeInBits>
                                        </xtce:BinaryEncoding>
                                        <xtce:EntryList>
                                                <xtce:FixedValueEntry binaryValue="1e"
sizeInBits="8"/>
                                                <xtce:ParameterRefEntry
parameterRef="CheckSum"/>
                                                <xtce:ArgumentRefEntry
argumentRef="TimerStartStop"/>
                                        </xtce:EntryList>
                                        <xtce:BaseContainer containerRef="Header">
                                                <xtce:RestrictionCriteria>
                                                        <xtce:ComparisonList>
                                                                <xtce:Comparison
parameterRef="ID" value="256"/>
                                                                <xtce:Comparison
parameterRef="Type" value="0"/>
                                                                <xtce:Comparison
parameterRef="SecH" value="0"/>
                                                        </xtce:ComparisonList>
                                                </xtce:RestrictionCriteria>
                                        </xtce:BaseContainer>
                                </xtce:CommandContainer>
                                <xtce:DefaultSignificance consequenceLevel="critical"/>
                                <xtce:VerifierSet>
                                        <xtce:CompleteVerifier>
                                                <xtce:ContainerRef
containerRef="HealthSafety1"/>
                                                <xtce:CheckWindow timeToStopChecking="PT10M"/>
                                        </xtce:CompleteVerifier>
                                        <xtce:FailedVerifier>
                                                <xtce:Comparison parameterRef="CommandReturn"
value="0"/>
                                                <xtce:CheckWindow timeToStopChecking="PT10M"/>
                                        </xtce:FailedVerifier>
                                </xtce:VerifierSet>
                                <xtce:ParameterToSetList>
                                        <xtce:ParameterToSet parameterRef="CommandReturn"
setOnVerification="release">
                                                <xtce:NewValue>0x00</xtce:NewValue>
                                        </xtce:ParameterToSet>
                                        <xtce:ParameterToSet parameterRef="CommandReturn">
```

```
                                        <xtce:NewValue>0x55</xtce:NewValue>
                                    </xtce:ParameterToSet>
                                </xtce:ParameterToSetList>
                            </xtce:MetaCommand>
                        </xtce:MetaCommandSet>
                    </xtce:CommandMetaData>
</xtce:SpaceSystem>
```

# ANNEX B

# KEYS

The following keys properly enforce XTCE1.2's @name uniqueness policy. These have been updated from XTCE 1.1's version, which in many cases were broken.

```
...
<element name="SpaceSystem" type="xtce:SpaceSystemType" nillable="true">
		<annotation>
			<documentation xml:lang="en">The top-level SpaceSystem is the root
element for the set of metadata necessary to monitor and command a space device, such as a
satellite.  A SpaceSystem defines a namespace.  Metadata areas include:  packets/minor frames
layout, telemetry, calibration, alarm, algorithms, streams and commands.  A SpaceSystem may
have child SpaceSystems, forming a SpaceSystem tree. See SpaceSystemType.</documentation>
		</annotation>
		<key name="parameterNameKey">
			<annotation>
				<documentation xml:lang="en">This key ensures a unique
parameter name at the system level.</documentation>
			</annotation>
			<selector
xpath="xtce:TelemetryMetaData/xtce:ParameterSet/xtce:Parameter|xtce:CommandMetaData/xtce:Para
meterSet/xtce:Parameter"/>
			<field xpath="@name"/>
		</key>
		<key name="parameterTypeNameKey">
			<annotation>
				<documentation xml:lang="en">This key ensures a unique
parameter type name at the system level.</documentation>
			</annotation>
			<selector
xpath="xtce:TelemetryMetaData/xtce:ParameterTypeSet/*|xtce:CommandMetaData/xtce:ParameterType
Set/*"/>
			<field xpath="@name"/>
		</key>
		<key name="metaCommandNameKey">
			<annotation>
				<documentation xml:lang="en">This key ensures a unique
metaCommand name at the system level.</documentation>
			</annotation>
			<selector
xpath="xtce:CommandMetaData/xtce:MetaCommandSet/xtce:MetaCommand"/>
			<field xpath="@name"/>
		</key>
		<key name="algorithmNameKey">
			<annotation>
				<documentation xml:lang="en">This key ensures a unique
algorithm name at the system level.</documentation>
			</annotation>
			<selector
xpath="xtce:TelemetryMetaData/xtce:AlgorithmSet/*|xtce:CommandMetaData/xtce:AlgorithmSet/*"/>
			<field xpath="@name"/>
		</key>
		<key name="streamNameKey">
			<annotation>
				<documentation xml:lang="en">This key ensures a unique stream
name at the system level.</documentation>
			</annotation>
			<selector
xpath="xtce:TelemetryMetaData/xtce:StreamSet/*|xtce:CommandMetaData/xtce:StreamSet/*"/>
			<field xpath="@name"/>
		</key>
		<key name="serviceNameKey">
			<annotation>
				<documentation xml:lang="en">This key ensures a unique service
name at the system level.</documentation>
			</annotation>
			<selector xpath="xtce:ServiceSet/*"/>
			<field xpath="@name"/>
```

```
                </key>
                <key name="containerNameKey">
                        <annotation>
                                <documentation xml:lang="en">This key ensures a container
stream name at the system level.</documentation>
                        </annotation>
                        <selector
xpath="xtce:TelemetryMetaData/xtce:ContainerSet/*|xtce:CommandMetaData/xtce:CommandContainerS
et/*"/>
                        <field xpath="@name"/>
                </key>
                <key name="messageNameKey">
                        <selector xpath="xtce:TelemetryMetaData/xtce:MessageSet/*"/>
                        <field xpath="@name"/>
                </key>
                <key name="argumentTypeNameKey">
                        <annotation>
                                <documentation xml:lang="en">This key ensures a unique argument
type name at the system level.</documentation>
                        </annotation>
                        <selector
xpath="xtce:CommandMetaData/xtce:ArgumentTypeSet/*|xtce:CommandMetaData/xtce:ParameterTypeSet
/*"/>
                        <field xpath="@name"/>
                </key>
                <key name="blockMetaCommandNameKey">
                        <annotation>
                                <documentation xml:lang="en">This key ensures a unique
BlockMetaCommand name at the system level.</documentation>
                        </annotation>
                        <selector
xpath="xtce:CommandMetaData/xtce:MetaCommandSet/xtce:BlockMetaCommand"/>
                        <field xpath="@name"/>
                </key>


        </element>
```

There is a key in another location as well:

```
<element name="MetaCommand" type="xtce:MetaCommandType">
                                <annotation>
                                        <documentation xml:lang="en">All atomic commands to be
sent on this mission are listed here.  In addition this area has verification and validation
information.</documentation>
                                </annotation>
                                <key name="ArgumentNameKey">
                                        <selector xpath="xtce:ArgumentList/*"/>
                                        <field xpath="@name"/>
                                </key>


                        </element>
```

# ANNEX C

# ACRONYMS AND TERMINOLOGY

| Term | Meaning |
|---|---|
| BCD | Binary Coded Decimal |
| CCSDS | Consultative Committee for Space Data Systems |
| CRC | Cyclic Redundancy Check |
| LRV | last reported value OR last recorded value |
| LSB | least significant bit; least significant byte |
| MSB | most significant bit; most significant byte |
| OMG | Object Management Group, Inc. |
| TAI | International Atomic Time |
| UML | Unified Modelling Language |
| W3C | World Wide Web Consortium |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |
| XTCE | XML Telemetric and Command Exchange |