# Table of Contents

**EXECUTIVE SUMMARY**

Intended Audience i

**COPYRIGHT AND LICENSE NOTICE** iii

## 1 INTRODUCTION TO CAPITAL FACILITY INDUSTRY XML (CFIXML) 1

### 1.1 What is Available and Where to Access 1
1.1.1 The cfiXML Package 1
1.1.2 Where to Find cfiXML Schemas 2
1.1.3 Schema File Folder Organization 2

### 1.2 Schema Content Overview 2
1.2.1 Equipment Data Foundation 3
1.2.2 Equipment Items 3
1.2.3 Construction Materials and Properties 3
1.2.4 Streams 3

### 1.3 Key Design Criteria and Features 4
1.3.1 Reusability 4
1.3.2 Flexibility and Extensibility 4
1.3.3 Full cfiXML Versions and Revisions System 5
1.3.4 Full cfiXML Units of Measurement 5

### 1.4 Introducing cfiXML LT Schema 5

### 1.5 Choosing Between LT and Full cfiXML 5
1.5.1 XML with SI, Metric, US Engineering or Customized Unit Sets. 6
1.5.2 Revision History and Data Change Tracking 6
1.5.3 Referencing Data Sets and Identifiers 6

## 2 SCHEMA ARCHITECTURE 7

### 2.1 Use of Namespaces 7
2.1.1 W3C Schema Namespace Declaration 7
2.1.2 cfiXML Schema Namespaces 7

### 2.2 XML Schema Definition (XSD) File Names and Associated Namespaces 7

### 2.3 Core Data Type Schemas 7
2.3.1 Extended Data Types (ext: namespace) 7
2.3.2 Physical Quantities (pq: namespace) 8
2.3.3 Engineering Type Library (etl: namespace) 8

### 2.4 cfiXML Objects 8
2.4.1 Object Basics 8
2.4.2 Object Namespaces (objb: and obj:) 8
2.5 Core Object Schemas
    2.5.1 Context (ctx: namespace) 9
    2.5.2 Projects (proj: namespace) 9
    2.5.3 Documents (dx: namespace) 9
    2.5.4 Materials and Properties (mtrl: namespace) 9
    2.5.5 Site (site: namespace) 9
    2.5.6 Unit Operation and Utility and Process Streams (uo: namespace) 9
    2.5.7 Equipment (eq: namespace) 10

2.6 Subject schemas 10

2.7 Collection-container Schemas 10

3 HOW TO BUILD APPLICATIONS TO USE CFIXML LT 12

3.1 Background Understanding of cfXML Schemas 12

3.2 Development Environments and Tools 12

3.3 Data Requirements 12

3.4 Setting Up an XML Import and Export Interface Software Application – a check list 12
    3.4.1 Establish the XML container to be used 12
    3.4.2 Ensure all required data is included 13
    3.4.3 Design for Substitution Groups and non-scalar data 13
    3.4.4 Confirm choice of using LT schema 13
    3.4.5 Design stores and program interfaces 13
    3.4.6 Implement application code 13
        3.4.6.1 Element order 13
        3.4.6.2 Parent element creation 13
        3.4.6.3 Supporting the alternativeOption attribute 14
        3.4.6.4 Maintaining existing XML data 14
        3.4.6.5 Access to individual elements where multiple instances exist 14
        3.4.6.6 Providing pick-lists 14
    3.4.7 Testing 14

4 HOW TO BUILD APPLICATIONS TO USE CFIXML FULL 15

4.1 Familiarity with cfXML Schemas 15

4.2 Development Environments and Tools 15

4.3 Data Requirements 15

4.4 Setting Up an XML Import and Export Interface – a check list 15
    4.4.1 Establish the XML container to be used 15
    4.4.2 Ensure all required data is included 16
    4.4.3 Design for specific schema features 16
    4.4.4 Confirm choice of using full schema 16
    4.4.5 Design stores and program interfaces 16
    4.4.6 Design the mapping table 16
    4.4.7 Implement the application code 16
        4.4.7.1 Element order 17
        4.4.7.2 Creating objects and managing cfXML objects 17
4.4.7.3  Parent element creation
4.4.7.4  Supporting the alternativeOption attribute
4.4.7.5  Maintaining existing XML data
4.4.7.6  Access to individual elements where multiple instances exist
4.4.7.7  Units support and issues with XSLT and direct XML access [Q.7]
4.4.7.8  Supporting revisions, versions and change management [Q.9]
4.4.7.9  Providing pick-lists and allowing for user values [Q.10]
4.4.7.10 Optional software implementation features [Q.11]

4.4.8  Testing

5  HOW TO DEVELOP AND EXTEND SCHEMA

5.1  Becoming Familiar with cfiXML Schemas

5.2  Data Requirements
5.2.1  Data Dictionaries
5.2.2  Representing Equipment Items in Schema

5.3  Extending cfiXML Schemas
5.3.1  Extensibility mechanism without adding schema definitions
5.3.2  Extensibility mechanism through user-provided schema definitions
5.3.3  Adding Equipment Extensions to Published Standard cfiXML Schema – check list
5.3.3.1  Establish the XML container to be used
5.3.3.2  Review of existing schema
5.3.3.3  Check conventions for naming elements, annotation requirements, naming files and version numbers [M].
5.3.3.4  Create types and elements [L]
5.3.3.5  Add attributes if required [L.9].
5.3.3.6  Consider special modeling requirements [N].
5.3.3.7  Determine whether or not elements need to occur more than once.
5.3.3.8  Write test xml
5.3.3.9  Write XPaths
5.3.3.10 Refine schema

A.  BACKGROUND, PURPOSE AND BENEFITS OF CFIXML

A.1  Fiatech and the AEX Project

A.2  Information Exchange in the Capital Facilities Industry

A.3  High Business Value of Interoperability

A.4  The AEX Solution for Data Exchange
A.4.1  XML
A.4.2  AEX Schema Development

A.5  Demonstration of Real Usage Scenarios
A.5.1  Centrifugal Pump Procurement Demonstration
A.5.2  Key Points from AEX cfiXML Testing and Demonstration

B.  CFIXML LT FEATURES

B.1  Optional ObjectID
E. CFIXML OBJECTS

E.1 Object Basics
   E.1.1 Rationale for Object Design

E.2 Object Namespaces (objb: and obj:)
   E.2.1 Purpose

E.3 Management – Unique Object identification and Referencing Attributes
   E.3.1 objectID
      Rationale for objectID
   E.3.2 contextURL - Object registration
      Rationale for Object registration
   E.3.3 organizationID – ANSI registration
   E.3.4 Version
   E.3.5 Addressing

E.4 Object Relationships – Contained and Referenced Objects
   E.4.1 Rationale for Contained and Referenced Objects
   E.4.2 objectState

E.5 Handling non-cfXML or customized cfXML Objects
   E.5.1 Object Contents
   E.5.2 External (non-cfXML) Objects
      Basics
   E.5.3 Hybrid objects
   E.5.4 Alien objects

E.6 Revisions, Versions, Transactions and Change Management

E.7 Capital Facility Equipment Special Concepts

F. CORE OBJECT SCHEMAS

F.1 Context and Projects (ctx: and proj: namespaces)

F.2 Documents (dx: namespace)

F.3 Materials and Properties (mtrl: namespace)

F.4 Site (site: namespace)

F.5 Unit Operation and Utility and Process Streams (uo: namespace)

F.6 Equipment (eq: namespace)

G. SUBJECT SCHEMAS

G.1 Subject Schema Overview
   G.1.1 Inheritance
   G.1.2 Effects of inheritance
      Sequences 62
      Content beyond requirements
G.2  Schema Representation of Equipment Items  62
G.3  Collection-container Schemas (‘Documents’) Overview  63

H. QUALITY DATA EXCHANGE AND DATA DICTIONARIES  65
H.1  Problems of defining data to be exchanged  65
H.2  Data dictionaries  65
   H.2.1  Developing a data dictionary  65
          Stakeholders  65
          Data item specifications  65
          Schema development and final checks  66
   H.2.2  The AEX Project and Hydraulic Institute (HI) pump data dictionaries  66

I. SCHEMA SETTINGS  67
I.1  Rationale for Schema Settings  67

J. XPATH AND XML EXAMPLES  68
J.1  XPaths  68
   J.2  Example XPaths and XML  68
      J.2.1  Document header data  68
      J.2.2  Site data  69
      J.2.3  General equipment data  69
      J.2.4  Equipment properties  70
      J.2.5  Construction materials  70
      J.2.6  Substitution group  71
      J.2.7  Usage context  72
      J.2.8  Equipment stream connections – substitutionGroup, usageContext and objectState  73
      J.2.9  Equipment inspection tests – extensions and enumeration union  74
      J.2.10  Use of alternativeOption  75
      J.2.11  Use of order attribute for array lists  75
      J.2.12  Use of fixedProperty  76

K. PRINCIPLES OF SCHEMA MODELING AND DEVELOPMENT, AND ENGINEERING DATA  77
K.1  XML Schema Development Process Overview  77
   K.1.1  Business-driven Data Requirements Definition (domain users)  77
   K.1.2  Schema Development (data modelers working with domain experts)  78
   K.1.3  Schema Deployment (domain application developers working with domain expert users)  78

K.2  Schema Information Modeling Principles  78

K.3  Modeling principles  79
   K.3.1  Develop and deliver business process models.  79
   K.3.2  Develop data models using common engineering terminology.  79
   K.3.3  Define data models using a simple, explicit and contained style.  79
   K.3.4  Use standard modeling notations where possible.  79
   K.3.5  Use modular data architecture.  80
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K.3.6</td>
<td>Define / develop reusable concepts underlying large amounts of data.</td>
</tr>
<tr>
<td>K.3.7</td>
<td>Adopt a set of naming rules and conventions.</td>
</tr>
<tr>
<td>K.3.8</td>
<td>Define needed alternative names in application or XSLT mappings.</td>
</tr>
<tr>
<td>K.3.9</td>
<td>Group related data items together into reusable logical groups.</td>
</tr>
<tr>
<td>K.3.10</td>
<td>Minimize complexity and depth of nesting.</td>
</tr>
<tr>
<td>K.3.11</td>
<td>Define appropriate relationships between data groups.</td>
</tr>
<tr>
<td>K.3.12</td>
<td>Treat user presentation issues separately from data modeling issues.</td>
</tr>
<tr>
<td>K.4</td>
<td>XML Schema Development Principles</td>
</tr>
<tr>
<td>K.4.1</td>
<td>Use XML Schema to define information models</td>
</tr>
<tr>
<td>K.4.2</td>
<td>Use object-oriented data models as the basis for XML Schema.</td>
</tr>
<tr>
<td>K.4.3</td>
<td>All elements in an XML schema should be optional by default.</td>
</tr>
<tr>
<td>K.4.4</td>
<td>Define/document minimum data sets for transaction validation.</td>
</tr>
<tr>
<td>K.4.5</td>
<td>Define common user extension conventions.</td>
</tr>
<tr>
<td>K.4.6</td>
<td>Support schema versions.</td>
</tr>
<tr>
<td>K.5</td>
<td>Engineering Data Principles</td>
</tr>
<tr>
<td>K.5.1</td>
<td>Develop reusable schemas that underlie various engineering schemas.</td>
</tr>
<tr>
<td>K.5.2</td>
<td>Define a systematic approach for units of measurement.</td>
</tr>
<tr>
<td>K.5.3</td>
<td>For simplicity, consider adopting SI units only.</td>
</tr>
<tr>
<td>K.5.4</td>
<td>Define a systematic approach for handling valid data ranges.</td>
</tr>
<tr>
<td>K.5.5</td>
<td>Support vectors, tables and multi-dimensional arrays.</td>
</tr>
<tr>
<td>K.5.6</td>
<td>Optionally support stored information about user access control.</td>
</tr>
<tr>
<td>K.5.7</td>
<td>Optionally support encrypted binary data for secured data access.</td>
</tr>
<tr>
<td>K.5.8</td>
<td>Provide standard mechanisms for extending XML schema.</td>
</tr>
<tr>
<td>K.5.9</td>
<td>Provide standard approaches to object identification and references.</td>
</tr>
<tr>
<td>K.5.10</td>
<td>Support embedded binary data.</td>
</tr>
<tr>
<td>K.5.11</td>
<td>Optionally support storage of associated revision data.</td>
</tr>
<tr>
<td>K.5.12</td>
<td>Provide descriptive data about roles or use of a data element.</td>
</tr>
<tr>
<td>K.5.13</td>
<td>Provide a mechanism to determine data type of “any” elements.</td>
</tr>
<tr>
<td>K.6</td>
<td>Deployment Requirements and Constraints</td>
</tr>
<tr>
<td>K.6.1</td>
<td>Complete Documents</td>
</tr>
<tr>
<td>K.6.2</td>
<td>Duplicate XML</td>
</tr>
<tr>
<td>K.6.3</td>
<td>XML Validation and Test Files</td>
</tr>
<tr>
<td>K.6.4</td>
<td>XSLT and Standard Parser Processing</td>
</tr>
<tr>
<td>L.1</td>
<td>Rationale for Types</td>
</tr>
<tr>
<td>L.2</td>
<td>Elements</td>
</tr>
<tr>
<td>L.2.1</td>
<td>Rationale for Use of Elements versus Attributes</td>
</tr>
<tr>
<td>L.2.2</td>
<td>Creating Base “Leaf” Elements – use of ext and pq namespaces</td>
</tr>
<tr>
<td>L.2.3</td>
<td>Optional by default</td>
</tr>
<tr>
<td></td>
<td>Rationale for Elements Being Optional By Default</td>
</tr>
<tr>
<td>L.3</td>
<td>Creating Complex types</td>
</tr>
<tr>
<td>L.4</td>
<td>Creating SimpleType Enumeration Lists</td>
</tr>
<tr>
<td>L.4.1</td>
<td>Enumeration list order and format</td>
</tr>
<tr>
<td>L.5</td>
<td>Format of complexType and simpleType Enumeration Names and Associated Elements</td>
</tr>
<tr>
<td>L.6</td>
<td>Location of Types and Elements in Schema</td>
</tr>
</tbody>
</table>
L.7 Use of Global Element References v Global Types

L.8 Content Models
L.8.1 No use of ‘All’
L.8.2 No use of ‘Choice’
L.8.3 Use of ‘Sequence’ and Ordering Conventions

L.9 Attributes and Qualifiers
L.9.1 Using Conditions or Qualifiers
L.9.2 ReferenceProperty and ReferencePropertyID
L.9.3 Usage Context Attribute
   Rules for the use of usageContext in schema and xml

L.10 Creating an object

M. CONVENTIONS – NAMING ELEMENTS AND FILES, ANNOTATION, VERSION NUMBERS

M.1 Naming Elements
M.1.1 Use of Case
M.1.2 Prefix and Suffix Codes
   Rationale for Prefix and Suffix Codes
M.1.3 Use of Standard Abbreviations and Acronyms
M.1.4 Use of Units of measurement
M.1.5 Company names, associations, non-international standard cataloguing or identification systems
   Rationale for Company names, association etc.

M.2 Annotation – xsd:appinfo and xsd:documentation
M.2.1 appShortName and appLongName

M.3 File Headers

M.4 Version Numbers

M.5 Schema Files

N. SPECIAL MODELING CONCEPTS

N.1 Use of Substitution Groups
N.1.1 Schema development considerations
N.1.2 XML content
N.1.3 Application support
N.1.4 Enumeration format for substitution groups
N.1.5 Rationale
   Derivable type rationale and structure in full schema
   Re-assessment of Choice
   Substitution Groups

N.2 Use of Derivable Types (xsi:type and xsiTypeNS) prior to Public Release V 4.0

N.3 Use of AlternativeOption attribute

N.4 key and keyref
Q.8  Populating XML structures – Processing Rules
  Q.8.1  Element Order
  Q.8.2  Creating Objects
  Q.8.3  Parent Element Creation
  Q.8.4  Supporting the alternativeOption Attribute
  Q.8.5  Maintaining Existing XML Data
  Q.8.6  Access to Individual Elements Where Multiple Instances Exist

Q.9  Supporting Revisions, Versions and Change Management
  Q.9.1  Versions and Transactions
  Q.9.2  Revisions
  Q.9.3  Change Tracking
  Q.9.4  Full Change Tracking, Revision and Versions System

Q.10  Providing Pick-lists and Allowing for User Values

Q.11  Optional Software Implementation Features
  Q.11.1  User Access Control
  Q.11.2  Read and Write Status
  Q.11.3  Mappings to Other Schemas
  Q.11.4  Default Values and Ranges

R.  KEY FOR FIGURES

S.  CENTRIFUGAL PUMPS
  S.1  Key Schema files
  S.2  Modeling of centrifugal pumps
  S.3  Description of some key data types
  S.4  Example XML files
  S.5  Pump Tutorial – Using the cfiXML Schemas
     S.5.1  Pump Tutorial Part 1 – Using the existing schemas
            Steps in discovering the schema and instance file structure
     S.5.2  Pump Tutorial Part 2 – Extending/Customizing the schemas
            Part 2a – Adding custom data directly through custom elements
            Part 2b – Adding elements by defining a custom extension schema
            Part 2c – Adding extensions to enumeration variable values
     S.5.3  Extending enumeration elements using the “OtherValue” method

T.  CENTRIFUGAL FANS
  T.1  Key Schema files
  T.2  Modeling of fans
  T.3  Description of data types
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>T.4</td>
<td>Example XML file</td>
</tr>
<tr>
<td>U.</td>
<td>SHELL AND TUBE HEAT EXCHANGERS</td>
</tr>
<tr>
<td>U.1</td>
<td>Key Schema files</td>
</tr>
<tr>
<td>U.2</td>
<td>Modeling of Shell and Tube Heat Exchangers</td>
</tr>
<tr>
<td>U.3</td>
<td>Description of some key data types</td>
</tr>
<tr>
<td>U.4</td>
<td>Example XML file</td>
</tr>
<tr>
<td>V.</td>
<td>AIR COOLED HEAT EXCHANGERS</td>
</tr>
<tr>
<td>V.1</td>
<td>Key Schema files</td>
</tr>
<tr>
<td>V.2</td>
<td>Modeling of Air Cooled Exchangers</td>
</tr>
<tr>
<td>V.3</td>
<td>Description of some key data types</td>
</tr>
<tr>
<td>V.4</td>
<td>Example XML file</td>
</tr>
<tr>
<td>W.</td>
<td>ELECTRIC MOTORS</td>
</tr>
<tr>
<td>W.1</td>
<td>Key Schema files</td>
</tr>
<tr>
<td>W.2</td>
<td>Modeling of Electric Motors</td>
</tr>
<tr>
<td>W.3</td>
<td>Description of some key data types</td>
</tr>
<tr>
<td>W.4</td>
<td>Example XML file</td>
</tr>
<tr>
<td>X.</td>
<td>BLOCK AND CONTROL VALVES</td>
</tr>
<tr>
<td>X.1</td>
<td>Key Schema files</td>
</tr>
<tr>
<td>X.2</td>
<td>Modeling of Valves</td>
</tr>
<tr>
<td>X.3</td>
<td>Description of some key data types</td>
</tr>
<tr>
<td>X.4</td>
<td>Example XML files</td>
</tr>
<tr>
<td>Y.</td>
<td>HVAC EQUIPMENT</td>
</tr>
<tr>
<td>Y.1</td>
<td>Air Handling Unit Key Schema files</td>
</tr>
<tr>
<td>Y.2</td>
<td>Modeling of Air Handling Units</td>
</tr>
<tr>
<td>Y.3</td>
<td>Description of some key AHU data types</td>
</tr>
<tr>
<td>Y.4</td>
<td>AHU Example XML file</td>
</tr>
<tr>
<td>Y.5</td>
<td>Vapor Compression Chiller Key Schema files</td>
</tr>
<tr>
<td>Y.6</td>
<td>Modeling of Chillers</td>
</tr>
<tr>
<td>Y.7</td>
<td>Description of some key Chiller data types</td>
</tr>
<tr>
<td>Y.8</td>
<td>Chiller Example XML file</td>
</tr>
</tbody>
</table>

INDEX 156
Figures

Figure 1  Equipment item coverage and groups of related schema files .............................................. 4
Figure 2  Schema model layered structure and reusability of AEX schemas using an example of a centrifugal pump data sheet ................................................................. 11
Figure 3  Software data exchange via standardized industry file formats or database .... 25
Figure 4  AEX Project analysis of equipment life cycles work flow processes for major capital facilities ..................................................................................................................... 27
Figure 5  Data exchange sequence in the centrifugal pump procurement demonstration 29
Figure 6  AEX cfiXML Namespace Overview .................................................................................... 36
Figure 7  Illustration of data type extension provided in the 'ext' namespace ............... 38
Figure 8  Using ext: data types and Base Attributes ........................................................................ 40
Figure 9  Illustration of use of pq: data types .................................................................................. 41
Figure 10 Key object components for recording data transactions ........................................... 55
Figure 11 Outline of some of the key components of mtrl:xsd .................................................. 58
Figure 12 Outline of key components of site:xsd ............................................................................. 58
Figure 13 Simplified schema modeling structure of equipment items ................................. 63
Figure 14 Example of a reusable sequence of elements ................................................................. 88
Figure 15 Example of derivable types - end connections .............................................................. 101
Figure 16 Example use of alternativeOption on a parent element (mtrl:material) ...... 102
Figure 17 Transaction elements provided for all objects in cfiXML ............................................. 119
Figure 18 Outline of key schema components of the Centrifugal Pump Data Sheet .... 126
Figure 19 Outline of key schema components of the Centrifugal Fan Data Sheet ............ 141
Figure 20 Outline of key schema components of the Shell and Tube Heat Exchanger Data Sheet ................................................................. 143
Figure 21 Outline of key schema components of the Air Cooled Heat Exchanger Data Sheet .................................................................................................................. 145
Figure 22 Outline of Key Schema Components for Electric Motors ........................................... 148
Figure 23 Outline of key schema components of the Control Valve Data Sheet and Block Valve Catalogue ................................................................................................. 150
Figure 24 Outline of key schema components of the Air Handling Unit Data Sheet .... 152
Figure 25 Outline of key schema components of the Vapor Compression Chiller Data Sheet ...................................................................................................................... 154
Tables

Table 1 Summary of Base Attributes in the ext namespace ........................................ 39
Table 2 Standard components of unit symbol strings ............................................. 42
Table 3 Standard prefixes for unit symbols .......................................................... 43
Table 4 Examples of data types included in etl ..................................................... 43
Table 5 Object Identification Attributes in objb:BaseObject .................................. 46
Table 6 objectState attribute enumerations .......................................................... 50
Table 7 Valid Attribute Combinations for Contained Objects ................................. 50
Table 8 Object Content Attributes .................................................................... 51
Table 9 Valid Combinations of Content Attributes for Objects .............................. 51
Table 10 Object components for recording data transactions .................................. 55
Table 11 Examples of dx: data types to describe and identify documents ............... 57
Table 12 Examples of schema definition files in the eq namespace ...................... 59
Table 13 Description of facility equipment item objects ....................................... 60
Table 14 Abbreviations and acronyms accepted in schema ................................. 96
Executive Summary

This reference guide explains the design, content, methods for development and points for implementation of the capital facilities industry XML (cfiXML) schema model for electronic data exchange and sharing of equipment data in the capital facilities industry.

This document is structured to provide high level support so that a user can pick and choose features and link to detail in appendices, as required. This format is to help support phased introduction and learning of the schema for software application and schema developers.

Many of the underlying principles are based on the work of others in the XML development community, e.g. ebXML, as well as the participants experience with software and standards development activities and the practical application of these in supporting interoperable software systems.

This reference material is intended for both users of the schema, and those who would extend it to new subject domains. The schema design reflects actual software implementation experience associated with the development and multiple software implementations of cfiXML. The cfiXML has been a joint development effort of Fiatech, Hydraulic Institute (HI), DIPPR (Design Institute for Physical Properties), Alar Engineering Software, Inc. and Protesoft Corporation.

Recent advances include:

- an initiative supported by the National Institute for Standards and Technology (NIST) resulting in cfiXML Light (LT) to enable simplification of software application development;
- the HI and Electric Power Research Institute (EPRI) assessment of cfiXML for data exchange of design to installation specifications for a safety pump, associated with development of new nuclear power plants and the capture of information required by latest regulations;
- Hydraulic Institute support for development of centrifugal and rotary pump procurement specifications.

In addition to documenting the architecture and core schemas of cfiXML, this document should be considered as an ongoing “best practices” knowledge capture of relevant practical industry experience for deploying useful XML interoperability solutions that work. It will continue to be refined for use in Fiatech projects and we encourage other capital facilities industry organizations to consider adopting the best practices described herein to facilitate successful software interoperability deployment efforts. We invite comments for improvement by all readers.

Schema developers may be seeking to develop XML-based software interoperability solutions, and wish to leverage and build upon a large base of previous work to extend this work into new subject domains. Alternatively, these developers may wish to extend
the already developed subject domains into more details to support a given usage scenario.

Application interface developers may wish to understand how to deploy cfiXML data exchange and sharing interfaces into their specific application software to support a specific usage scenario. These interfaces will include “mapping interfaces” between internal application-specific data storage structures and the industry-consensus object structure of cfiXML files for the purpose of electronically exchanging or sharing data from or into their application software with other application software to meet a known usage scenario.

If you have comments or recommendations for this document, or would like more information on the Fiatech program and the development of these guidelines, you may contact Mark Palmer, email: mark.palmer@nist.gov, or contact any of the other members of the Guidelines Team (listed on the cover page).

**Intended Audience**

This document is provided in sections intended to meet the needs of specific audiences, as follows:

- **Chapter 1 is for any interested party** who requires an introduction to the cfiXML schema model, including benefits of the design and features, and how it works to solve interoperability problems. This chapter assumes no detailed knowledge of IT.

- **Chapter 2 outlines the architecture of cfiXML** for schema and application developers in particular with links to details in the appendices.

- **Chapter 3 lists considerations for an application developer using cfiXML LT.** The developer can pick and choose features, and link to details in appendices, as required.

- **Chapter 4 lists considerations for an application developer using full cfiXML.** The developer can pick and choose features, and link to details in appendices.

- **Chapter 5 outlines key considerations for a schema developer.** The developer can pick and choose features, and link to details in appendices, as required.

Links to the appendices are most commonly provided in square brackets [ ] and are throughout the text.
Copyright and License Notice

Several organizations contributed to the development of the XML schemas for exchanging capital facilities equipment information and we anticipate that additional organizations will openly contribute to its development in the future. Fundamental to this open, cooperative development is the agreement that the resulting schemas will be freely available for anyone to use. In order to maintain suitable configuration control, while allowing organizations to continue to develop improvements and additions, the following open source licenses and copyright language is included with all schema files.

“Portions Copyright (C) 2001-2006 AIChE-DIPPR, All Rights Reserved.
Portions Copyright (C) 2001-2012 Alar Engineering Software, All Rights Reserved.
Portions Copyright (C) 2001-2006 ePlantData, Inc., All Rights Reserved.
Portions Copyright (C) 2002-2012 Fiatech, All Rights Reserved.
Portions Copyright (C) 2001-2006 Protesoft Corporation, All Rights Reserved.

This file contains Original Code and/or Modifications of Original Code (the "Contents") as defined in and that are subject to the DIPPR Public Source License Version 1.0, the Alar Engineering Software Inc. Public Source License Version 1.0 and the Fiatech Public Source License Version 1.0\(^1\). Copies of these licenses are available at http://www.cfixml.org. You may only use the Contents according to the terms and conditions in these Licenses.

The Contents are distributed for use on an ‘AS IS’ basis, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED. LICENSOR HEREBY DISCLAIMS ALL SUCH WARRANTIES, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT OR NON-INFRINGEMENT.”

---

\(^1\) Directly derived from the Apple Public Source License Version 1.2 provided by the Open Source Initiative on their website, http://opensource.org/licenses/apsl.php
1 Introduction to capital facility industry XML (cfiXML)

The cfiXML schemas have been developed to provide a pragmatic solution for electronic data exchange in the capital facilities industry and are available for use now.

The Fiatech Automating Equipment Information Exchange (AEX) Project was launched in 2002 with the objective of automating equipment design and delivery through software interoperability [A.1]. Automatic exchange of technical data between stakeholders in the capital facilities industry has significant advantages [A.2]. Deployment of an interoperability solution will potentially save the capital facilities industry millions of dollars a year [A.3]. A key part of the solution requires industry-wide agreements on, and deployment of, a software-independent electronic data exchange system, based on a standardized method of describing and storing data [A.4].

The AEX capital facilities industry XML (cfiXML) model developed to be capable of handling any equipment item, engineering document and material property for any information exchange usage scenario across the entire facilities life cycle. The model is based on XML as the goal of XML is to be application-independent. Being text-based ensures that most software applications should be definable in terms of XML. Additionally, text files will always be readable, so data will always be accessible: an important consideration given the decades involved in a facility life cycle [A.4.1].

The AEX Project collaboration focused on the high-value business processes of the equipment supply chain and key equipment items to develop required common XML data types, structures and definitions [A.4.2]. However, schema has been designed so that it will be capable of handling any equipment item, engineering document and material property for any information exchange usage scenario [Figure 4]. The cfiXML model has been demonstrated to work as a data exchange standard language usable by different software applications [A.5.1] and continues to develop. The schema files are reusable, flexible and easily extendable to ensure that cfiXML provides a pragmatic and cost effective system to extend and implement.

To use cfiXML to support electronic data exchange requires a key initial investment of time and programming to create an XML interface. In 2011, the cfiXML Light (LT) schema version was developed to support the use of simple XML serialization tools and so enable developers with limited XML experience to be able to develop data exchange systems more easily and cost-effectively [1.4].

1.1 What is Available and Where to Access

1.1.1 The cfiXML Package

The following support for the process of automating information exchange using cfiXML has been developed.

- The cfiXML schemas provide the common language for software applications. These are available as public domain files to encourage widespread uptake and
acceptance. There are 2 schema models available: full cfiXML and cfiXML Light (LT). LT schema provides an easier and quicker route to developing standard-based data exchange applications.

- **Guideline documentation** (this document) describes design and structure of cfiXML, how to build applications using full cfiXML or cfiXML LT, and how to develop or extend the schema model.

- **Data dictionaries** are spreadsheets which have been created to list specifications for a particular type of engineered equipment, in terms relating to data items in cfiXML schemas. These allow different organizations to ensure that there is consensus on which data are being exchanged for specific work processes.

- **Example XML files** are provided for key equipment items and for a particular work flow (e.g. data from a purchaser to a supplier to initiate a request for quotation – a Bid RFQ transaction). These illustrate the principles of the design of cfiXML as an aide to software developers and reference standards for testing software. Additional documentation on testing criteria using some of these example XML files is also available.

### 1.1.2 Where to Find cfiXML Schemas

The latest versions of cfiXML schemas are available from the following web sites:

1) cfiXML: [http://www.cfixml.org/](http://www.cfixml.org/) (navigate to schema and documentation download pages)

2) SourceForge: ‘cfidev’ project on SourceForge ([http://sourceforge.net/projects/cfidev](http://sourceforge.net/projects/cfidev)).

   (Note: if you open an account with SourceForge, the hosting open source software development web site, then you will be able to input into the bugs & requests system.)

### 1.1.3 Schema File Folder Organization

The cfiXML schemas are provided with the following file folder structure:

```
[cfiXML]
  [documentation]  (Public cfiXML documentation files, example data dictionaries)
  [examples]        (Example XML instance files)
  [schema]          (All cfiXML Core Data Type, Core Object and Subject schemas)
      [custom]    (Implementer specific customized schemas)
      [document]  (All cfiXML Collection-container schemas, e.g. data sheets)
```

### 1.2 Schema Content Overview

The cfiXML model has been developed to be capable of handling any equipment item, engineering document and material property for any information exchange usage scenario.
1.2.1 Equipment Data Foundation
AEX schemas are based on a set of foundation files which describe non-technical data on the context, project and documentation associated with a data set. In addition, data on physical properties and units of measurement are included. The foundation files therefore cover a wide range of information, from an organization’s name, URL and project details, to ambient temperatures at a capital facility site.

1.2.2 Equipment Items
Building on from this foundation, schema has been produced for key equipment items, with particular emphasis placed on meeting the needs of a procurement work flow cycle. For example, detailed schema is available for:
- centrifugal pumps and fans,
- shell and tube heat exchangers,
- air-cooled heat exchangers,
- centrifugal and reciprocating compressors,
- electric motors,
- control and block valves,
- air handling units and chillers.
In addition, base sets of data for numerous equipment items and accessories are available. The equipment or accessory schema files are divided into groups (illustrated in Figure 1).

1.2.3 Construction Materials and Properties
The equipment item schemas are supported by a comprehensive set of material properties, which were originally developed in collaboration with the Design Institute for Physical Properties (DIPPR). Description of process materials and associated properties as well as equipment construction materials and properties are included. The model also provides for exchange of property curve and property table data suitable, for example, as predictive model parameters.

1.2.4 Streams
Files covering streams and stream conditions complete the set of schemas required to describe conceptual through design and operating performance specifications for an equipment item.
1.3 Key Design Criteria and Features

There are many ways of representing XML data, each of which have different consequences in terms of reusability, extensibility, ease of processing and ease of writing XML documents. The design principles are based on extensive research of available resources in the XML Schema development community to be in line with best practices. The standards for XML schema provided by W3C for specifying and validating the contents of XML data have been adopted. Hence, cfiXML is constructed using W3C standard XML Schema as its basis and developed to handle the technical information requirements of the capital facilities industry. The AEX schema implementation of the standards is verified with available XML testing systems (parsers).

1.3.1 Reusability

A major consideration of the development of the robust, reusable object framework was to ensure that extending cfiXML to handle equipment items is a straightforward and cost-effective process. The reusability of the core data types, core object schemas and subject schemas is illustrated in Figure 2.

1.3.2 Flexibility and Extensibility

Another key part of the design of cfiXML schemas is the built-in flexibility provided by relatively simple methods to extend schema files. Organizations and individuals may wish to make locally used modifications to the common industry XML schemas. Provisions have been made throughout the schema to allow extra values to be added to enumeration (pick-) lists, and to add custom data items to support particular software requirements, without having to extend the existing XML Schema Definition (XSD) files. Provisions for more extensive development are also available with recommended approaches for user defined custom schemas. Extensions built in the recommended manner guarantees that anyone using cfiXML compliant software will still be able to use

---

2 http://www.xfront.com/B...5 standards/xml/
and exchange the majority of such information, with the exception of the local extended variations.

1.3.3 Full cfiXML Versions and Revisions System
The core features of the schema provide support for version management so that records can be kept throughout the life cycle of the equipment item. The design also incorporates a means to record any changes to data items, i.e. change tracking.

1.3.4 Full cfiXML Units of Measurement
A final key feature of AEX schemas is the inclusion of units of measurement which enables the required units to be set for any defined physical property throughout the schema. So for example, the physical property of temperature can be specified in units of K, degrees C or degrees F.

1.4 Introducing cfiXML LT Schema
The cfiXML LT initiative was a NIST\(^4\) sponsored project to encourage further implementation of cfiXML by establishing ways to simplify and reduce the effort of building practical applications that use cfiXML. The primary goal of cfiXML LT is to provide an easier and quicker route to developing standard-based data exchange applications.

Relatively simple XML application development tools are now available, which provide developers with little XML experience a means to build software applications. The LT project focused on the challenges of using XML serialization tools to assess whether possible alterations in schema design could support easier implementation. Also, the contents and support provided by the full schema is extensive. Many relatively simple data exchange usage scenarios can be supported by a subset of the current schema model. For specific equipment items and usage scenarios, the LT project aimed to define the common subset of required data specifications that an application should support.

The cfiXML Light (LT) Specifications document describes the goals, requirements and constraints for cfiXML LT. Further information is included to: (1) support decision-making on whether LT or full schema will meet a user’s requirements; (2) describe considerations, recommended practices and testing criteria when developing a cfiXML LT-based data exchange system, and (3) explain the LT-tool and process used to create schema versions.

1.5 Choosing Between LT and Full cfiXML
Implementers must examine their system’s needs and understand which version is best suited for their use-case. Most implementers considering the use of cfiXML as a means of transacting or exchanging point-in-time data between 3\(^{rd}\) party systems will find that schema LT meets their immediate needs. Systems developers contemplating relying on cfiXML for data repository and management needs and for their underlying business models, will find full cfiXML schema more suitable.

\(^{4}\) www.nist.gov
If looking to adopt cfiXML in an existing application (import, export, or both), the easiest approach is to begin by supporting LT. Implementing LT will not preclude upgrading the application to support full cfiXML if needed. The following is a summary of the differences in key features of full versus LT cfiXML. **For simpler applications with stable requirements LT is a good choice. If the 3 items below are NOT a MUST for your application LT will probably meet your needs.**

1.5.1 XML with SI, Metric, US Engineering or Customized Unit Sets.

LT cfiXML focuses on supporting point-in-time data exchange only, and to simplify implementation it prohibits specification of unit sets within the XML document. All physical quantities are required to be stored in SI units within an LT cfiXML document and unit conversions are left to the applications working with the XML. Full cfiXML supports SI (the default), Metric and US Engineering unit sets and customization, which mean applications supporting full cfiXML must be capable of reading data in any unit set supported by the schema. [See B.3 for more details.]

1.5.2 Revision History and Data Change Tracking

LT cfiXML documents are specifically not designed to store the history of an equipment data set over time. LT documents are designed to be *snapshots* of the data being exchanged between parties, where each application likely stores master copies of the data (perhaps with revision history) in their own backend systems. Revision support, which is embedded in full cfiXML schema, is removed from LT to provide a simplified data model. Applications which are required to include revision and version data, and change tracking of individual data items, within the XML should consider using the full schema version. LT cfiXML documents do contain XML elements suitable for time-stamping and commenting document-wide revisions. [See B.2 for more details.]

1.5.3 Referencing Data Sets and Identifiers

The key idea of a cfiXML object is that it can be thought of as a “smart container” which can be accessed by its name, like understanding the contents of a box without having to open the box. Providing the mechanism to uniquely identify objects anywhere in the XML enables software to refer to and use these data objects (smart containers). So, data objects can reference other objects elsewhere in the XML or even other resources. For example, a reference to a specific object via its unique identification can be used to locate data that is held elsewhere, either in the same file, or in different files over the network without having to replicate the schema or the data. [2.4.1]

Full cfiXML guidelines require an implementer to uniquely identify objects by (at least) creating objectID attributes on core data sets (objects) to enable object referencing if required [E.3.1; E.4]. This feature is useful in a variety of applications, as it allows a *software* artifact to be traced within an XML document, or across several XML documents and applications. While optional in LT, this function is not seen as required for simple data exchange and implementers are not required to use this functionality. When using cfiXML as an application’s primary data model, the full version may be more suitable. [See B.1 for more details.]
2 Schema Architecture

The AEX cfiXML model is a layered structure (Figure 2). This ensures that there is a common consistent foundation and maximum potential for the reuse of items for efficient schema development. There are four basic layers of the cfiXML architecture.

Inheritance is used throughout the cfiXML model, so that sets of common data types or objects can be reused, as required. Inheritance allows the extended data types to include all the features of common data sets, with extensions to handle additional requirements [G.1.1].

2.1 Use of Namespaces

Rationale for Namespace Design [C.1]
Namespaces are used throughout cfiXML, primarily as support for schema development.

2.1.1 W3C Schema Namespace Declaration
As cfiXML is based on the schema specification from the World Wide Web Consortium (W3C) all cfiXML schemas include the W3C schema namespace declaration [C.2].

2.1.2 cfiXML Schema Namespaces
The cfiXML schema model (Figure 2) uses uniquely named XML namespaces to allow multiple reusable core and subject-specific schemas [C.3] to be used together in a single XML ‘Collection-container’ schema file (‘document’) [C.4].

2.2 XML Schema Definition (XSD) File Names and Associated Namespaces
The cfiXML schema definition files include the name of the associated namespace identifier [C.5]. For examples and file content explanation see C.5.1.

2.3 Core Data Type Schemas
Core data type schemas are built from the W3C XML Schema Standard basic data types to provide a common foundation of features available for all data in cfiXML. There are 3 Core Data Type namespaces.

2.3.1 Extended Data Types (ext: namespace)
Extended Data Types (ext:) add base attributes to standard W3C XML Schema data types to support annotation and revision tracking at the element level and to define scalar (single value) and list (vector) data types. [D.1.1]

Additional attributes provided at the element level include support for alternativeOption and usageContext, enable elements to be nillable, as well as supporting schema extension without altering the definition files (for example, through the use of otherValue in simple types).
[Table 1 Summary of Base Attributes in the ext namespace].
2.3.2 Physical Quantities (pq: namespace)

In pq, physical quantity data types are defined for data elements that are associated with units of measurement [D.2.1] The units of measurement for each physical quantity are provided as enumeration lists [D.2.2] as well as the default unit sets of SI, Metric and US Engineering [D.2.3]. Physical quantity types are defined for scalar (single value) types, extended from ext:Double [D.2.4].

2.3.3 Engineering Type Library (etl: namespace)

The etl collects miscellaneous data types, which are not objects, including simple shape geometries and electrical source and area classification specifications. [D.3] [Table 4 Examples of data types included in etl]

2.4 cfiXML Objects

Not relevant to cfiXML LT application implementations that do not use objectID. Rationale for Object Design [E.1.1]

2.4.1 Object Basics

A cfiXML object can be thought of as a “smart container” which can be accessed by its name - like understanding the contents of a box without having to open the box [E.1]. Providing the mechanism to uniquely identify objects [E.3] anywhere in the XML enables software to refer to and use these data objects (smart containers). So, data objects can reference other objects elsewhere in the XML or even other resources. Without the capability of a persistent, object-oriented structure, such functionality becomes extremely complex and difficult. [Table 5 Object Identification Attributes in objb:BaseObject]

Objects can and commonly do contain other objects within them [E.4]. In this way, objects of arbitrary complexity can be constructed to meet the needs of describing real world applications.

Except for the Core Data Types schemas described in the previous section, most of Core Object and Subject Schema content extends, or inherits from, objects.

2.4.2 Object Namespaces (objb: and obj)

objb: defines the Base Object, which includes the fundamental attributes and base element data required to identify, name and manage objects that can be defined and referenced by other elements and objects in the schema [E.3]. All other cfiXML objects are derived from objb:Object.

obj: extends the Base Object to provide the capabilities for recording object transactions, e.g., software systems and persons who made a transaction, the details of the transaction, revision notes, tracking current object type and status [E.6]. All cfiXML objects, such as equipmentItem and dataSheet, are based on obj.
2.5 Core Object Schemas

The Core Object Schemas represent a number of key areas which are reusable and applicable across multiple subject domains.

Core object schemas include reusable base engineering objects that can be used by multiple engineering disciplines and subject domains. These objects consist of a base set of data types and attributes which enable any item extended from an object to be uniquely identified throughout the life-cycle of the item. Information on documents (ownership and identification), context (organization, person, location), the project, site and material properties are all included in the core object schemas.

In general, engineering objects contain a large number of individual engineering data elements that are typically grouped together in logical ways. For example, in Figure 2 an eq:FacilityItem is a type of Object that contains a number of elements that describe the facility item, such as the element weight. There are many types of physical items in a facility, but all of them can have the common characteristic of having one or more weights associated with that item.

2.5.1 Context (ctx: namespace)
Context (ctx:) objects provide details on the context for data, location, organization and person [F.1].

2.5.2 Projects (proj: namespace)
Project (proj:) defines the basic objects needed to describe projects and activities [F.1].

2.5.3 Documents (dx: namespace)
Document (dx:) defines the properties common to all documents associated with projects and organizations, e.g. document IDs and any other meta-data associated with documents [F.2].

2.5.4 Materials and Properties (mtrl: namespace)
Material (mtrl:) objects describe construction and process materials, associated material properties and thermodynamics [F.3].

2.5.5 Site (site: namespace)
Site (site:) objects that describe site information, facilities, facility systems, environmental data [F.4].

2.5.6 Unit Operation and Utility and Process Streams (uo: namespace)
Unit operation (uo:) includes objects that describe unit operation, conceptual specifications, and utility and process material streams [F.5].
2.5.7 Equipment (eq: namespace)

Equipment (eq:) defines the **data types and properties common to all facility equipment items and parts**, e.g. weight, environmental constraints, cost, etc [F.6].

[Table 12 Examples of schema definition files in the eq namespace]

2.6 Subject Schemas

Subject schemas are extensions of the core object schemas and provide details on specific engineering equipment items and accessories [G.1]. Examples of the subject domains are shown in Figure 1.

Subject Schema namespaces include:
- eqElec: electrical equipment types
- eqHvac: heating, ventilation and air conditioning equipment types
- eqHx: heat exchanger and fired equipment types
- eqRot: rotating equipment types
- eqSld: solids handling equipment types
- eqVesl: pressure and storage vessel equipment types

2.7 Collection-container Schemas

Collection-container Schemas are used to combine core and subject-specific engineering objects in various ways to support required data transactions and usage scenarios. These can be considered as data exchange ‘documents’ such as data sheets, equipment lists and catalogues [G.3].
Figure 2 Schema model layered structure and reusability of AEX schemas using an example of a centrifugal pump data sheet.

The data sheet includes the centrifugal pump data set, which is extended from a pump, which is extended from the rotating equipment set, and so on to the base object. The rotating equipment set is also used for equipment items such as centrifugal fans and compressors. Additionally, any data item, such as weight which is included as part of the core schema from which the rotating equipment set is developed, is available to be used to describe pump specifications. (See [R] for key.)
3 How to Build Applications to Use cfiXML LT

The following outlines considerations and steps for an application developer using cfiXML LT. The developer can pick and choose features to access details in appendices, as required.

Before starting it is essential to ensure that the scope and data exchange requirements are clearly established.

3.1 Background Understanding of cfiXML Schemas

The design of cfiXML LT is to enable developers with limited XML experience to be able to develop data exchange using XML serialization tools and XPaths provided in a data dictionary. LT features are described in [B], and more details are available in cfiXML Light (LT) Specifications.

3.2 Development Environments and Tools

A key reason for developing cfiXML LT is to enable the use of tools that allow:

1) development of applications much faster than using DOM or SAX parsers;
2) developers to minimize their learning curve and exposure to the intricacies of the schema and XML.

Examples of such tools are: XMLBeans (for JAVA), the XSD tool that comes with Microsoft’s Visual Studio (for C#, Visual Basic, Jscript and Visual J#) and Code Synthesis’ XSD tool (open source) (for C++).

These tools can auto-generate program code or data structures (classes in C++, C#, JAVA) that make it easy to read and write the XML data. A disadvantage of the tools is that they require you to rebuild the application code or critical components every time there is a change to the schema, which you do not have to do in general with an application that uses a DOM or SAX parser. It is possible to write applications using either approach that do not require recompiling or rebuilding when the schema changes in minor ways. The DOM or SAX approach, which loads the schema at run-time, can be designed to be highly tolerant of additions to schema such that customers and users can upgrade schema without needing any application upgrade.

3.3 Data Requirements

A domain specific data dictionary defining data items for a specified data exchange usage scenario, including associated XPaths, should form the primary reference and be provided in the public domain. [H].

3.4 Setting Up an XML Import and Export Interface Software Application – a check list

3.4.1 Establish the XML container to be used

Determine the XML exchange documents to be used for importing and exporting data items, or for the data exchange transactions in the usage scenario, and confirm whether or not the required collection-container schema is available [G.3].
If existing exchange documents do not meet requirements, consider defining a custom document schema definition file [P.2]. However, if the scope of data exchange is narrow then consider not using a container at all and only using one of the global elements defined in any of the schema definition files.

3.4.2 Ensure all required data is included
If the data items to be exchanged are not all covered by existing schema, determine the most suitable mechanism for extending schema [P]. Collaboration with the cfidev project may be useful if the application will be widely used.

3.4.3 Design for Substitution Groups and non-scalar data
If the data includes substitution groups [N.1], lists or simple tables [N.7] ensure that the application design can handle these.

3.4.4 Confirm choice of using LT schema
Do cfiXML LT features and support meet foreseen requirements, or may use of cfiXML Full be appropriate [1.5]?

3.4.5 Design stores and program interfaces
How is the information stored in the application? What programming interfaces are required to put information into application storage on import, and to get information out of application storage for export?

3.4.6 Implement application code
In LT a tool such as XMLBeans, MS XSD tool or Code Synthesis XSD can be used to auto-generate classes that include serialization functions that make import and export of the XML simple. The auto-generated classes make up the application’s data structure. This means the data dictionary effectively forms the mapping table with XPaths.

Ensure that the following schema requirements are me by the application code.

3.4.6.1 Element order
A valid XML file must use the correct element order. This is due to the use of ‘sequence’ content models throughout the schema [L.8.3]. One approach to achieve this is to “preset a frame” for the export interface [Q.8.1].

3.4.6.2 Parent element creation
When exporting a value, particularly when using an application interface that directly uses XPaths, all element nodes (ancestors) in the XPath before the element that contains the data, must be created in the XML. If any filters or qualifying attributes are required in the XPath or to correctly specify meta-data associated with the value, these must be included in the XML.
3.4.6.3 Supporting the alternativeOption attribute
If multiple alternative option elements are to be used in the XML, the application must recognise such elements as alternatives when importing data and must include these attributes when exporting. In particular, if only a single element instance exists, then the alternativeOption attribute should not be used. If there is more than one instance of exactly the same element, with no other or the same attribute qualifiers, then all such elements must include the alternativeOption attribute [N.3].

3.4.6.4 Maintaining existing XML data
When an application uses an XML resource that already exists or was created by another program, it is essential that the application does not corrupt or delete any data that existed before the application used it [Q.8.5].

3.4.6.5 Access to individual elements where multiple instances exist
cfiXML elements and attribute values must always be accessible through the use of a standard W3C XPath which does not include ‘not’ [Q.8.6].

3.4.6.6 Providing pick-lists
Implement a means to ensure that the user selects only valid enumeration values to maintain valid XML [Q.10].

3.4.7 Testing
The testing procedures clearly depend on the application created. Details of the criteria to be met by cfiXML LT-based application development, along with example point to point data exchange work flows, are described in the cfiXML Light (LT) Specifications.

Testing may involve import-export-import round-tripping, if an application accommodates the same data sets for import and export. Testing data exchange between different software applications and confirming all expected data are present provides comprehensive checking opportunities. A key point is that resulting XML documents are valid using at least 2 different standard XML validators.
4 How to Build Applications to Use cfXML Full

The following outlines considerations and steps for an application developer using cfXML Full. The developer can pick and choose features to access details in appendices, as required.

Before starting it is essential to ensure that the scope and data exchange requirements are clearly established.

4.1 Familiarity with cfXML Schemas

It is recommended that an application developer aiming to use full cfXML is familiar with the full schema and the contents of this guidelines document. The cfXML schemas are large, complex and designed to be used in multiple usage scenarios. The check list below emphasizes key tasks that need to be considered.

4.2 Development Environments and Tools

Besides an Integrated Development Environment (IDE) or language parsers and editors a DOM or SAX parser is required. The W3C web site includes a list of many tools and parsers available for a range of popular languages. Alternatively a search of the Web will yield a large list of the latest parsers and tools available.

4.3 Data Requirements

Software interfaces should support one or more well defined use cases. For each of these use cases the data to be imported from and exported to specific programs need to be established. Some tried and tested application developments are listed [Q.2].

If available, a data dictionary can be extended to provide all the fundamental data required to create cfXML interfaces [H].

4.4 Setting Up an XML Import and Export Interface – a check list

4.4.1 Establish the XML container to be used

Determine the XML exchange documents to be used for importing and exporting data items, or for the data exchange transactions in the usage scenario, and confirm whether or not the required collection-container schema is available [G.3].

If existing exchange documents do not meet requirements, consider defining a custom document schema definition file [P.2].

However, if the scope of data exchange is narrow then consider not using a container at all and only using one of the global elements defined in any of the schema definition files.
4.4.2 Ensure all required data is included
If the data items to be exchanged are not all covered by existing schema, determine the most suitable mechanism for extending schema [P]. Collaboration with the cfdxdev project may be useful if the application will be widely used.

4.4.3 Design for specific schema features
Full cfiXML makes additional demands on the application design compared to simple business XML schemas. The impact of the following cfiXML features are particularly important to consider:
- 1. Objects [E];
- 2. Substitution Groups [N.1];
- 3. Units of measurement [D.2.2, D.2.3];
- 4. Revisions and change tracking [E.6];
- 5. Lists and Tables [N.7, N.8];
- 6. The cfiXml.xsd File for Loading Schema into Software [Q.4];
- 7. Messaging Usage and Business Protocol Containers Section [Q.5]

4.4.4 Confirm choice of using full schema
Do full cfiXML features and support meet foreseen requirements, or may use of cfiXML LT be appropriate [1.5]?

4.4.5 Design stores and program interfaces
How is the information stored in the application? What programming interfaces are required to put information into application storage on import, and to get information out of application storage for export?

4.4.6 Design the mapping table
Design and document a mapping table between the data structures in the application and the XML for the data to be transferred. If a data dictionary is available, this can be extended [H.2]. The key contents will include:
   1) a column to define the storage location and function calls required to get or put information from and to the application storage;
   2) a column to define the XPath to the element in the cfiXML schemas that match the data items to be transferred.

XPath is a query language that provides a unique path definition to data contained within an XML instance file.

4.4.7 Implement the application code
Write application code for the import and export functionality using an XML parser with either a DOM or SAX application programming interface. Use the parser exchange data between an XML instance file, or in memory store, and your application’s storage according to the mapping table you defined in 4.4.6.

Ensure that the following schema requirements are met by the application code.
4.4.7.1 **Element order**
A valid XML file must include the correct element order. This is due to the use of ‘sequence’ content models throughout the schema [L.8.3]. One approach to achieve this is to “preset a frame” for the export interface [Q.8.1].

4.4.7.2 **Creating objects and managing cfiXML objects**
A cfiXML object must also include a minimum set of attributes. An export application must recognise each element that is an object and ensure that at a minimum the objectID and contextURL attributes are specified with valid strings [E.3]. Uniquely identifiable cfiXML objects enable both transient exchange of information and the maintenance of information in object repositories, ranging from a single file or small collection of files to large databases. [Q.6]

4.4.7.3 **Parent element creation**
When exporting a value, particularly when using an application interface that directly uses XPaths, all element nodes (ancestors) in the XPath before the element that contains the data, must be created in the XML. If any of these element nodes are objects, then they must be treated specially as given in the previous section. Also, if any filters or qualifying attributes are required in the XPath or to correctly specify meta-data associated with the value, these must be included in the XML.

4.4.7.4 **Supporting the alternativeOption attribute**
If multiple alternative option elements are to be used in the XML, the application must recognise such elements as alternatives when importing data and must include these attributes when exporting. In particular, if only a single element instance exists, then the alternativeOption attribute should not be used. If there is more than one instance of exactly the same element, with no other or the same attribute qualifiers, then all such elements must include the alternativeOption attribute [N.3].

4.4.7.5 **Maintaining existing XML data**
When an application uses an XML resource that already exists or was created by another program, it is essential that the application does not corrupt or delete any data that existed before the application used it [Q.8.5].

4.4.7.6 **Access to individual elements where multiple instances exist**
cfiXML elements and attribute values must always be accessible through the use of a standard W3C XPath which does not include ‘not’ [Q.8.6].

4.4.7.7 **Units support and issues with XSLT and direct XML access [Q.7]**
1) Converting engineering values from XML [Q.7.1]
2) Using XSLT or directly accessing XML values [Q.7.2]

4.4.7.8 **Supporting revisions, versions and change management [Q.9]**
Full cfiXML support is provided for revisions, versions and change management through a combination of elements and attributes. The developer can determine the level of functionality required.
1) Versions and transactions [Q.9.1]
2) Revisions [Q.9.2]
3) Change tracking [Q.9.3]
4) Full change tracking, revision and versions system [Q.9.4]

4.4.7.9 Providing pick-lists and allowing for user values [Q.10]
Implement a means to ensure that the user selects only valid enumeration values to maintain valid XML [Q.10].

4.4.7.10 Optional software implementation features [Q.11]
1) Read and write status [Q.11.2], based on objectState attributes [E.4.2].

(Support for User Access Control [Q.11.1], Mappings to other schemas [Q.11.3] and Default values and ranges [Q.11.4] are not included in schema).

4.4.8 Testing
The testing procedures clearly depend on the application created. This may involve import-export-import round-tripping, if an application accommodates the same data sets for import and export. Testing data exchange between different software applications and confirming all expected data are present provides comprehensive checking opportunities. Key points to look out for include:

1) Resulting XML documents are valid using at least 2 different standard XML validators.
2) The XML instance file can be thought of as a “database” in a text file. Therefore, it is important to preserve data that is in an imported XML instance file into the exported XML instance file, even if the data in the file is not used by the application.
3) Unique ObjectID’s are assigned to every object in an instance file.
4) Values for physical quantities are specified in the correct units, whether by default units or by using the ‘symbol’ attribute.
5) Data associated with a specific unique object must be entered once, and if needed again elsewhere in the file, then it must be referred to by setting the objectState attribute for the object making the reference equal to “refLocal”.

AEX cfiXML Schema Reference Guide 18
5 How to Develop and Extend Schema

The key principles of schema modeling and development, and dealing with engineering data, are outlined in the Appendix [K].

The majority of data in cfiXML are represented by elements [L]. Some simpleType enumerations may be used as ‘qualifiers’ rather than the primary data item. Some attributes are also used, either as qualifiers for the primary data item in a similar manner to the simpleType elements [L.9], or to support special concepts [N].

5.1 Becoming Familiar with cfiXML Schemas

Before starting work on schema development, developers should have at least a basic working knowledge of XML, XML Schema, XPath and object-oriented information modeling principles, especially the concepts of inheritance, containment and references. Outline of the design, structure and content of cfiXML are provided [1.2; 1.3; 2].

One effective way of becoming familiar with cfiXML schema structure is by following XPaths through a schema tree-view. For this purpose, a selection of XPaths is provided in J.2.

5.2 Data Requirements

Before extending schema, a developer needs to have clearly defined the equipment item requirements that are to be modeled. One of the key steps is to specify the work process that is being represented.

5.2.1 Data Dictionaries

Data dictionaries have been recognized as a key support for schema development and communication between stakeholders involved in developing electronic data exchange [H]. Example data dictionaries are available in cfiXML documentation for specific work processes [1.1.2].

For cfiXML LT: Data dictionaries defining data items for a specified data exchange usage scenario, including associated XPaths, should form the primary reference and be provided in the public domain. There should be no need to refer to the schema or schema reference guide to create cfiXML LT data exchange applications.

5.2.2 Representing Equipment Items in Schema

The schema developer needs to have a model of the physical components of the equipment item being defined [G.2]. In addition, a basic understanding of the different fluid streams (process and utility) to be represented is required [N.6].

The cfiXML schemas may already include detail for the equipment item and components involved. The developer will need to ensure that the new requirements fit into the existing model, or determine which additions or updates are needed.
5.3 Extending cfiXML Schemas

Extensibility is an important part of the schema design. The XML files validated against a schema must be able to include elements that were not included in the original design. It must also be possible for users to extend the schemas on local systems. The cfiXML framework supplies schema that can be used as is, or can be readily extended as required by two parties who need to exchange data that are not contained in the current schemas without having to edit the base standard schema files. Examples of how to undertake schema extensions are provided [S.5].

Extension of existing standard elements in the schema by editing the published xsd files is not recommended at any time, regardless of how the original XSD was structured or how the extension is done.

A robust system to allow schema developers to report issues and request changes to existing cfiXML schema is available. Details can be found on the SourceForge cfidev project. Opening an account with SourceForge allows access to the development website to report issues.

5.3.1 Extensibility mechanism without adding schema definitions

Extension mechanisms are provided in the schema design so that XML instance files may be extended without having to provide any additional schema definitions, as follows.

1) Each element “sequence” and “enumeration” list in the schema allows the user to insert additional custom elements, provided they are in “well-formed” XML syntax. This is the “customXXX” element method [P.1.1].

2) The enumeration list for each enumeration data type in the schema can be extended by using the “otherValue” mechanism [P.1.2].

The advantage of these approaches is that they are easy to use and do not require any schema change at all.

The disadvantage is that data validation is the responsibility of the receiving and exporting application software. No automated data validation by the XML parser is available.

5.3.2 Extensibility mechanism through user-provided schema definitions

Extension mechanisms are provided in the schema design so that XML instance files may be extended through user-provided schema extensions. This captures the benefit of XML parser validation for user extended schemas. There are three mechanisms provided [P.2].

5.3.3 Adding Equipment Extensions to Published Standard cfiXML Schema – check list

The following check list covers extension of the published standard cfiXML schemas. If you wish to include such developments into the published cfiXML standard, the procedures and review standards are described in the SourceForge cfidev project. This
ensures that all such extensions become public domain and part of a single widely used standard and not a limited private version of the schema. The procedures and review process ensure quality and adherence to the cfiXML schema development standards. If these procedures are not used, then the extensions will not be compatible with the published cfiXML standard.

The cfiXML schema has a large number of equipment types that have yet to be fully developed. The steps provided in the following section describe how to add to an existing cfiXML subject namespace.

If external objects are being linked, then the developer should review the description of these types of objects in [E.5.2]. Familiarity with the contents of core schema objects and facility items is assumed before starting to add new equipment items to the schema. Also, the use of at least 2 standard schema development environment tools is assumed.

5.3.3.1 Establish the XML container to be used
Is a new collection-container schema required [G.3] or is it appropriate to include the data exchange ‘document’ you require in an existing file? Follow existing examples for naming conventions and element content.

5.3.3.2 Review of existing schema
Are there any existing elements related to the ‘new’ items to be added?

1) Check through existing cfiXML core engineering schemas and the subject domain ‘xxBase.xsd’ files and note any elements which already exist and are apparently applicable to the ‘new’ items, i.e. any base sets of elements (complexType), simpleTypes or ‘individual’ elements which are apparently common to the new equipment item. Review data dictionary contents based on these findings [H.2.1].

2) Research existing simple types, focusing on enumeration lists. Any programmer’s editor can be used to search multiple cfiXML schema files for a few of the more unusual enumeration values in the set to be added. This should confirm whether a simpleType already exists which contains the enumeration values required, or some of the enumerations required. If a simpleType does exist with at least some of the enumeration values, then before adding new data, check:
   a. the existing use of the element in schema (where is it referenced), and whether there is any issue with the new enumeration content in these contexts;
   b. domain expert input and research to confirm whether the new enumeration values and the existing enumeration values form a single valid set.

5.3.3.3 Check conventions for naming elements, annotation requirements, naming files and version numbers [M].

1) Naming elements - Use of case [M.1.1]; no prefix or suffix [M.1.2]; Use of abbreviations, acronyms and units [M.1.3]; company names, associations [M.1.5].

2) Annotation requirements - xsd:appinfo and xsd: documentation [M.2].
3) **Naming files** – example provided [C.5.1]. **File headers** [M.3].
4) **Version numbers** [M.4].

### 5.3.3.4 Create types and elements [L]
Create types and elements to extend schema and model the new equipment item specifications. Elements contain virtually all the engineering data in cfiXML [L.2].

1) **Determine where to locate the types and elements** in schema [L.6].
2) **Add objects to the subject domain namespace** either in the ‘xxBase.xsd’ file or in a separate file [C.5.1]. Consider the physical form of the ‘new’ item. Decide whether the object should be based on **EquipmentItem**, **BulkItem** or **FabricatedItem**, or just obj:Object [L.10]. If any existing **complex types** are valid for the extension, add them.
3) **Use sequence and ordering conventions** [L.8.3].
4) When adding extension content to your newly created object or existing objects, add **complexType** elements [L.3].
5) Create **simpleType** enumeration lists [L.4]. Ensure that the enumeration list sets do not contain combinations of data of different types or sources. If possible, the different data should be divided into appropriately named simple enumeration types. Once the enumeration sets are confirmed as valid, add them to the schema.
6) Add **base ‘leaf’ elements** using the ‘ext’ namespace or the ‘pq’ namespace for the data types [L.2.2]. Any that already exist in schema, can be copied and moved into position. For ‘new’ elements, a detailed search for elements with a similar name should be undertaken through all the cfiXML schema files using a programmer’s editor.

### 5.3.3.5 Add attributes if required [L.9].

1) Using **conditions or qualifiers** – attributes are used for particular metadata [L.9.1].
2) **ReferenceProperty** and **ReferencePropertyID** – to enable a property to reference another property within a single document on which the primary property value is dependent [L.9.2].
3) **UsageContext** – to define whether the data of an element is associated with a particular usage scenario (e.g. Design, Operating Performance) [L.9.3].

### 5.3.3.6 Consider special modeling requirements [N].

1) The **substitutionGroup** – provides a means to access related objects, for example different drivers, which may be associated with the containing equipment item object [N.2].
2) **Use of AlternativeOption attribute** – to pick more than one enumeration from the same list or to allow different child elements to be considered as alternative options. [N.3]
3) **Key and keyref** – used in cfiXML within tables [N.4].
4) **Modeling reference relationships** - to relate objects [N.5].
5) **Material streams** – to consider process or utility stream flows through the plant facility [N.6].
6) **Order element or order attributes** - for array lists and simple tables [N.7N.8].
7) **PropertyTable** – for complex data tables [N.8].
8) **EquipmentConnection** – how equipment connections may be modeled [N.9].
9) **Life cycle status** - cfiXML schema provides a combination of elements and attributes which can be used to define the life cycle status of an object at any stage through this process [N.10 ].

**5.3.3.7 Determine whether or not elements need to occur more than once.**
If elements do not need to occur more than once, set the element occurrence to 0..1 (minOccurance of 0 and maxOccurance of 1) so that all elements are optional. If more than one instance of the element is required, use 0..n to indicate more than one can be used (unless the model defines a specific number required). If attributes are used on an element, multiplicity is commonly required.

**5.3.3.8 Write test xml**
Write test xml for a real-usage scenario and check that the xml is succinct, clear and well-formed [K.6.3]. If available, test in application being developed.

**5.3.3.9 Write XPaths**
Write XPaths for the most common usage scenario queries.

**5.3.3.10 Refine schema**
Refine schema based on results from writing the test xml and XPaths.

Consideration of whether new schema elements are all applicable to users of cfiXML LT should be undertaken [See cfiXML Light (LT) Specifications ]. Any schema not applicable for cfiXML LT will require processing by use of the LT Tool to mark for removal when release versions of the schemas are produced.
A. BACKGROUND, PURPOSE AND BENEFITS OF cfiXML

Electronic data exchange allows data to be reliably transferred between stakeholders without time consuming, costly, and potentially error prone manual transcription of data. AEX schemas provide a common language to enable stakeholders to share this information electronically.

A.1 Fiatech and the AEX Project

Fiatech\(^5\) is a non-profit, joint industry consortium. The Fiatech mission is to identify and accelerate the development and deployment of fully integrated and automated technologies to deliver the highest business value throughout the life cycle of all types of capital facilities projects.

The Fiatech Automating Equipment Information Exchange (AEX) Project\(^6\) was launched in 2002 with the objective of automating equipment design and delivery through software interoperability. AEX Project members include owner-operators, engineering companies, equipment suppliers and software suppliers. AEX has working relationships with industry standard groups, including the American Petroleum Institute (API), American Society for Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), Design Institute for Physical Properties (DIPPR), Hydraulic Institute (HI) and Process Industry Practices (PIP).

A.2 Information Exchange in the Capital Facilities Industry

The design, construction, operation, and maintenance of a capital facility involve the collaborative effort of many different companies and individuals. These enterprises include owner-operators, architectural and engineering, procurement and construction (AEC and EPC) contractors, equipment suppliers, and government bodies, all of whom are supported by commercial software and technology suppliers. These organizations collaborate in a complex network of work processes, commonly exchanging and sharing information through traditional paper or ‘electronic paper’ documents. Throughout the facility life cycle, this information is developed, used and reused in many different software systems in the different organizations. The resulting, usually manual re-entry of data is labor-intensive, time-intensive, subject to occasional costly human transcription errors, and adds significant cost and time to the creation, operation and maintenance of a capital facility. The solution is a system to reliably and efficiently share data electronically, i.e. interoperability.

A.3 High Business Value of Interoperability

Deployment of an interoperability solution will potentially save the capital facilities industry millions of dollars a year. In 2004, NIST reported that lack of software interoperability costs the capital facilities industry at least $15.8 billion annually\(^7\). A study undertaken by a large owner-operator company, its engineering, procurement and

\(^5\) www.fiatech.org
\(^7\) NIST GCR 04-867, August, 2004
construction contractors (EPCs) and suppliers, estimated annual benefits from interoperability of a 5% reduction in the cost of equipment procurement. Interoperability for other work flow processes would clearly also provide considerable benefits.

A.4 The AEX Solution for Data Exchange

A key part of the solution to the lack of interoperability has been recognized for many years. It requires industry-wide agreements on, and deployment of a software-independent electronic data exchange system, based on a standardized method of describing and storing data. Defining or ‘mapping’ data sets in different software applications in terms of a standard language enables information flow to and from multiple software systems within a company or with other companies (Figure 3). One interface written to the common language gives access to all compliant applications.

![Diagram of software data exchange via standardized industry file formats or database]

**Figure 3 Software data exchange via standardized industry file formats or database**

A.4.1 XML

The eXtensible Markup Language (XML) is an electronic text format, which was recommended by the World Wide Web Consortium (W3C)\(^8\) as an Internet Standard in 1998. Being text-based ensures that most software applications should be able to be defined in terms of XML. Additionally, text files will always be readable, so given the decades involved in a facility life cycle, data will always be accessible.

Like the HyperText Markup Language (HTML), XML is derived from the ISO 8879 Standard Generalized Markup Language (SGML). The concept of defining or ‘tagging’ a written text document is taken to powerful new levels by XML, in which numerous types of data are defined. These data types can be combined into the standard system for describing capital facilities industry information.

The foundation of all the Web and Internet, and associated development environments, tools and components are all based on tagged languages such as XML. Basing the

---

\(^8\) [http://www.w3.org/standards/xml/](http://www.w3.org/standards/xml/)
standard language for the capital facilities industry on XML has technological and economic benefits as numerous development tools are already available to support XML.

A.4.2 AEX Schema Development

XML specifies formatting rules for structured text files. The XML standard does not say what the information content is or how the information content is to be structured. So, in order for XML to be an effective technology for software interoperability, industry users need to agree upon common XML data types, structures and definitions, i.e. an electronic vocabulary. A collaborative approach is clearly required to be able to define the detail and design of an XML information model to be shared by multiple organizations and software systems in the many associated work processes.

The AEX Project first identified key high-value business processes and key equipment items. Subsequently, preliminary schema development focused on the equipment supply chain for centrifugal pumps and shell and tube heat exchangers. However, the schema structure was defined from first steps to anticipate the need to describe technical data about all engineered equipment and materials, and to be used across the entire facilities life cycle including front-end planning, design, procurement, construction, commissioning and startup, and operations and maintenance. As the use of the AEX capital facilities industry XML (cfiXML) for software interoperability has been tested and proven, the schema has continued to develop to provide more detailed coverage of other equipment items.

A.5 Demonstration of Real Usage Scenarios

Preliminary work of the AEX Project included analysis of equipment lifecycle work flow processes for major capital facilities (Figure 4). Information flows between each of the process stages. Software systems, introduced to improve data quality and reduce the labor and time involved, have resulted in ‘electronic documents’ becoming increasingly common throughout the information flow process.

The volume of information exchanged during equipment procurement, that is, the numerous exchanges between buyer and supplier in the inquiry/quotation process, was considered of primary interest to determine the effectiveness of the AEX schema solution.

The inquiry/quotation process commonly involves information passing between the purchaser, via a number of engineering, procurement and construction (ECP) contractors to multiple suppliers. Only one supplier and EPC may get the contract, so the cost for the other participants is significant. Each interaction between buyer and supplier represents one or more information transaction. Savings can clearly be made by reducing the time and effort made in performing each transaction. The use of a data exchange standard can substantially reduce the time and effort in each information transaction and improve data accuracy.
Figure 4 AEX Project analysis of equipment life cycles work flow processes for major capital facilities
A.5.1 Centrifugal Pump Procurement Demonstration

Using the example of the procurement of a centrifugal pump, key stages in the work process can be described. The first stages involve the sharing of technical pump data on system design, specifications, and performance and construction. The later stages of the process include the exchange of commercial data, such as equipment costs, life cycle cost evaluations, terms and conditions and delivery lead-times.

Two key demonstrations of electronic data exchange of real capital project data for the procurement of centrifugal pumps have been completed by the AEX Project. The most comprehensive involved an owner operator company, software suppliers and pump suppliers. A set of 66 data items was selected for the demonstration. For each information exchange transaction, the associated software program required an interface to import and export data from an XML file, as prepared by the software suppliers. All the information exchanged strictly adhered to the cfiXML standard and ensured that there was no ambiguity or possibility for human interpretation errors.

The exchange sequence is summarized in Figure 5 and consisted of the following key steps:

1) The owner operator manually entered basic data into an Excel® data sheet which included an extension to enable import and export of cfiXML standard XML. These data were exported in an XML file to the sizing program.

2) This system modeling program was run and then the resulting data imported back into the data sheet to view.

3) These data were then sent as standard cfiXML XML to a couple of different pump suppliers’ selection and configuration programs.

4) The suppliers’ programs were run, various pumps selected and the specifications for each exported.

5) The various sets of data from these different suppliers were again viewed in the data sheet and were also imported into an equipment procurement company’s BidTab analysis program.

---

9 (Demonstration of cfiXML in real usage scenario (compilation of slides); Electronic data exchange in the pump industry.pdf)
Figure 5  Data exchange sequence in the centrifugal pump procurement demonstration

A.5.2  Key Points from AEX cfXML Testing and Demonstration
1) The use of XML data exchange between different software programs works.
2) A standard set of XML schemas is essential.
3) All participants in the demonstration consider that full implementation of XML interfaces will be advantageous for their organizations.
4) The use of an XML interface does not require alteration of existing software and does not otherwise detract from existing software.
5) The software user does not need to know details of XML as the interfaces to import and export cfXML can be provided as background additional functionality.
B. cfiXML LT Features

The following text is taken from cfiXML Light (LT) Specifications Chapter 4, to provide details on each feature of cfiXML LT compared to the features of full cfiXML schema.

Developing cfiXML-based implementations using XML serialization tools, and the associated challenges, highlighted a few key design elements in full schema to assess. These included the use of polymorphic derivable types and the required object identifier. Following detailed consideration of potential changes to schema to simplify application development, it was agreed that full schema would be changed. This resolution ensured, for example, that LT schema would be compatible with full cfiXML schema. Therefore, LT and full schema no longer include polymorphic derivable types and as the replacement design structure (substitution groups) is common to both, it is not discussed further in this document. If any of the alterations made to full schema to support the use of XML serialization tools still require a different approach when a developer is using LT versus full schema, then these points are discussed below. The alterations to full schema associated with this project are included in cfiXML Schema Public Release v 4.0 and are described in the associated release notes10.

LT is a sub-set of full schema, so it is conceptually acceptable for an LT XML file to use the full cfiXml namespace. The LT cfiXML schema can be thought of as a convenience for implementers who are generating LT XML and as a machine readable means of defining the LT subset of full.

LT schema files include the namespace declaration ‘xmlns:aexLt=http://www.cfixml.org/Light/aexLt’ which can be used by software to identify LT xsd files.

To indicate that an XML file adheres to the LT subset schema, an additional, optional, Boolean attribute ‘isLightXml’ has been added to the objb:BaseObject.

Below, each cfiXML LT feature is described, with the rationale for it. If there are any points that an application developer should consider when deciding whether to base their software on LT or full schema, these are also outlined, with illustrative examples.

B.1 Optional ObjectID

The objectID attribute in cfiXML objects is altered from ‘required’ to ‘optional’.

Though this alteration occurs in full cfiXML, full schema should adhere to the schema guidelines and so objectIDs remain a required attribute [E.3.1]. This ensures consistency in all full cfiXML-based applications and resulting XML documents.

Unless parties involved in a data exchange sequence have agreed to include objectIDs, then it must be assumed that objectIDs will be omitted in cfiXML LT documents.

In addition to this fundamental change, the objectID attribute is no longer restricted by pattern to enable LT schema users to enter any string, if required. Again, full schema users should adhere to the requirements outlined in the guidelines to maintain a consistent approach [E.3.1].

10 See also MantisBT Support for the project at https://sourceforge.net/projects/cfidev/.
B.1.1 Rationale

LT Schema-based XML documents can be created with no reference or presence of objectID attributes in the document and still validate against both LT and full schemas using standard XML validators.

The format required to create unique objectIDs is specified in the schema guidelines: there is a defined pattern for users of full schema. A cfiXML LT user will not need to use guidelines.

An application developer will use schema LT without having to see or be concerned with objectID attributes.

B.1.2 Implications

Objects in cfiXML LT cannot be uniquely identified. For the development of data exchange systems typified by the work processes defined, object identification was considered by the XML serialization tools users as unnecessary in most cases.

The cfiXML object identifying attributes were designed to support a comprehensive data exchange system and associated applications. This might, for example, include an interface to an equipment process simulation program and reference to a local database of material or fluid properties. The cfiXML object can be uniquely identified globally through the use of the provided attributes.

A fundamental purpose of cfiXML objects are “to pass and maintain data on any single object throughout the life cycle of a facility item, from one domain expert to the next, across multiple usage scenarios. Schema object design also includes support to record and track data changes and maintain a full version and revision system. Without the capability of a persistent, object-oriented structure, such functionality becomes extremely complex and difficult.” [E.1]

Object Referencing

The full cfiXML mechanism to reference a specific object via its unique identification is not supported by cfiXML LT.

The full cfiXML mechanism can be used to locate persistent data that is held elsewhere, either in the same file, or in different files over the network without having to replicate the schema or the data.

a. Example (1) a process simulation program with a pre-existing database of fluid properties or material stream components used in the simulation might be referenced via the cfiXML full object mechanism.

b. Example (2) use of external XML standards to extend cfiXML is also facilitated by use of objectID attributes.

Compatibility of LT documents and Full schema-based Applications

Conversion or inclusion of data from cfiXML LT documents into full schema–based documents will be significantly (potentially prohibitively) more challenging for application developers due to the lack of identifiable cfiXML objects in the LT documents.
XML documents without objectIDs will not follow the full schema guidelines requirements. Applications requiring all XML documents to adhere to the full cfiXML guidelines will not be required to process XML documents which omit objectIDs if they make no claim of supporting cfiXML LT.

B.2 Revisions, versions and change tracking

The cfiXML LT schema defines all data items found in a typical procurement data sheet revision table. For example, this would include document revision identifier, date, description and the person who did the modification, checking or issuing.

All elements and attributes associated with full schema support for revision, versioning and change tracking at the object level, and not required in the above usage, will not be available in cfiXML LT.

B.2.1 Rationale

The primary purpose is to simplify cfiXML full. The elements and attributes associated with revisions, versioning and change tracking are mainly located in cfiXML objects. Additional attributes on all elements flag changes made to data. An application developer will use schema LT without having to see or be concerned with these schema components.

Versions of XML documents based on cfiXML LT can be managed in document management systems. Users in an organization will be able to create an audit trial using familiar in-house procedures.

B.2.2 Implications

Use of in-house management systems

When using cfiXML LT, parties who wish to manage different versions of documents, will use a document management system. The XML documents will not contain the information on who changed what and in which version or revision, as supported by full schema.

The consequences of the use of in-house document management systems in place of containing these data within the XML may include:

1) Change, version and revision information will only be available to the users of the in-house document management system and will not be available to any partners, associates, customers or other parties with whom data is exchanged. That is, the change, version and revision data are not being considered part of the data sheet information to be exchanged.

2) Audit trails of who made what change when will not be available if the data were exchanged. This then removes the benefits for legal purposes.

3) Contractors, suppliers to large organizations and other service providers may be required to maintain multiple document management systems to be able to be compatible with all their clients’ systems.
B.3 SI units of measurement

Only SI units of measurement are to be used for physical property values in cfiXML LT documents. Temperatures will be in Kelvin. No SI prefixes are to be used. For details see NIST reference (http://physics.nist.gov/cuu/Units/units.html).

B.3.1 Rationale

The rule that only SI units are to be entered into cfiXML LT documents results in significant simplification of application development.

Attributes used to support dynamic units of measurement in full schema can be filtered out to simplify schema structure. The removal of these attributes ensures that users do not have the option of using other unit sets, as provided in the full schema version.

B.3.2 Implications

Dynamic units of measurement are not supported by cfiXML LT.

In full schema each physical quantity type has a defined attribute ‘symbol’, which is used to set the units for the measured quantity and is recorded in the instance file. Objects include the attribute ‘unitSet’, which can be set to the name of a unit set such as ‘SI’, ‘USEngineering’ or ‘Metric’. This provides a means of specifying units for contained elements. The default unit set in full schema is ‘SI’ which ensures that if an application imports a cfiXML LT-based document, the units of measurement for physical property values can be correctly supported. [D.2]

B.4 Schema simplification

In addition to removal of elements and attributes associated with cfiXML objects, revisions, versions and change tracking, and unit of measurement support, particular schema files and schema components have been identified as not required for cfiXML LT. All items that were removed or altered to create the version of LT schema are listed in cfiXML Light (LT) Specifications, with rationale. LT schema downloads will always include release notes specifying these details.

Nillable: This feature is to enable recording of change from a value to no value as a user deletes. It is supported in full schema as low level functionality so that schema developers do not have to specifically implement nillable to every new element. It has been removed from cfiXML LT to simplify application handling of numeric data items.
C. NAMESPACES AND SCHEMA DEFINITION FILES

C.1 Rationale for Namespace Design

C.1.1 Summary

cfiXML is extensive. The division of schemas into separate parts dealing with distinct areas, and clearly and consistently named files, makes the task of documenting, explaining and understanding the schema much easier.

The use of namespaces allows a schema developer to:

- effectively identify and find schema elements;
- avoid conflicts in that schemas in different namespaces can contain the same element name.

Potentially, different working groups can work on different subject schemas. The use of namespaces therefore fundamentally supports schema development and extension. Application developers must be able to deal with this structure. However, development of namespaces is always carefully reviewed to ensure balance between the requirements of schema developers versus an application developer.

C.1.2 Detail

In principle, different individuals or groups can work on different subject schemas. We believe that the use of different namespaces by each of these groups will help prevent conflicts. Use of no-namespace, “chameleon” schemas is impractical in a cooperative effort of this magnitude. We have applied the same principle to divide up the core schemas into separate namespaces, where each namespace describes a different area and is contained in a separate file (or files sharing common initial letters).

1. The use of a single schema file to contain all core schemas in a single namespace is cumbersome and imposes too much overhead on XML referring only to a small part of the schema.

2. Dividing the core schema into separate parts dealing with distinct areas and in clearly and consistently named files makes the task of documenting, explaining and understanding the schema much easier. It also helps to identify and find schema elements - ext:Nm identifies the source of the element (file) and the type of the element (extension to xsd types, not a physical quantity, i.e. not torque).

3. If we accept the need to use namespaces and to divide up the schema files, we must have separate namespaces, even within the core schema:

- xsd can only import a single schema file into one namespace. If someone working on a subject schema needs to import two or three parts of the core schema, these must be imported into different namespaces. (They cannot be included since the subject schema will have its own namespace).
xml files (at least as implemented by MSXML parser) can only associate one schema file with a given namespace. From MS documentation:

XmlSchemaCollection.Add Method (String, String)

Parameters
ns The namespace URI associated with the schema. For XML Schema Definition language (XSD) schemas, this will typically be the targetNamespace.

uri The URL that specifies the schema to load.

Remarks
If ns has already been associated with another schema in the collection, the schema being added replaces the original schema in the collection. For example, in the following C# code, authors.xsd is removed from the collection and names.xsd is added.

schemaColl.Add("urn:author", "authors.xsd");
schemaColl.Add("urn:author", "names.xsd");

C.2 W3C Schema Namespace Declaration

AEX cfiXML is based on the schema specification from the World Wide Web Consortium (W3C). Therefore, all cfiXML schemas include the W3C schema namespace declaration:

<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>

The namespace abbreviation of “xsd” is used throughout the cfiXML schemas for consistency.

C.3 cfiXML Schema Namespaces

The cfiXML schema model uses uniquely named XML namespaces to allow multiple reusable core and subject-specific schemas to be used together in a single XML ‘Collection-container’ schema file (‘document’). All cfiXML namespaces consist of a string composed of the globally unique identifier URI “owned” by the organization developing the schema and the namespace short identifier.

Unique Namespace Name = URI + “/” + “short ID”

For the capital facilities industry XML, all namespaces belong to a common “root” URI, specifically “http://www.cfixml.org”. To the end of this common root, a short identifier is added, separated by a forward slash. For example, the “pq” (physical quantities) namespace would have a full unique identifier of http://www.cfixml.org/pq.

Figure 6 illustrates key capital facility industry namespaces and their relationships to each other. By convention, the short identifier tag (e.g. “pq:”) is used in a schema declaration to mean the same thing as “http://www.cfixml.org/pq” so that the XML files are more human readable, yet maintain uniqueness for the XML parsing program. In this document the file name, namespace prefix and full namespace qualifier will often be treated as synonymous, for convenience and simplicity.
C.4 cfiXML Schema Document Namespaces

The ‘Collection-container Schemas’ (Figure 2) are used to combine any of the core or subject schemas and so effectively provide the requirements for all data items to describe a document, such as a data sheet or catalogue. A similar method is used to uniquely name these namespaces, for example: http://www.cfixml.org/document/eqHxDoc for the cfiXML Schema “document” for heat exchange equipment.

![Diagram of namespaces](image)

Figure 6. AEX cfiXML Namespace Overview
(See [R] for key.)

C.5 XML Schema Definition (XSD) File Names and Associated Namespaces

The cfiXML schema definition files include the name of the associated namespace identifier. For example the main schema file that contains the elements in the mtrl (material) namespace is mtrl.xsd.
The layered schema structure and inheritance are applied when a large family of related namespaces exists, as indicated by the use of the same root namespace identifier as a prefix. A suffix is then added to create the specialized associated schema file.

C.5.1 Example of Related Namespaces and Subject XSD files:
“eq” (equipment) namespace
   Includes related namespaces of:
   “eqHx” (heat exchanger equipment)
     Includes related schema files of:
     “eqHxAc” (air-cooled heat exchangers)
     “eqHxSt” (shell and tube heat exchangers)
     Include related schema file:
     “eqHxBase” (common heat exchanger elements; developing schema)
   “eqRot” (rotating equipment)
     Includes related schema files of:
     “eqRotPmp” (pumps)
     “eqRotFan” (fans)
     etc.
   “eqBase” (common equipment item elements; developing schema)

This example shows the subject schemas of “eqRot” and “eqHx” as being extended from the core object schema file “eq” (Figure 6). The schema files within the subject schema namespaces include equipment item specific prefixes, e.g. “Ac” for Air-cooled”.

The “-Base” schema files
The “-Base” named schema files have two purposes. The first is as a ‘placeholder’ of subject - specific equipment types that have been created as an object of some form, but are not further extended. The second is as a repository for elements which are likely to be (or already are) used in more than one, higher level namespace.

Collection-container schemas
Collection-container schemas (Figure 2) are found in the cfiXML/schema/document folder and take the form of, for example, eqRotDoc.xsd as this represents a rotating equipment document.
D. CORE DATA TYPE SCHEMAS

D.1 Extended Data Types (ext: namespace)

D.1.1 Purpose
Extended Data Types (ext:) add base attributes to standard W3C XML Schema data types to support annotation and revision tracking at the element level and to define scalar (single value) and list (vector) data types.

The content of the data allowed in the elements is determined by the use of XSD types and may be further constrained by the application of pattern templates. An illustration of how ext elements are defined relative to the XSD base data types is shown in Figure 7.

![xsd:double](ext:Double)

Figure 7 Illustration of data type extension provided in the ‘ext’ namespace

D.1.2 Base Attributes for Elements and SimpleTypes
All types defined in ext may have one or more attribute group defined. Use of all attributes is optional. Key base attribute groups are:

1) **BaseAttributeGroup** is used for all data types in the ‘ext’ namespace and enables annotated, version and revision tracking at the data element level. The group of attributes includes: alternativeOption; comment; commentReference; changed; recipientToProvide; revChanged and usageContext. [Detailed in Table 1.]

2) **BaseEnumerationAttributeGroup** is for the creation of simple types and contains the same attributes as BaseAttributeGroup, along with the additional attributes such as otherValue and the xsd:anyAttribute namespace=“##other”. [Detailed in Table 1.]

3) **ListAttributeGroup** - used to support metadata for vectors.

The attributes comment, commentReference, changed, revChanged and usageContext are also available for objects. If an attribute such as alternativeOption is required for an element based on a complexType (whether an object or not), then they are added ‘manually’.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Purpose</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternativeOption</td>
<td>xsd:boolean</td>
<td>1) On a parent element (with multiplicity), this allows for alternative child elements to be selected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Allows for more than one enumeration to be chosen from the same simpleType enumeration list. [See N.3]</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Purpose</td>
<td>Multiplicity</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>comment</td>
<td>xsd:string</td>
<td>Allows addition of comments on an individual item.</td>
<td>No</td>
</tr>
<tr>
<td>commentReference</td>
<td>ext:ID</td>
<td>The unique identifier for the comment can be added.</td>
<td>No</td>
</tr>
<tr>
<td>changed</td>
<td>xsd:boolean</td>
<td>Provides a changed flag indicating that the data item has changed since the previous version (indicated by version number in objectID) [See E.6].</td>
<td>No</td>
</tr>
<tr>
<td>recipientToProvide</td>
<td>xsd:boolean</td>
<td>Indicates that the recipient of the XML is required to fill in this value.</td>
<td>No</td>
</tr>
<tr>
<td>revChanged</td>
<td>xsd:boolean</td>
<td>Indicates that the revision has changed. [See E.6]</td>
<td>No</td>
</tr>
<tr>
<td>otherValue</td>
<td>xsd:string</td>
<td>If the enumeration list does not include the required enumeration, then this attribute allows an ‘otherValue’ to be entered. [See P.1]</td>
<td>No</td>
</tr>
<tr>
<td>usageContext</td>
<td>xsd:string</td>
<td>Allows specification for the usage context to be set. [See L.9.3]</td>
<td>Yes if not fixed</td>
</tr>
</tbody>
</table>

Table 1 Summary of Base Attributes in the ext namespace

**D.1.3 Nillable**

In addition to extending W3C base types with the attributes described above, elements in schema are nillable. This enables the recording of change from a value to no value as a user deletes. Also, if a property is associated with the value of another property and there is no actual value entered into the latter, the use of ‘Nil’ enables the reference property to still be specified (see L.9.2).

All key ext element types, such as ext:Double, ext:String and ext:Integer, include the option of a value of ‘Nil’. This option is therefore available throughout schema wherever an element is based on one of these ext types. The ‘Nil’ value has been added to the data types by use of a union mechanism.

A few elements in schema are required (i.e. compulsory), for which ext base types with a suffix of ‘NotNillable’ are available.

As schema editors do not work well if simple type elements involve unions, simpleType elements have ‘Nil’ added to all enumeration lists.

*(Note: The W3C standard attribute xsi:nil has not been used as it cannot be implemented in type definitions or added with element references.)*
D.1.4 Use of ext: data types
Elements which define data types in cfiXML are ext: types (e.g. ext:String, ext:Boolean, ext:Double).

Example: To create a new element type I (integer) called ‘numberTubePass’, the element type is set to ext:Integer (Figure 8).

```
<xsd:element name="numberTubePass" type="ext:Integer" minOccurs="0" />
```

This element will then have the attributes from BaseAttributeGroup available. The XML instance of this element can be, for example:
```
<numberTubePass changed="true" comment="check tomorrow">2</numberTubePass>
```

Figure 8 Using ext: data types and Base Attributes

D.2 Physical Quantities (pq: namespace)
[cfiXML LT assumes all physical quantities are in SI so support for units of measurement is not provided.]

D.2.1 Purpose
In pq, physical quantity data types are defined for data elements that are associated with units of measurement. The units of measurement for each physical quantity are provided as enumeration lists as well as the default unit sets of SI, Metric and US Engineering. Physical quantity types are defined for scalar (single value) types, extended from ext:Double.

D.2.2 Symbol attribute
Each physical quantity type has a defined attribute “symbol”, which is used to set the units for the measured quantity and is recorded in the instance file. The permitted symbols for each pq: data type are defined in an associated enumeration type.

D.2.3 Unit sets
Default unit sets of ‘SI’, ‘USEngineering’ and ‘Metric’ are provided in pq. For each unit set, there is a corresponding complexType defined in pq.xsd that specifies the unit to use for each variable type, e.g. for SI the unit to use for acceleration is m/s².
Objects include the attribute ‘unitSet’, which can be set to the name of a unit set such as the default ‘SI’, ‘USEngineering’ or ‘Metric’. This provides a means of specifying units for contained elements, as stipulated by the following:

1) If no units are specified for an individual element, i.e. no symbol attribute is present, then the units for the individual unit are obtained from the units for the corresponding physical quantity in the unit set for the containing object.

2) If 1) applies AND the containing object has no unitSet attribute specified for the object, the unit set defaults to SI. Therefore every object, containing elements for which units other than SI are to be used, must specify the unitSet.

*Note: unitSet and all other similar object data is only applicable within the object itself and its child elements, but NOT to any contained objects. This is to support the use of objects that are not ‘fullXML’ (See E.4).*

If a set does not include the units for a physical quantity, then the units for that physical quantity are to be taken as the SI units (or the units in the original set from which the unit set is derived).

### D.2.4 Use

All elements in cfiXML which require units of measurement should be based on a pq: data type.

**Example:** To create ‘weight’ in the eq: namespace, the element type is set to pq:WeightMass (Figure 9):

```xml
<xsd:element name="weight" type="pq:WeightMass" minOccurs="0"/>
```

This element will then have the attribute symbol available. All pq data types also include the attributes from ext:BaseAttributeGroup. The XML instance of this element can be, for example:

```xml
<weight changed="true" symbol="lb" weightType="Dry">20.5</weight>
```

(Where weightType is an additional attribute which has been added to eq:weight.)

![Diagram](image)

*Figure 9 Illustration of use of pq: data types*

### D.2.5 Unit Symbol Strings

Names for units of measurement are standardized in pq to ensure consistency
**Rationale for Unit Symbol Strings**

Including symbols such as the degree sign and superscripts would be imposing implementation details related to reports and user interfaces, especially when multiple language support is required. We use standardized ‘names’ for symbols which implementation software can convert as required for display or printing as needed by the application software.

**Syntax**

The unit symbol strings defined in the pq namespace have a precise syntax to ensure that they can be automatically processed. Table 2 describes the syntax:

<table>
<thead>
<tr>
<th>Character</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td>Power. e.g. (m^2) indicates meters squared.</td>
</tr>
<tr>
<td>/</td>
<td>Division including for powers, e.g. (m/s) is meters per second. (Pa^{1/2}) is square root Pascals.</td>
</tr>
<tr>
<td>.</td>
<td>Dot or period: multiplication or separator for individual unit symbols to make up compound symbols: e.g. (N.m) as Newton meters.</td>
</tr>
<tr>
<td>()</td>
<td>Brackets may be included to apply a power to more than one symbol at the same time. Processors must be able to equate all valid options. e.g. ((ft^3/lbmol)^2) and (ft^6/lb^2) are both valid and equivalent. (But would both need to appear in the XSD to be valid in XML.)</td>
</tr>
<tr>
<td>_</td>
<td>Underscore may be used to separate parts of a single symbol name such as “cal_15”.</td>
</tr>
<tr>
<td>deg</td>
<td>Represents the degree symbol.</td>
</tr>
<tr>
<td>Diff</td>
<td>A suffix that indicates that the symbol is for a differential value. Absolute is the default and requires no additional specification.</td>
</tr>
</tbody>
</table>

**Table 2  Standard components of unit symbol strings**

**Unit Symbols**

All unit symbols that have been defined for use in cfiXML are available in pq.xsd.

**Common errors**

degK does not exist. The symbol for Kelvin is capital ‘K’ alone.
hr The correct symbol for hour is lower case ‘h’.
amp The correct symbol for Ampere is A
gm should be just g
Watt should be W
sec should be s

**Standard Prefix Multipliers**

To particularly ensure consistent capitalization the following Standard symbols should be used in symbol strings. These are the authorized SI prefixes. The only non-standard use within capital facilities industry XML is the substitution of “micro” for \(\mu\) to ensure that no problems occur with different character sets or having to escape the character.

<table>
<thead>
<tr>
<th>Power of 10</th>
<th>Name</th>
<th>Standard symbol</th>
</tr>
</thead>
</table>
Table 3 Standard prefixes for unit symbols

**D.3 Engineering Type Library (etl: namespace)**

**D.3.1 Purpose**

The etl collects *miscellaneous* data types, which are *not objects*, including simple shape geometries and electrical source and area classification specifications. As there is no dependency upon the ‘obj’ namespace when the ‘etl’ namespace is used, these data types are **reusable in all Core Object and Subject Schemas**. The types should therefore be commonly used in many engineering domains (see examples in Table 4)).

- **ConditionType**
  - Types of conditions, e.g., Damped, Load - full, Voltage - reduced (used as a simpleType or as part of an attribute group on an element)

- **ElectricalAreaClassification**
  - Classifies areas by safety requirements, e.g., Class I, Division 2, Group C

- **PropertyType**
  - Characteristic for reported material properties, e.g., Average, Normal, Maximum, Rated (used as a simpleType or as part of an attribute group on an element)

- **RectangularShape**
  - Extended from Shape – adds dimensions for describing rectangular block shapes

- **ServiceCategory**
  - Environmental characteristics of the service where an item is deployed, e.g., serviceClass, serviceT, serviceFluidType

- **ValueServiceType**
  - Type of service, e.g., Clean, Dirty (used as a simpleType or as part of an attribute group on an element)

Table 4 Examples of data types included in etl
E. cfiXML Objects

The cfiXML schemas can be used to pass and maintain data on any single object throughout the life cycle of a facility item, from one domain expert to the next, across multiple usage scenarios. Schema object design also includes support to record and track data changes and maintain a full version and revision system. Without the capability of a persistent, object-oriented structure, such functionality becomes extremely complex and difficult.

E.1 Object Basics

The W3C XML Schema standard handles the object-oriented modeling concepts of inheritance and containment well, but the built-in capabilities are limited in terms of accommodating the need to reference and use data objects. To meet this need, cfiXML provides schemas to define uniquely identified data objects that can be referenced and reused. The 2 object namespaces are:

1) objb: (Base Object)
2) obj: (Object extended from Base Object)

Except for the Core Data Types schemas, most of Core Object and Subject Schema content extends, or inherits from, objects.

A cfiXML object can be thought of as a “smart container” which can be accessed by its name - like understanding the contents of a box without having to open the box. Providing the mechanism to uniquely identify objects anywhere in the XML enables software to refer to and use these data objects (smart containers). So, data objects can reference other objects elsewhere in the XML or even other resources. For example, a reference to a specific object via its unique identification can be used to locate persistent data that is held elsewhere, either in the same file, or in different files over the network without having to replicate the schema or the data.

Another key concept is that objects can and commonly do contain other objects within them. In this way, objects of arbitrary complexity can be constructed to meet the needs of describing real world applications. For example, a centrifugal pump (object) contains the various parts of the pump (impeller, shaft, coupling, bearing, driver, etc.), all of which are also objects (see Figure 18). The data sheet object contains the centrifugal pump object, along with objects that describe the site, the project, the material streams etc.

The fundamentals that schema and application developers should be aware of are described below. This includes details for object identification and referencing for basic external data exchange transactions, with mention of a few key points for other usage scenarios. Details of the full support provided by cfiXML objects, for example the anticipated use of XML files for data storage in an organization’s internal database and access mechanisms, are described below.
Objects being managed within a set of cfiXML-compliant files may contain or refer to data (XML or other) in formats, which do not themselves, comply with cfiXML standards. These are called external objects and are also supported by cfiXML.

**E.1.1 Rationale for Object Design**

We believe strongly that having a consistent set of attributes and elements for all objects of similar type is essential. In particular it is necessary to have an objectID attribute and a set of elements that capture such engineering properties as the status and revision history of objects. These can be defined as groups of elements and attributes which can be applied to all defined elements. However, we believe that using inheritance from a common “ancestor” Object is the best and simplest mechanism for doing so:

1. It is the simplest to use. Top-level elements may inherit through several intermediate objects: base object, engineering object, generic equipment item, generic heat exchanger, etc. To create new attribute and element groups for each level and then apply these consistently to the top-level elements is more complicated than using the inheritance mechanism.

2. Using obj with external namespaces is no more complex than any other mechanism. If we wish to add the engineering attributes to objects from other namespaces then a “hybrid” schema which combines cfiXML and other schema is necessary. In such cases, obj can just as easily be used as the base type from which such hybrid objects are derived.

3. Use of a common ancestor allows for variable content containers. For example, a list of equipment items can be defined. Any object which has inherited from the generic equipment item object can then be included in the list, identified by type. If there is no common ancestor, all possible equipment types must be included in a “choice” element, making maintenance far more troublesome.

**E.2 Object Namespaces (objb: and obj:)**

**E.2.1 Purpose**

objb: defines the Base Object, which includes the fundamental attributes and base element data required to identify, name and manage objects that can be defined and referenced by other elements and objects in the schema. All other cfiXML objects are derived from objb:Object.

obj: extends the Base Object to provide the capabilities for recording object transactions, e.g. software systems and persons who made a transaction, the details of the transaction, revision notes, tracking current object type and status. All cfiXML objects, such as equipmentItem and dataSheet, are based on obj.

**E.3 Management – Unique Object identification and Referencing Attributes**

[Not relevant to cfiXML LT application implementations that do not use objectID.]
The following details object elements and attributes provided to support all anticipated usage scenarios for cfiXML data exchange and maintenance of data repositories.

With the exception of objectID [E.3.1], all object attributes are optional.

Attributes in objb:BaseObject are used to identify objects and determine relationships between objects. The most important of these are in bold in Table 5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectID</td>
<td>ext:ID</td>
<td>These attributes allow object to be interrelated and managed in application software.</td>
</tr>
<tr>
<td>objectState</td>
<td>EObjectState</td>
<td></td>
</tr>
<tr>
<td>contextURL</td>
<td>ext:URINilA</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>ext:CRCVersion</td>
<td></td>
</tr>
<tr>
<td>revision</td>
<td>xsd:string</td>
<td></td>
</tr>
<tr>
<td>directReferenceURI</td>
<td>xsd:anyURI</td>
<td></td>
</tr>
<tr>
<td>organizationID</td>
<td>xsd:string</td>
<td>Universal unique identifier for organization that owns the object</td>
</tr>
<tr>
<td>contentType</td>
<td>xsd:string</td>
<td>Four attributes optionally determine the contents of an object</td>
</tr>
<tr>
<td>approvedNameSpace</td>
<td>EApprovedNameSpace</td>
<td></td>
</tr>
<tr>
<td>nameSpace</td>
<td>xsd:string</td>
<td></td>
</tr>
<tr>
<td>xmlContent</td>
<td>EXMLContent</td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Object Identification Attributes in objb:BaseObject

The combination of objectID, contextURL and version are sufficient to uniquely identify an object on a global level and to implement the software to retrieve it.

directReferenceURI is an alternative to contextURL.
directReferenceURI is an URI to an object that is directly addressed (type xsd:anyURI)

E.3.1 objectID

For cfiXML LT, objectID is optional. Schema no longer constrains objectIDs [B.1].

To be compliant with standard cfiXML Full, all objects require an objectID: it is the key internal identifier. This ensures consistency in all full cfiXML-based applications and resulting XML documents.

The format of the objectID is defined through the use of the ext:ID data type extension, as below.

If an application is only to support cfiXML Full, then quick alterations to the provided schema can be used to reinstate the constraints on objectIDs. These are:

1) in ext.xsd, comment out existing simpleType ObjID, create a copy of simpleType ID and rename (name and id) to ObjID;
2) in objb.xsd, replace `<xsd:attribute name="objectID" type="ext:ObjID" use="optional"` with `<xsd:attribute name="objectID" type="ext:ObjID" use="required"` (Occurs twice.)

Format: namespacePrefix.type.idOrName
The delimiter is period “.” between the parts

**namespacePrefix:** The cfXML namespace identifier or prefix.

**type:** The cfXML complex type of the object. Therefore, this will always begin with a capital letter.

**idOrName:** May include letters, numbers, hyphen, underscore and colon characters.

**Example:** `proj.Project.BrownfieldUpgrade209515930`

---

**Rationale for objectID**

The use of namespacePrefix and type as the first two parts of the objectID ensure that the same code or name can be used for different types of data, where use of the same value will not cause problems and may be necessary. For example a pump may use the ID A921, which may be the same as the code for a project.

The type is used, rather than the element, because elements with different names may still refer to the same object. Provided all these elements have been defined from the same type this should not cause any problems to the processing software. Use of namespacePrefix plus type is analogous to the use of xsi:type in variable content containers. This therefore ensures that an application has in principle all the information it requires to be able to process the XML appropriately.

`ext:ID` is NOT based on xsd:ID because multiple instances of the same objectID may occur in a file, where a single object is being reused. Therefore uniqueness cannot be enforced.

---

**E.3.2 contextURL - Object registration**

If XML may be used outside of the owning organization, or XML is likely to be imported from other organizations, use of contextURL (‘registration’) ensures that objects remain distinct.

The contextURL attribute is a unique identifier that is specified by the creator of the XML object and is recommended to include the URL of the organization that is creating and owns the object. e.g. www.aCompany.com/Project101

The combination of objectID and version of all objects that have the same contextURL must be unique. However, two objects with different contextURLs can have the same objectID.

---

**Rationale for Object registration**

Object registration provides a mechanism for referencing objects across files that is not supported by W3C Schema but is an important requirement of a coherent data resource. Also, by providing a registry or directory (however implemented), where the physical location of an object is recorded, allows the location of objects to be changed with only an update of the registry, and not all the references to the object that may be distributed through a vast collection of XML resources.
E.3.3 organizationID – ANSI registration
An additional, optional attribute organizationID provides a unique organization identifier for the organization that owns the data object. Under this approach a company obtains a unique identifier from the registrar, ANSI (see ANSI, “Procedures for registering organization names in the United States of America under the Joint-ISO-CCITT arc”, http://www.ansi.org/other_services/registration_programs/reg_org.aspx?menuid=10).

E.3.4 Version
The optional version attribute provides the final detail to ensure an object is uniquely identifiable throughout the life cycle of the object and associated data during exchange and/or storage. The version of the object is represented as a randomized numeric and character string of the type ext:CRCVersion.

E.3.5 Addressing
Objects are always identified by their objectID attribute. There are 3 ways in which an object can be addressed:
1) Locally within the same resource, such as in the same document or file.
2) Via a registry, directory or other implementation specific resource that uses an ID to obtain a resource.
3) Directly using a URI.

E.4 Object Relationships – Contained and Referenced Objects
[Not relevant to cfiXML LT application implementations that do not use objectID.]

It is common within engineering data to reuse the description of an item in many different locations, such as the same stream being referenced by the source and destination unit operations in a flow sheet. The cfiXML object can be referenced, or specified in place (contained). Whether an object is contained or referenced is not dictated in the Schema but can be flexibly implemented when the XML is being generated.

Therefore, objects in an XML file are either root objects, corresponding to the root element of an XML file, or contained objects, which are elements nested within the XML of the root element. Contained objects may themselves contain other objects, nested deeper within the XML. The XML describing contained objects may exist within the XML of the root object or may exist in a different file or different part of the same file. In the latter two cases, the XML of the root element will merely contain the address of the contained object via a reference to the object.

Since an object may be only a reference and the referred to object may be edited and changed by other software or people, there is no means to control the way that objects may be contained in one another. It cannot therefore be assumed that the object data associated with the root object will be inherited by a contained object, e.g. the attribute of a parent is not inherited by a child object.
E.4.1 Rationale for Contained and Referenced Objects

The “Requirements and Constraints” includes the following two requirements:

1. It must always be possible to send a single file or document of XML that is complete and contains no references to other XML.

2. An XML document should never contain more than one copy of the XML data of an object.

By allowing containment of the full XML for an object or a reference to a location that contains the full XML, both of these requirements can be fulfilled. The contained object or referenced object mechanism, selectable at XML generation time, provides a flexible way to support anything from simple transactions of complete XML data through to complex database or distributed storage XML based repositories that describe a facility through its whole life-cycle.

An alternative option would be to use a separate mechanism to refer to objects. However, this has been rejected in favour of using an element, which is empty but keeps the same attributes as the full, contained object. Reasons include:

1. Re-use of contained objects is necessary to avoid repetition of data. Therefore a reference to an object must be able to take the place of a contained object without invalidating the XML; adding a second mechanism complicates the structure of the schema.

2. Using a single mechanism, which allows either for reference or containment, ensures that a single valid XML file can always be constructed which includes all relevant data. There is no necessity to provide multiple files and assume processing in some unspecified software.

3. The type of object to which reference is being made is always enforced.

4. When a reference to an object is contained in the XML data but the reference cannot be resolved at a particular time, such as not being connected to a network at home, then the Obj generic description attributes, or meta data, about the referred to object will still be available. This means that, say, an XSLT template applied to generate HTML will include a meaningful description of the referenced object instead of a cryptic identifier.
E.4.2 objectState

In addition to the required objectID, and recommended option of contextURL, the objectState defines whether the object is fully contained or a reference. (see Table 6).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectState</td>
<td>enumerations:</td>
<td>The state of the xml contained in the object.</td>
</tr>
<tr>
<td>fullXml</td>
<td>(Default)</td>
<td>fullXml: The complete XML for the object which may be edited. E.g. it is a checked out copy when version control is being used.</td>
</tr>
<tr>
<td>refLocal</td>
<td></td>
<td>refLocal: The referenced object is contained in the same resource or files. It is located by its objectID.</td>
</tr>
<tr>
<td>deltaXml</td>
<td></td>
<td>deltaXml: The difference between the previous version of the object and the current version. Only the changes are recorded. This copy may be edited. I.e. it is a checked out copy when version control is being used.</td>
</tr>
<tr>
<td>external</td>
<td></td>
<td>external: Reference via a URI to an external resource.</td>
</tr>
<tr>
<td>fullXmlReadCopy</td>
<td></td>
<td>fullXmlReadCopy: full XML which is a copy that may or may not be up to date with the master instance of the object. The purpose is to provide a local version of an object for reading ONLY. It must not be edited.</td>
</tr>
<tr>
<td>refUri</td>
<td></td>
<td>refUri: Reference via a URI to a cfiXML object.</td>
</tr>
<tr>
<td>refViaReg</td>
<td></td>
<td>refViaReg: Reference via a registry</td>
</tr>
</tbody>
</table>

Table 6 objectState attribute enumerations

Unregistered objects (i.e. those without a contextURL) may be referenced from within another object, but such a reference should only be local and ideally temporary and must be considered unreliable. If the XML file for the contained object is moved then the original reference will no longer be valid and if the XML for the root object is sent to a different location, the physical path to the contained object may no longer be valid, even if all necessary XML files are sent together.

Details of all valid attribute combinations for contained objects are provided in Table 7.

<table>
<thead>
<tr>
<th>objectState</th>
<th>contextURL</th>
<th>directReferenceURI</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullXml*</td>
<td>Optional</td>
<td>No</td>
</tr>
<tr>
<td>deltaXml</td>
<td>Optional</td>
<td>No</td>
</tr>
<tr>
<td>external</td>
<td>Optional</td>
<td>Yes</td>
</tr>
<tr>
<td>refUri</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>refLocal</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>refViaReg</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 7 Valid Attribute Combinations for Contained Objects
E.5 Handling non-cfiXML or customized cfiXML Objects

[Not relevant to cfiXML LT application implementations.]

E.5.1 Object Contents

To support other XML standards, Object has four attributes which allow for various types of content.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentType</td>
<td>xsd:string.</td>
<td>Allows cfiXML objects to contain other information in the future such as image/svt+xml to support images contained in cfiXML.</td>
</tr>
<tr>
<td>nameSpace</td>
<td>Full path namespace to an XSD namespace. xsd:string</td>
<td>Provides support for miscellaneous XML for which there is schema.</td>
</tr>
<tr>
<td>approvedNameSpace</td>
<td>Enumeration: EApprovedNameSpace</td>
<td>To provide support for approved namespaces such as for PIDX, ebXML or other international standard XSD.</td>
</tr>
<tr>
<td>xmlContent</td>
<td>EXMLContent: approvedXML cfiXML* cfiXMLCustom otherXML</td>
<td>approvedXML: Supports other approved or recommended XML such as PIDX for company information that is to be processed by cfiXML compatible software. cfiXMLCustom: Indicates that the XML is cfiXML with user extensions that use the cfiXML extension mechanisms. Such XML will be processed by cfiXML compliant software. otherXML: Covers other XML that will NOT be processed by cfiXML compatible software.</td>
</tr>
</tbody>
</table>

**Table 8 Object Content Attributes**

<table>
<thead>
<tr>
<th>xmlContent</th>
<th>contentType</th>
<th>approvedNameSpace</th>
<th>nameSpace</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfixml*, cfixmlCustom</td>
<td>text/xml*</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>approvedXml</td>
<td>text/xml*</td>
<td>Required</td>
<td>No</td>
</tr>
<tr>
<td>otherXml</td>
<td>text/xml*</td>
<td>No</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Table 9 Valid Combinations of Content Attributes for Objects**
(Defaults marked with “*”.)

**Example XML**

`<ctx:egCompany objectState="refUri" xmlContent="approvedXml" nameSpace="http://api.org/pidx/company" directReferenceURI="\Dim4100\Data(D)\Xynchron\CoList.xml#xpointer(id(AC ompany))"/>`
E.5.2 External (non-cfiXML) Objects

Basics
Objects being managed within a set of cfiXML-compliant files may contain or refer to data (XML or other) in formats that do not themselves comply with cfiXML schemas. These are called external objects.

Such data can occur in a variety of ways:

1) XML may include user extensions to the schema, which allows for additional elements or enumeration values. Such objects are to all intents and purposes cfiXML objects but still require special care in processing by software.

2) XML data that conforms to valid schema from other namespaces can be incorporated directly into XML, which would otherwise be completely cfiXML compliant. These are hybrid objects (see below), which can be treated at the object level as a valid cfiXML object but where the contents of the object will not conform to cfiXML schemas.

3) XML or non-XML data may be relevant which has not been modified in any way to conform with cfiXML schema or even cfiXML conventions. These are alien objects (see below).

E.5.3 Hybrid objects
Hybrid objects occur where both cfiXML and other standard, but non-cfiXML, schemas have been used to create XML. This can be done in two ways:

1) by referring directly to the non-cfiXML schema in the XML data using the directReferenceURI attribute, or

2) by importing a schema from a different namespace into the compound schema used to associate cfiXML objects.

In either case the root of the XML file or the top-level element for a contained object is still a valid cfiXML object element, with objectID and all other attributes as necessary. Therefore these objects can be contained in cfiXML data and can also be referenced just as if they were normal cfiXML objects.

In order to “flag” the contents of such objects as potentially troublesome for cfiXML-compliant software, such objects must include the xmlContent attribute set to either “cfiXMLCustom”, for hybrid XML that uses standard cfiXML extension mechanisms (described below) or “approvedXML” for XML that is cfiXML approved but which does not use the cfiXML extension mechanisms.

Example XML file
File which combines standard cfiXML schema with an external, non-cfiXML compliant schema.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<equipItem objectID="eq.Bucket.b101" objectState="fullXml" xmlContent="cfiXmlCustom"
```
Example XSD file
Non-cfiXML compliant schema defining elements of hybrid object - eqBkt1.xsd

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema targetNamespace="http://www.proprietaryschema.org/bucket"
xmlns="http://www.proprietaryschema.org/bucket"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.00.0000">
    <xsd:element name="volNormal" type="xsd:decimal"></xsd:element>
    <xsd:element name="height" type="xsd:decimal"></xsd:element>
</xsd:schema>
```

E.5.4 Alien objects
Alien objects are by necessity external. The data cannot be contained within a valid
cfiXML file either because it belongs in its entirety to a different namespace or because it
is not XML-compliant. To relate this data to cfiXML data as if it were an object, an
empty object is used, which contains all necessary information about the object in the
object attributes.

```xml
<dxEqExample xmlns="http://www.cfixml.org/eq" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cfixml.org/document/eqRotDoc eqRotDoc.xsd">
    ...
    <equipItem objectID="obj.External.b101" objectState="external"
        contentType="text/xml" xmlContent="otherXml"
        nameSpace="http://www.proprietaryschema.org/bucket"
        directReferenceURI="C:\B101.xml"/>
    ...
    <equipItem objectID="obj.External.b102" objectState="external"
        contentType="application/vnd.ms-excel"
        directReferenceURI="C:\B102.xls"/>
    ...
</dxEqExample>
```
The objectID for external objects should use the obj.External for the namespace and type parts.

References can then be made to the external objects elsewhere as if it were cfiXML. Therefore a reliable and robust method to handle external objects is to create a “placeholder” XML file that simply contains the content type and location information for the data. This object is then referred to whenever the data are required in another set of data. This placeholder could also optionally use the standard set of object or equipmentItem elements to define some cfiXML-compliant data about the alien object.

**Example XML file**
```xml
<?xml version="1.0" encoding="utf-8" ?>
<bucket xmlns="http://www.proprietaryschema.org/bucket"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsi:schemaLocation="eqBkt0.xsd">
  <volNormal>1.0</volNormal>
  <height>2.0</height>
</bucket>
```

**Example XSD file**
eqBkt0.xsd
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema targetNamespace="http://www.proprietaryschema.org/bucket"
xmlns="http://www.proprietaryschema.org/bucket"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.00.0000">
  <xsd:element name="bucket">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="volNormal" type="xsd:decimal"></xsd:element>
        <xsd:element name="height" type="xsd:decimal"></xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**E.6 Revisions, Versions, Transactions and Change Management**

One of the key functions supported by obj is the means of recording data transactions and revisions. This is supported by obj:BaseObject attributes, as well as elements found in obj:Object, as outlined in Figure 10 and Table 10.
Figure 10  Key object components for recording data transactions  
(See [R] for key.)

BaseObject   Includes version, revision, changed and revChanged attributes
Object       Includes one or more transaction elements, previousVersion element.
Transaction  Each transaction includes elements defining the transaction type, 
version, revision date/time, person, etc.

Table 10  Object components for recording data transactions

These key components enable applications to be able to record when there is a change in 
the version of an object and maintain previous versions. Details of the person 
responsible, the type of change (e.g. Created, Changed, Issued etc.) and date and time can 
be linked with the specific version. Revisions, which may include one or more object 
versions, are also supported. The final component to enable a full change tracking and 
version management system is provided by the optional Boolean attribute “changed” 
found in every element in cfiXML (as part of the ext:BaseAttributeGroup).

Details on how applications can use cfiXML for recording and maintaining revisions, 
versions, transactions and change tracking are provided [Q.9].

E.7  Capital Facility Equipment Special Concepts

The cfiXML object includes a few key elements and attributes which support particular 
concepts associated with capital facility equipment requirements. These include 
usageContext and lifeCycleStage. Details of these schema components are included with 
considerations of how equipment item specifications are modeled in cfiXML (see L.9.3 
and N.10).
F. CORE OBJECT SCHEMAS

The Core Object Schemas represent a number of key areas which are reusable and applicable across multiple subject domains.

In general, engineering objects contain a large number of individual engineering data elements that are typically grouped together in logical ways. For example, in Figure 2, an eq:FacilityItem is a type of Object that contains a number of elements that describe the facility item, such as the element weight. There are many types of physical items in a facility, but all of them can have the common characteristic of having one or more weights associated with that item.

F.1 Context and Projects (ctx: and proj: namespaces)

Context (ctx:) objects provide details on the context for data, location, organization and person.

Project (proj:) defines the basic objects needed to describe projects and activities.

Since these context and project objects are generally fixed and version tracking is not needed, most are derived from BaseObject, not Object.

F.2 Documents (dx: namespace)

Document (dx:) defines the properties common to all documents associated with projects and organizations, e.g., document IDs and any other meta-data associated with documents.

When creating specific documents for given usage scenarios or a particular corporate environment, a document object from dx can be extended with objects from other core or subject namespaces (see G.3 and DataSheet example below).

DataSheet The dx:DataSheet object is an extension of obj:Object with the dx:DataSheetHeader element, to allow project and document metadata to be recorded. Collection-container schemas [G.3] representing data sheets are extended from dx:DataSheet.

AssociatedDocument Extended from obj:Object to include details to identify and provide documents associated with the primary data exchange document (e.g. document location, version, status, issueDate).

AssociatedDataSheet Extended from dx:AssociatedDocument to identify and provide data sheet documents supporting the primary data exchange document (e.g. type of equipment item the associated document describes, type of data document).

SupportDocument Extended from dx:AssociatedDocument to identify and provide details of supporting documentation associated with the equipment and components described in the primary data document (e.g. material certificates, operation manuals).
EquipmentDocument Extended from dx:SupportDocument to specify requirements, identify and provide supporting documents for the specific equipment and bulk items (e.g. number and when required, retention times, material certifications).

AlternativeDocumentID An element to provide alternative document identifiers or numbers used for the primary data document.

Table 11 Examples of dx: data types to describe and identify documents
Note that the use of objects ensures that object referencing is available to users of cfiXML Full.

F.3 Materials and Properties (mtrl: namespace)

Material (mtrl:) objects describe construction and process materials, associated material properties and thermodynamics.

The mtrl namespace defines materials used in process streams, stream or fluid properties as used on equipment data sheets, or materials of construction.

Stream or fluid materials can be made of multiple components (each normally a single chemical species) and mtrl defines the relationship between a complex material or material mixture and its constituent components. Each material (or component) can also have one or more properties associated with it. Numerous properties are defined in mtrl, as well as the structures and elements to describe more complex data, as used by thermodynamic specialist and process engineers. This includes details of experimental or calculation methods, both simple and complex property tables associated with various experiments and calculations, and material stream property tables.

The mtrl namespace also defines reactions and their parameters, for use in defining reaction-based properties. Defining chemical reactions is needed to support the definition of chemical reaction properties such as reaction equilibrium constants and heat of reaction. Some of the key schema components of mtrl:xsd are outlined in Figure 11.

Materials of construction are defined in mtrl in terms of material types based on commonly used descriptions, standard organization material codes, structure, processing and properties. Example XPaths are provided [J.2.5].
Figure 11 Outline of some of the key components of mtrl:xsd  
(See [R] for key.)

F.4 Site (site: namespace)

Site (site:) objects that describe site information, facilities, facility systems, environmental data.

The site namespace defines the context information relevant to a facility. It is used for facility data at a relatively high level and uses elements from the core schemas. The key schema components of site:xsd are outlined in Figure 12.

Figure 12 Outline of key components of site:xsd  
(See [R] for key.)
F.5  Unit Operation and Utility and Process Streams (uo: namespace)

Unit operation (uo:) includes objects that describe unit operation, conceptual specifications, and utility and process material streams.

The uo: namespace was originally defined for only unit operation and conceptual specifications. During schema development and in particular with the development of the usageContext attribute [L.9.3], some data types which could be used in other contexts, have been removed from the unit operation schema. The concept of streams, as originated in uo, were thought to be an effective method to represent material stream flows wherever and whenever required, e.g. from conceptual simulation to measured conditions in a plant. The related elements, derived from the purely unit operational data types remain in the uo: namespace.

F.6  Equipment (eq: namespace)

Equipment (eq:) defines the data types and properties common to all facility equipment items and parts, e.g. weight, environmental constraints, cost, etc.

The ‘eq’ namespace schema definition files include:

- eq.xsd Serves as a namespace placeholder, including all files.
- eqBase.xsd Contains reusable types for multiple eq subject areas and acts as a ‘placeholder’ for undeveloped equipment types.
- eqActuator.xsd Contains information about actuators.
- eqInstrBase.xsd Contains reusable types related to instrumentation and acts as a ‘placeholder’ for undeveloped instrumentation equipment types.
- eqPvfValve.xsd Contains details on valves, including block and control valves.
- eqPvfBase.xsd Contains reusable types related to pipes, valves and fittings & is a ‘placeholder’ for undeveloped pipe, duct, valve and fittings

Table 12 Examples of schema definition files in the eq namespace

The eqBase.xsd file includes 3 basic types of facility equipment items, namely EquipmentItem, BulkItem and FabricatedItem (as defined in Table 13). These basic equipment types are used for the creation of all subject specific schemas relating to facility items, possibly via intermediate “generalized” extensions. For example, a centrifugal pump is derived from a generic eqRot:Pump, an extension of eqRot:RotatingEquipmentItem, which in turn is derived from eqRot:BaseRotatingEquipmentItem and eq:EquipmentItem according to the following inheritance hierarchy (and also illustrated in Figure 2).

```
obj:BaseObject
  |__/ obj:Object
  |__/ eq:FacilityItem
  |__/ eq:EquipmentItem
      |__/ eqRot:BaseRotatingEquipmentItem
```
In early versions of schema instrumentation and pipe, valves and fittings were in separate subject schemas. However, piping connections and instrumentation are both closely associated with the specification of major equipment items, such as pumps, fans, and motors. Because of the complexity of the interdependencies, it was essentially too difficult to maintain this separation in the schema model. The result is that there remain a number of types in eqBase, which are related to pipes, valves and fittings and instrumentation, which should ideally be in eqPvfBase.xsd and eqInstrBase.xsd, respectively. All ongoing schema development has ensured that instrumentation and pipe, valve and fitting data are placed in the correct files.

<table>
<thead>
<tr>
<th>Facility equipment item object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq:BulkItem</td>
<td>A BulkItem is a FacilityItem that has been extended optionally to include ElectricalUtilityServices and is typically a part of either an EquipmentItem, or a FabricatedItem where multiple BulkItems have been assembled together. BulkItems are intended to be specific instances of an item, potentially with its own serial number and tag. The same object could be used as a “template” defining a standard catalog model or other item type that could be used at several locations in a facility.</td>
</tr>
<tr>
<td>eq:EquipmentItem</td>
<td>An EquipmentItem is a FacilityItem that has been extended to handle equipment items that may need EquipmentConnections, MaterialUtilityServices and ElectricalUtilityServices. EquipmentItems are intended to be specific instances of an item, with its own serial number and tag. The same object could be used as a “template” defining a standard catalog model or other item type that could be used at several locations in a facility.</td>
</tr>
<tr>
<td>eq:FabricatedItem</td>
<td>A FabricatedItem is a FacilityItem that has one or more PipingConnections. It is typically assembled from BulkItems such as a collection of PipingComponents and other BulkItems such as bolts, gaskets, etc.</td>
</tr>
</tbody>
</table>

Table 13 Description of facility equipment item objects
G. SUBJECT SCHEMAS

G.1 Subject Schema Overview

The Subject Schemas build on and extend from the Core Object Schemas in a particular subject domain.

Subject Schema namespaces include:
- eqElec: electrical equipment types
- eqHvac: heating, ventilation and air conditioning equipment types
- eqHx: heat exchanger and fired equipment types
- eqRot: rotating equipment types
- eqSld: solids handling equipment types
- eqVesl: pressure and storage vessel equipment types

These namespaces have been set up to separate various domain disciplines, which include similar types of equipment.

The concept behind the eq namespaces, which reflects schema structure, can be illustrated by considering the eqRot namespace. This namespace is for the rotating equipment domain. Rotating machinery have similar parts, for example, pumps, fans and compressors all have bearings, shafts and impellers. Rotating equipment also share basic equipment information items with totally different subject domains, such as heat exchangers. Therefore, the common elements for equipment are captured in the core ‘eq’ namespace, while the detailed subject-specific data for the rotating equipment domain are captured in the ‘eqRot’ namespace.

G.1.1 Inheritance

Inheritance is part of the fundamental structure of AEX cfiXML. The Subject Schemas extend from the core schema files.

According to XML schema and object-oriented modeling principles, any specific item in an inheritance/generalization hierarchy at a lower level in the hierarchy automatically includes all of the ‘inherited’ item contents above it. So, eqRot:CentrifugalPump objects inherit characteristics from an EquipmentItem object, which in turns inherits characteristics from FacilityItem and from there the base characteristics of obj:Object. The full hierarchy is shown as:

```
obj:BaseObject
  |_ obj:Object
    |_ eq:FacilityItem
      |_ eq:EquipmentItem
        |_ eqRot:BaseRotatingEquipmentItem
          |_ eqRot:RotatingEquipmentItem
            |_ eqRot: Pump
              |_ eqRot:CentrifugalPump
```
Each object level extends from the set of data types provided in the level above. As another example, eq:FacilityItem includes the elements associated with BaseObject and Object and ‘new’ data elements, such as weight.

G.1.2 Effects of inheritance

**Sequences**

When navigating through a tree-view of the schema, one effect of inheritance needs to be noted. As each object is extended from an object in a higher level, a sequence of data elements is added (as complexTypes are developed). Each sequence should be in alpha order (excluding namespaces and with a few rare exceptions). This means that when viewing a low level object, such as a centrifugal pump (see G.1.1), there will be a sequence of data items, in alpha order for each object, which is in turn extended into the next object with another sequence. Therefore, the centrifugal pump object will consist of 8 sequences. This may seem confusing when first viewed, but when more familiar with schema it becomes clearer which data items are associated with which object, because each object is related to a particular namespace and data group. Also, there are a significant number of existing objects which are commonly extended for defining equipment item requirements, and so these become familiar.

A key example to illustrate the potential effects of inheritance is provided by examining the valve schema files (see X.2).

**Content beyond requirements**

Inheritance allows the reuse of sets of data so that similar data are consistently represented throughout schema. This reduces potential error and increases efficiency when creating new schema. This may mean, however, that the schema includes considerably more data elements than the specific requirements for a particular work process. This is not of concern as the same schema files may be used for different work processes and contexts. The key point to note is that familiarity with the contents of commonly used data sets in the cfiXML schemas is of paramount importance.

G.2 Schema Representation of Equipment Items

The Subject Schemas are the representations of engineering equipment items and components. Examples of key equipment items which have been developed in considerable detail include centrifugal pumps and fans, shell and tube heat exchangers, air-cooled heat exchangers, centrifugal and reciprocating compressors, electric motors, control and block valves etc. Outlines of the key schema components for some of these equipment items are provided in Appendices S-Y.

The schemas detailing specific equipment items, involve modeling concepts which have been developed to closely reflect a generic, physical form of the item, whenever possible. For example, a heat exchanger unit comprises an assembly of its parts. Properties and streams associated with the conceptual, design and/or operating performance of the equipment item as a whole or of elements comprising the item are placed into the model at the appropriate level within the physical form. Elements that are used in more than
one ‘usage context’ (though have the same name) can be distinguished by using the ‘usageContext’ attribute [L.9.3]. This reduces repetition and provides a logical and clear way of developing a sophisticated and intuitive model. This is illustrated in Figure 13 which represents the kind of schema structure associated with an Air-Cooled Heat Exchanger. Examples of key components of various equipment items are provided in Appendices S-Y.

Numerous manufacturers and suppliers provide information on their web sites and it can be useful to examine real examples of the equipment item to support information provided by domain experts and data dictionaries.

Figure 13 Simplified schema modeling structure of equipment items

### G.3 Collection-container Schemas (‘Documents’) Overview

The Collection-container Schemas provide an effective way to combine core and subject-specific engineering objects in various ways to support required data transactions and usage scenarios. These can be considered as data sample exchange ‘documents’ such as data sheets, equipment lists and catalogues. They can be constructed more or less
arbitrarily as needed to suit the required purpose. All that is needed is that two or more parties agree to use the same exchange document to support a work process transaction.

Many document types can be assembled using the same base underlying data including process simulation inputs and results, equipment design program inputs and results etc.

In the context of a project, datasheets and datasheet libraries are a principal mechanism for defining the scope of a project. Such a “view” on the data can present a subset of the whole facility or site.

**Examples of these schemas are found in the schema/document directory.** Schemas representing data sheets for equipment items consist of dx:DataSheet extended to include the equipment item (e.g. eqRot:centrifugalPump), site:siteFacility and uo:materialStream. The dx:DataSheet object is an extension of obj:object, with the dx:DataSheetHeader element to allow project metadata to be recorded. The combination of all these objects provide, for example, access to commonly used data items during an engineering equipment procurement work process.

A datasheet library can be created by associating a project with a collection of DataSheetHeader/EquipmentItem combinations. The schema file eqDoc.xsd gives a basic example of such a library. A developed example is provided in eqPvfDoc.xsd, which contains a catalogue and catalogue library, as required for block valves. Note that creating a Library does not require any changes to be made to the source document. The same set of documents can be “filed” in different ways in different projects, while still referring to a single common document. The same document could also be referred to directly in the context of a specific equipment item.

Similar to dx:Datasheet, the basic capabilities are provided in dx for commercial transaction Orders containing Order Lines which may be extended to specific types of orders.

The schema also includes the special collection-container schema file ‘cfiXml.xsd’, which has been created to allow rapid schema installation and increased performance during the use of applications based on cfiXML. This file includes all the schema document files and therefore all the namespaces.
H. QUALITY DATA EXCHANGE AND DATA DICTIONARIES

H.1 Problems of defining data to be exchanged

Demonstrations and testing of the use of cfiXML for interoperability has shown that even when collaborators have chosen the schema version to use, it is important to confirm the actual data items to be exchanged. Different organizations and domains within the capital facilities industry use their own in-house data manipulation and storage systems, including data sheet type products and other software applications. When considering which data are to be exchanged, terminology and ‘local’ practices may not coincide. For example, individuals from different domains within the industry may need to agree that the data item in question is the rated volumetric flow, not the normal flow. A good solution is the creation of a data dictionary.

H.2 Data dictionaries

Example data dictionaries are provided as part of the cfiXML package [1.1.1]. These include:

- details on data items required for a particular workflow process;
- how these are represented in cfiXML.

They may also include data from real example files. This kind of dictionary allows clear communication between different participants in a data exchange work process and helps to ensure quality data exchange.

Collaborating organizations can develop their data dictionaries to whatever level of detail required, for example, specifying allowable contents for pick-lists. This provides a means of further constraining schema content for a particular work process and such a data dictionary enables application developers to build-in these constraints.

H.2.1 Developing a data dictionary

Stakeholders

It is important to use input from domain experts as well as other appropriate resources (e.g. text associated with industry recognized standards). This is because the aim is to develop the schema model to represent a generic physical form of the equipment item [G.2] and include terminology that is recognized throughout the associated engineering domain, and not just the from the perspective of one domain expert or organization.

It has also been found important that the stakeholders developing the data dictionary include persons familiar with cfiXML. Being aware of existing schema structure and engineering data already contained in cfiXML helps in the definition of new data requirements and can save considerable time.

Data item specifications

The data items included in the data dictionary examples are split into data groups, such as: general metadata; data associated with the equipment item as a whole; key equipment components; construction of these components, and fluid stream specifications. Each example of data to be exchanged is examined and broken down, if required, into the
constituent data types and clearly defined. Units of measurement are also recorded, and can be used to confirm that the property described from various example sources, is actually the same.

Example - data items in a comprehensive data dictionary are defined in terms of:
1. A data group heading and a data item name, based on standard industry data sheet headings, section headings and captions, (e.g. API 610 11th Edition);
2. Data type (e.g. String, Boolean, Pick-list, Integer, Double);
3. A definition based on industry standards and existing cfiXML content;
4. The cfiXML XPath (noting schema version detail);
5. A ‘Required’, ‘Desired’ and ‘Supplementary’ designation by domain experts of the importance of the data item being included to satisfy specified work process data exchange requirements (e.g. for BidRFQ);
6. Units of measurement.

[See H.2.2]

Sufficient detail is required in the full data item definition (including the heading context, data item name and provided definition) to enable the schema developers to clearly identify potential overlap with existing schema content by searching on key words or synonyms. With enumerations (pick-list data types), required content choices may also need checking with existing schema for equivalent items and the already provided structure and content incorporated into the data dictionary.

**Schema development and final checks**

The schema is designed so that a particular data item should only occur once in the model. The schema developer will be confirming whether ‘new’ data items, including the contents of pick-lists, do not already exist in some form in cfiXML. Final refinement of the data dictionary occurs at this stage as the schema developer works with domain experts.

**H.2.2 The AEX Project and Hydraulic Institute (HI) pump data dictionaries**

The AEX Project and Hydraulic Institute (HI) have compiled comprehensive data dictionaries to define data items which may be exchanged during centrifugal and rotary pump Bid RFQ and Bid Quote work flow processes.

The data dictionaries were produced as a major collaborative effort and the resulting standard will enable any data exchange participant to quickly and efficiently ensure quality data exchange.
I. Schema Settings

This section covers definitions for the XML Schema Namespace, targetNamespace and Form Default Qualification and the use of the schemaLocation attribute.

Throughout the cfiXML schemas the namespace prefix “xsd” is used for the W3 XSD namespace using the following definition: within the xsd:schema element:

```
xmns:xsd=http://www.w3.org/2001/XMLSchema
```

The targetNamespace is set to be the default namespace in all cfiXML schema files with the statements of the form:

```
targetNamespace="http://www.cfixml.org/mtrl" xmlns=http://www.cfixml.org/mtrl
```

In cfiXML attributeFormDefault= “unqualified” and elementFormDefault = “qualified”. The use of schemaLocation attributes is optional, however to enable the schema files to be validated easily using developers’ tools, they have been used throughout. However, no pathnames are included so that the only requirement is that all the schema files are contained in the same directory (either in the /schema or schema/document directory).

An example schema file header:
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.cfixml.org/pq"
     xmlns="http://www.cfixml.org/pq"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ext="http://www.cfixml.org/ext"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified"
     version="1.00.0000">
     <xsd:import namespace="http://www.cfixml.org/ext"
     schemaLocation="ext.xsd"/>
</xsd:schema>
```

I.1 Rationale for Schema Settings

This allows all sub-schema files to be included into a master schema file and coerce the default namespace of the sub-schema files to be that of the master file. However, “chameleon” schemas have not been used other than at this level as clearly identified separate namespaces are required as describe in the “Rational for Namespace Design” above.

The owning namespace for all elements is easily identified. Attributes are used mostly for cfiXML structure information that is required to validate relationships and provide change and comment information. Hence there is little to be gained by including the namespace qualifiers on attributes. In particular it is important not to mix FormDefault specifications as the XML would otherwise become a very confusing mixture of qualification required and not required parts.
J. XPath and XML Examples

J.1 XPaths
We recommend the following documentation for XPaths: [http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath).

Simply, an XPath describes the way through the cfiXML schema model to a specific element, and subsequently, also a means for locating a data item in an XML instance file. Each element is uniquely identified by its XPath within a particular context. Thus, XPaths for an equipment data sheet will differ from those of an equipment list since the starting document element in each case is different. However, after the document element step, the XPaths will be identical if describing the same equipment data item. Example XPaths provide a useful means to become familiar with the schema by following the path through the schema to each element addressed. Example XPaths from cfiXML have been provided below.

J.2 Example XPaths and XML
The following provides example XPaths, XML and further details required to navigate through and become familiar with the AEX cfiXML model. It is assumed that the user has a tree-view of the schemas for a clear view of the model.

The first sets of XPaths navigate to commonly used sections of schema. An outline explanation of the XPaths is provided. It is recommended that the user browses around the sequences that the XPaths navigate to, in order to note the other kinds of data provided in these particular sequences.

Later examples are provided to illustrate a number of more complex concepts used in schema as explained in the main text.

J.2.1 Document header data
1) \texttt{dx:dataSheetHeader/dx:documentID/dx:id}

Data item: Project/Job Number
All documents extended from \texttt{dx:DataSheet} include \texttt{dx:dataSheetHeader} (e.g. \texttt{eqRotDoc:CentrifugalPumpDataSheet}). The child element \texttt{dx:documentID} provides identification details on the specific XML ‘document’ (or effectively the collected data set in the XML file) through the \texttt{dx:id} content.

2) \texttt{dx:dataSheetHeader/dx:associatedDocumentID[dx:documentType="Requisition"]/dx:id}

Data item: Requisition Number
From \texttt{dx:dataSheetHeader} the child element \texttt{dx:associatedDocumentID} has a filter element of the \texttt{dx:documentType} enumeration = “Requisition” and then the \texttt{ext:String} type \texttt{dx:id}.

AEX cfiXML Schema Reference Guide 68
3) `dx:dataSheetHeader/proj:project/proj:organizationContext[ctx:organizationRole="Owner"]/ctx:organization/objb:name`

**Data item: Owner**
The `proj:project/proj:organizationContext` has a filter of `ctx:organizationRole="Owner"` set to define the subsequent organization’s name.

### J.2.2 Site data

1) `site:siteFacility/ctx:locationAndGeographicArea/ctx:location/objb:name`

**Data item: Site Location Name**
Core schema element `site:siteFacility` is commonly added to Collection-container schemas (`documents`, e.g. `eqRotDoc:CentrifugalPumpDataSheet`). The `site:siteFacility` is extended from `obj:Object` and the sequence added includes `ctx:locationAndGeographicArea`. The child element `ctx:location` is also an object, so that the `objb:name` can be used as the name of the site location object.

Note: Details on the geographic area are also available via `ctx:locationAndGeographicArea` (e.g. elevation, area geometry).

2) `site:siteFacility/site:siteSubArea/mtrl:environmentalProperty/mtrl:property[mtrl:context/etl:propertyType="Ambient"][mtrl:context/etl:propertyType="Minimum"]/mtrl:t`

**Data item: Local Ambient Temperature, Minimum**
The `siteFacility` sequence includes `site:siteSubArea`, which is used to describe specifications local to a facility area, i.e. the area for which an equipment item is required. The local ambient temperature is found via `mtrl:environmentalProperty`, which is type `mtrl:MaterialProperty`, an object used throughout schema to describe data on materials and their properties. Multiplicity is on the `mtrl:property` element and a filter of the `mtrl:context/etl:property = “Ambient”` and `mtrl:context/etl:property = “Minimum”` is defined for the `mtrl:t` element.

Note that `site:electricalAreaClassification` is available via `siteFacility/site:facilitySystem` and `site:siteSubArea`, and the former also includes data elements describing utility services which are provided.

### J.2.3 General equipment data

1) `eqRot:centrifugalPump/eq:id/eq:tag`

**Data item: Item Tag Number**
All items extended from `eq:EquipmentItem` include the `eq:id` element to describe identification information.
Note: the eq:id data set includes details on the equipment part size and type designation, model, manufacturer and/or supplier company information.

2) eq:blockValve/etl:serviceCategory/etl:serviceClass

Data item: Valve service class
All items extended from eq:EquipmentItem include the etl:serviceCategory element to specify the service for which the equipment item is required.

J.2.4 Equipment properties
1) eqRot:centrifugalPump/eqRot:pressureDifference

Data item: Pump pressure difference (or pressure drop or deltaP)
The eqRot:pressureDifference is found as a immediate child of the pump. There is also multiplicity on it. This implies that pressure difference can be specified with or without (as in this example) additional qualifiers, and in particular, this suggests the property is used in more than one usage context (see below).

2) eqRot:centrifugalPump/eqRot:rotationalSpeed[@propertyType="Rated"]

Data item: Pump rotational speed, rated
The eqRot:rotationalSpeed element is found in the 4th sequence from the bottom and is associated with the eqRot:RotatingEquipmentItem element. There is a selection of attributes available to use with the eqRot:rotationalSpeed element, such as propertyType.

In addition to equipment properties found as immediate descendants of the equipment item in question, there are those that are associated with only 1 usage context and grouped together under a suitably named parent (see C.7). And, there are many properties associated with the fluid streams passing through the equipment (See C.8)

J.2.5 Construction materials
1) mtrl:constructionMaterialProperty/mtrl:material/mtrl:materialType
Value: "Steel – Carbon"

Data item: Carbon Steel construction material
The element mtrl:constructionMaterialProperty is used throughout the schema to be able to describe construction materials of equipment items and components. In some locations in the schema, an element has been named as the "[component]MaterialProperty" and made a type mtrl:MaterialProperty, to access the same data types. The mtrl:materialType provides an enumeration list from which to select the material.

2) mtrl:constructionMaterialProperty/mtrl:material[mtrl:processingType="Galvanize"
Data item: Galvanized Steel
The sibling elements of mtrl:material include various elements which can be used to further define the construction material, such as the mtrl:processingType="Galvanized" in this example.

3) mtrl:constructionMaterialProperty/mtrl:material/mtrl:externalName/mtrl:astmName
   Value: "A105"

Data item: ASTM code
As commonly required for engineering equipment, the mtrl:material/mtrl:externalName sequence allows the user to add external organization material codes or identification details, as required.

4) mtrl:constructionMaterialProperty/
   mtrl:material[mtrl:auxiliaryMaterial[mtrl:materialApplication="Lining"]/]mtrl:materialType="Polymer - Perfluoro-Alkoxyalkane (PFA)"]/mtrl:materialType
   Value: "Steel – Carbon"

Data item: PFA-lined Carbon Steel
More complex construction materials can be described by using the mtrl:auxiliaryMaterial element, and child elements of mtrl:materialApplication and mtrl:materialType.

J.2.6 Substitution group
Example - end connections
Example XPath:
   eqRot:centrifugalPump/eq:equipmentConnection/eq:endDescription[eq:endConnectionType="Flange"]/eq:flange

All items extended from eq:EquipmentItem include the eq:equipmentConnection. The XPath shows that detailed information about a flange end connection is required and so the endConnectionType “Flange” is selected and as eq:flange is part of the substitutionGroup “endConnectionItem” all flange data elements are available.

Example XML:

```xml
<eq:equipmentConnection
   objectID="eq.EquipmentConnection.EquipConnDownstream">
   <eq:endDescription>
     <eq:flange objectID="eq.Flange.PipingOutlet">
```
J.2.7 Usage context

The considerations for determining which ext:EUsageContext enumeration is appropriate are summarized as:

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual specification</td>
<td>If the specification is a process specification of what you want the plant to do without consideration of the mechanical dimensions, models, availability etc. of the equipment, contains heat and mass balance data, &quot;unit operation&quot; data, streams with no pressure drop connected to ports, then you have a conceptual specification.</td>
</tr>
<tr>
<td>Construction specification</td>
<td>If the item is in existence - then the dimensions and measurable mechanical details of the item are the construction specification, e.g. As built</td>
</tr>
<tr>
<td>Design specification</td>
<td>If the item is still on paper and no individual equipment item exists yet then the dimensions and other requirements for the item are the design specification.</td>
</tr>
<tr>
<td>Operating performance</td>
<td>If the information covers the operating conditions and performance, either from manufacturer performance curves, rating program output or measured conditions in a plant then you have operating performance.</td>
</tr>
<tr>
<td>Unspecified</td>
<td></td>
</tr>
<tr>
<td>custom</td>
<td></td>
</tr>
</tbody>
</table>

Example XML

```xml
<eq:controlValveTrim>
  <eq:controlValveSeat>
    <eq:valveLeakageClass usageContext="Design specification" propertyType="Minimum" >IV</eq:valveLeakageClass>
    <eq:valveLeakageClass usageContext="Operating performance" propertyType="Rated" >V</eq:valveLeakageClass>
  </eq:controlValveSeat>
</eq:controlValveTrim>
```
The above xml example shows data for the element valveLeakageClass for either a
design or an operating performance specification, as well as an additional attribute of
propertyType (see also 2a & b XPaths below).

Example XPaths:
1) /eqPvfDoc:controlValveDataSheet/eq:controlValve/eq:designSpecification/
   eq:flowCharacteristicType

   The data element eq:flowCharacteristicType provides a list of enumerations, and as it is
   only a design specification, occurs as a child of a clearly named element,
   ‘designSpecification’.

2) Two XPaths illustrating how the same element may have a different usage context and
in this case, a different conditionType too.

   a) /eqPvfDoc:controlValveDataSheet/eq:controlValve/eq:controlValveTrim/
      eq:controlValveSeat/ eq:valveLeakageClass [@usageContext="Design specification"]
      [@propertyType="Minimum"]

   b) /eqPvfDoc:controlValveDataSheet/eq:controlValve/eq:controlValveTrim/
      eq:controlValveSeat/ eq:valveLeakageClass[@usageContext="Operating performance"]
      [@propertyType="Rated"]

   The element describing the leakage class of a valve seat is required in at least 2 known
usage contexts, i.e. within a design specification and within an operating performance
specification. This element is therefore a child of the control valve seat itself and as
eq:valveLeakageClass is an enumeration, the attribute usageContext is automatically
available within cfiXML. The element in the schema must have multiplicity on it to
allow for more than one occurrence of it as the attributes are used.

J.2.8 Equipment stream connections – substitutionGroup, usageContext and objectState

The explanation of material streams is provided [N.6]. The following XPath and XML
illustrate the concepts of a locally referenced contained materialStream object, the use of
the usageContext attribute [L.9.3] and the materialStream substitutionGroup head
element [N.1].

Example XPath:
1) eqRot:centrifugalPump/eqRot:streamConnection[@usageContext="Operating
   performance"][etl:normalFlowDirection="Inlet"]/uo:stream[uo:streamType="MaterialStr
   eam"]/uo:materialStream/uo:materialFlow[etl:flowPropertyType="Rated"]/uo:actualVolumet
   ricFlow

Example XML:
<uo:materialStream objectID="uo.MaterialStream.MaterialStreamInOpPerf" usageContext="Operating performance" contenttype="text/xml" unitSet="SI" xmlContent="cfiXML" objectState="fullXML" language="en" currencyCode="USD">
  <obj:name>Demineralized Water</obj:name>
  <uo:materialFlow>
    <etl:flowPropertyType>Minimum</etl:flowPropertyType>
  </uo:materialFlow>
</uo:materialStream>

<eqRot:centrifugalPump>
  <eqRot:streamConnection usageContext="Operating performance">
    <etl:normalFlowDirection>Inlet</etl:normalFlowDirection>
    <uo:stream>
      <uo:materialStream objectID="uo.MaterialStream.MaterialStreamInOpPerf" objectState="refLocal"></uo:materialStream>
    </uo:stream>
  </eqRot:streamConnection>
</eqRot:centrifugalPump>

J.2.9 Equipment inspection tests – extensions and enumeration union

Example XPath:

```
eqRot:centrifugalPump/eq:inspection/eqRot:pumpTest [eqRot:pumpInspectionTestType="Hydrostatic"]/eq:verificationType
```

Data item: Centrifugal pump hydrostatic inspection test verification type

All eq:EquipmentItem objects have eq:requirement. Child elements include items such as analyses, documents, inspection tests, shipping and storage, paint, welding etc. requirements.

There are inspection test data elements common to equipment items and pumps. The pumpInspection is an extension of Inspection (in eqBase.xsd). A similar structure is used for fan tests, impellers etc.

In addition to the inspection test sets being built up of common data types, the element eqRot:pumpInspectionTestType is the simpleType enumeration list which uses xsd:union. This is not commonly used in cfiXML as some schema editors do not allow the enumerations to be easily viewed. However, as the same inspection test types are used by different equipment items, and then in addition to these, others are used for rotating equipment items, and still more specific ones for pumps, the union system has been used. So, eqRot:pumpInspectionTestType includes pump-specific tests and enumerations associated with eqRot: rotatingEquipmentInspectionTestType. The latter
consists of a union of eq:EInspectionTestType and test types specific to rotating equipment items. Some schema developer’s applications may not be able to view the full union of the enumerations and so a text view of each subset may be required to see all test types available.

J.2.10 Use of alternativeOption

Example XML:

1) In this case the alternativeOption attribute is used to allow more than one enumeration to be selected.

```xml
<eq:applicableStandard>
  <eq:valveDesignStandardType alternativeOption="true">ASME/ANSI B16.34</eq:valveDesignStandardType>
  <eq:valveDesignStandardType alternativeOption="true">API 608</eq:valveDesignStandardType>
</eq:applicableStandard>
```

2) In this example the alternativeOption attribute is added to the material object to allow different child elements to be considered as alternative options.

```xml
<mtrl:constructionMaterialProperty objectID="mtrl.MaterialProperty.1">
  <mtrl:material objectID="mtrl.Material.870" alternativeOption="true">
    <mtrl:auxiliaryMaterial>
      <mtrl:materialApplication>Lining</mtrl:materialApplication>
      <mtrl:materialType>Polymer - Perfluoro-Alkoxyalkane (PFA)</mtrl:materialType>
    </mtrl:auxiliaryMaterial>
    <mtrl:materialType>Steel - Carbon</mtrl:materialType>
  </mtrl:material>
  <mtrl:material objectID="mtrl.Material.871" alternativeOption="true">
    <mtrl:auxiliaryMaterial>
      <mtrl:materialApplication>Lining</mtrl:materialApplication>
      <mtrl:materialType>Polymer - Perfluoro-Alkoxyalkane (PFA)</mtrl:materialType>
    </mtrl:auxiliaryMaterial>
    <mtrl:materialType>Iron - Ductile</mtrl:materialType>
  </mtrl:material>
</mtrl:constructionMaterialProperty>
```

J.2.11 Use of order attribute for array lists

Example XML:

In this example for eqRot:centrifugalPump/eqRot:operatingPerformance/eqRot:performanceCurve array lists, the order links each flow volume with the associated head.

```xml
<eqRot:performanceCurve>
  <eqRot:curveNumber>headCurveRated</eqRot:curveNumber>
```
<eqRot:orderedPointData>
  <eqRot:flowVolume symbol="m3/h" order="1">0.0</eqRot:flowVolume>
  <eqRot:flowVolume symbol="m3/h" order="2">5.74</eqRot:flowVolume>
  <eqRot:flowVolume symbol="m3/h" order="3">11.4</eqRot:flowVolume>
  <eqRot:flowVolume symbol="m3/h" order="4">17.2</eqRot:flowVolume>
  <eqRot:head order="1">124.99956425680377</eqRot:head>
  <eqRot:head order="2">124.75718642384413</eqRot:head>
  <eqRot:head order="3">124.45144127962853</eqRot:head>
  <eqRot:head order="4">124.08603282307767</eqRot:head>
</eqRot:orderedPointData>
<etl:propertyType>Rated</etl:propertyType>
</eqRot:performanceCurve>

J.2.12 Use of fixedProperty

In this example the fixedProperty element provides the temperature at which other properties are determined.

Example XML:

```xml
<!--Material stream -->
<uo:materialStream>
  <mtrl:materialProperty>
    <!—Normal Temperature property -->
    <mtrl:property>
      <mtrl:t symbol="degF">120</mtrl:t>
      <mtrl:context>
        <etl:propertyType>Normal</etl:propertyType>
      </mtrl:context>
    </mtrl:property>

    <!—Properties at Normal temperature -->
    <mtrl:property>
      <mtrl:cpMass symbol="BTU/lb.degF">1.8</mtrl:cpMass>
      <mtrl:specificGravity>0.888</mtrl:specificGravity>
      <mtrl:vaporP basis="gauge" symbol="psi">1.8</mtrl:vaporP>
      <mtrl:viscosity symbol="cP">0.87</mtrl:viscosity>
      <mtrl:context>
        <mtrl:fixedProperty>
          <mtrl:t symbol="degF">120</mtrl:t>
        </mtrl:fixedProperty>
      </mtrl:context>
    </mtrl:property>
  </mtrl:materialProperty>
</uo:materialStream>
```
K. PRINCIPLES OF SCHEMA MODELING AND DEVELOPMENT, AND ENGINEERING DATA

K.1 XML Schema Development Process Overview

The recommended process for the development of XML schemas is to start with a detailed definition of business-driven requirements for interoperability. This would include defining a usage scenario that involves two or more software systems and a detailed set of data requirements for an exchange or sharing transaction that supports a common industry work process. A full description of a methodology for identifying, prioritizing and fully describing business-driven industry-deployed interoperability projects is described in the Fiatech Interoperability Plan. There are several criteria that help to ensure success in the schema development work process.

- Collaborative active participation by all stakeholders, including owners, designers and equipment purchasers, equipment suppliers and software suppliers, is essential to project success.
- A common understanding of the commercially significant work processes and usage scenarios are included because it is essential to align project scope with anticipated business benefits.
- A demonstration of commercial software implementations is included as part of the development process to validate the XML schema and to validate business benefits.

Engineering XML schema development projects need to be approached as a collaboration work activity between domain experts (users and suppliers of materials or equipment), data-modeling/XML schema experts, and software suppliers.

When approaching an XML schema development and deployment project, the following steps are recommended.

K.1.1 Business-driven Data Requirements Definition (domain users)

1. Domain experts identify documents that are heavily reused in the work process, e.g., equipment datasheets or catalogues. Identify, review and use relevant existing industry engineering and data standards that may be available.

2. Domain experts identify the key usage scenarios and data exchange transactions that clearly show the anticipated business value from electronic data exchange. Describing these usage scenarios and data transactions should include designating what people, roles and software are typically used to produce or use the document and the data items found on these documents. Capture the result of this in information flow diagram that shows the key software and key documents produced and used by the software.

3. Domain experts identify the specific data items, in detail, in the exchange document that supports the key usage scenarios (may be all items or a subset). At the same time, define the minimum required data to support each key usage scenario and data transaction so that business value is achieved.
4. Domain experts develop a detailed terminology dictionary for the data items in these documents that support the key usage scenarios. The data dictionary should include the engineering terminology itself, well-written definitions for the engineering terminology, designation of data type, e.g., numeric value with units of measurement, yes/no, enumeration selection, date, time, any role associated with the data item (e.g., provided by buyer or supplier), and so forth. If units of measurement are needed to support the value, specify the customary, metric and SI units that are typically used. Capture this data glossary in a spreadsheet.

5. Domain experts arrange the glossary data items into logical data groups by identifying the object to which it belongs, e.g., “impeller diameter” is a part of the definition for “impeller.”

K.1.2 Schema Development (data modelers working with domain experts)

6. Data model/XML schema experts, in collaboration with software suppliers, develop an object model and XML schema definition from these domain terminology spreadsheets. UML models may be produced if needed in this process to aid understanding. The result of this will be an object-oriented XML Schema that should be readily implemented in software data mappings. This result will be reviewed with domain experts and software suppliers to ensure that key usage scenarios are supported in the XML schema and that the schema can be supported by software implementations.

K.1.3 Schema Deployment (domain application developers working with domain expert users)

7. Software suppliers validate the proposed XML schema by building software implementations to support the electronic data exchange using the XML schema. Adjust the schema as needed to address any software implementation issues that arise.

8. Publish the XML schema and associated documentation, such as the minimum data requirements for a transaction set that can be validated and used in commercial software implementations.

9. Incorporate XML schema definitions, associated minimum data exchange requirements and software implementation guidance into existing industry standards efforts.

10. Extend and maintain the XML schemas over time as more and more software implementations are built to support various commercial usage scenarios over time.

K.2 Schema Information Modeling Principles

There are several dimensions to consider when approaching the development of an XML Schema. We have organized the basic principles into several categories related to the following dimensions that range from very broad to very specific as they progress:

- Modeling Principles
• XML Schema Principles
• Engineering Data Principles

The modeling principles are proposed to serve as useful practices to generate useful, sufficiently rigorous work process descriptions and rigorous, yet flexible, data models that support the business process through software implementations.

XML schema principles are proposed as a means to represent the data models that are developed through the previous set of principles.

Engineering data principles are proposed that place requirements on the data models and XML schemas over and above the use of the May 2001 XML Schema Recommendation.

K.3 Modeling principles
The modeling principles are proposed to serve as useful practices to generate useful, sufficiently rigorous work process descriptions and rigorous, yet flexible, data models that support the business process through software implementations.

K.3.1 Develop and deliver business process models.
Projects should deliver business process models to systematically document work process information flows, the common transactions and the message sequences across software systems and company boundaries. Understanding these work process models are essential context for developing business-focused data models.

K.3.2 Develop data models using common engineering terminology.
Define and organize data item names using common engineering terminology and engineering concepts that are readily understandable to users and to all types of software owners, e.g., “Inlet Temperature” would be an explicit element defined in the schema. Using this approach should simplify the effort to build software mapping interfaces.

K.3.3 Define data models using a simple, explicit and contained style.
The ease with which software mappings can be built is a key success factor for widespread adoption and use. There should be only one way to do a mapping and that way should be simple. Extensive use of “overloading” techniques to enable a single piece of schema, e.g., an “any” element, to be used with multiple meanings should be avoided. Similarly, avoid overusing referencing mechanisms to minimize the complexity of software mappings.

K.3.4 Use standard modeling notations where possible.
To the extent they are developed, data and information flow modeling documentation should employ standard information analysis techniques and methodologies to facilitate cross-industry communication. One examples of such an approach is the UMM (Unified Modeling Methodology), which includes UML (Unified Modeling Language).
K.3.5  **Use modular data architecture.**
Projects should use a modular data architecture that allows the using companies to deploy well-defined subsets of scope. These scope subsets would typically relate to the support of specific business and work processes or subject matter domains. Modular data architectures should also provide for defining and developing reusable core component libraries and/or adopting and using core component libraries developed by others as appropriate. These core component libraries should be stored in repositories for the use of various user communities. The core component repository should be indexed in a registry.

K.3.6  **Define / develop reusable concepts underlying large amounts of data.**
Related to the previous item, projects should define and develop reusable concepts and ‘metadata’ (data that describes other data) that underlie large amounts of engineering data, for example, define a common strategy for handling units of measurement, data access security, object naming, object referencing, etc. as described under XML Schema Principles related to engineering data requirements.

K.3.7  **Adopt a set of naming rules and conventions.**
For example, consider how to name “nominal impeller diameter” for a pump. Should it be nominal_impeller_diameter, NOMINAL_IMPELLER_DIAMETER, NominalImpellerDiameter, nominalImpellerDiameter, nomImplrDiam, vNomImplrDiam, or vDiamImplrNom? What strategy should be used for separating pieces of compound names? Should prefixes be used? Should abbreviations be allowed? These are issues that should be debated and settled early on, and then applied with a high degree of consistency across all areas of scope. Adopting a set of naming rules and conventions will enhance consistency, usability, and readability of large complex schemas.

K.3.8  **Define needed alternative names in application or XSLT mappings.**
If alternative names are needed for data model elements, then these alternative names should be contained in application-specific mappings, or in XSLT mappings. A key principle for reducing complexity of software mapping to a standard vocabulary is to have only one name for a given data item. In some sense, it doesn’t matter what we call it, as long as we know what it means and can easily build explicit software mappings to it.

K.3.9  **Group related data items together into reusable logical groups.**
Some suggested approaches are to define a reusable data group that:
(a) supports 2 or more work activities
(b) appears on more than one document or equipment type, e.g., more than one type of equipment datasheet.
(c) is commonly referred to by a common unique label, ID, or tag, where the unique identifying label enables you to refer to a whole collection of data by its unique label.
(d) have common type, access and persistence. Some information is very dynamic, while other types of information may never change. By grouping data objects with common persistence together, simplified access of the common information is facilitated.

K.3.10 Minimize complexity and depth of nesting.
The proper level of nesting is largely a judgment call and depends on modeling requirements. Some level of nesting is likely to be required, but excessive levels of nesting, if not needed, can lead to excessive complexity. Mapping data to transports over low bandwidth communications links is of common and major concern. Additionally, user interfaces have practical presentation limits of nesting presentation. Therefore, mapping strategy should take advantage of techniques that minimize complexity of resulting models mapped to specific protocols to shorten resulting messages.

K.3.11 Define appropriate relationships between data groups.
In developing data models, it will be necessary to have relationships between data groups. Some groups will contain other groups. Some groups will use other groups. Some groups will extend other groups so that one group is essentially a specialization extension of a common reusable group. Making the distinction in a group-to-group relationship between “contains,” “uses” and “is a specialization extension of” is important to model correctly, considering the context of the multiple various work activities that use the data grouping. These three relationships between groups are perhaps the most common, but there may be others as well. One thing to note is that these relationships may extend to schema elements in other schemas, allowing references to made to objects described by other XML schemas contained in other namespaces (see principle 4.6.9 about using a standard approach to defining and referencing objects).

K.3.12 Treat user presentation issues separately from data modeling issues.
Data modeling is effective for defining unambiguous exchange of information. User presentation of data includes the use of alternative languages, or units of measure within a user interface. User presentation is separate from data modeling representations. Avoid use of alternate representations of the same thing as in temperature, degrees C or F. Tailor the local use and presentation of data to the local preference in an application user interface, but such tailoring is not required for electronic exchange between software systems. The burden of translating the data into an effective user presentation should be in the application, and application translation mapping, not in the data model representation.

K.4 XML Schema Development Principles
XML schema principles are proposed as a means to represent the data models that are developed through the previous set of principles.

K.4.1 Use XML Schema to define information models
Because the W3C XML Schema recommendation currently enjoys the strongest industry support among non-DTD XML schema languages, XML vocabularies should be published in the form of W3C XML schemas.
Consider using RELAX NG because it is both easier to read and is more expressive than the W3C XML schema language. Software tools exist for converting RELAX NG schemas to W3C XML schemas. So using this approach can be accomplished with minimal effort by using RELAX NG for schema development and auto-generating the W3C XML schemas to include in published specifications.

K.4.2 Use object-oriented data models as the basis for XML Schema. XML schemas should be based on object-oriented data models developed using the data modeling principles of the previous section. It may be helpful for projects to use UML class diagrams to help understand how an XML Schema representation should be structured.

K.4.3 All elements in an XML schema should be optional by default. By default, all elements in an XML schema should be optional, unless there is a good reason to make an entry required, e.g., a unique object identifier. The rationale for this is that there may be multiple business transactions that reuse a common piece of schema. In one transaction an element be required, where in another it may not be. We recommend that XML schema required elements not be used as the enforcement mechanism to ensure a valid data transaction. This strategy avoids burdening applications with populating required fields with null values in order to have a valid XML instance file.

K.4.4 Define/document minimum data sets for transaction validation. For the identified work processes and data transactions, it is important to define and document the minimum data to accomplish a data exchange transaction to adequately support the work process in a minimum way. In conjunction with the previous principle, however, the enforcement of the validity of this minimum transaction set should be external to the XML schema definition and not included as required elements in an XML schema.

K.4.5 Define common user extension conventions. Define common usage conventions for handling user extensions to the standard XML schemas that are developed. These conventions should allow seamless user customization of models without affecting the interoperability of standardized features, i.e., extensions to a schema should not require that the standard schema be edited.

K.4.6 Support schema versions. It is essential that schema versions be supported to enable the XML schema definitions to evolve over time.

K.5 Engineering Data Principles Engineering data requirements on the data models and XML schemas over and above the use of the May 2001 XML Schema Recommendation are also included.
K.5.1  Develop reusable schemas that underlie various engineering schemas.
Engineering data share a number of common characteristics that should be handled using broad reusability mechanisms, e.g., extended base data types that support units of measurement in a consistent way. These underlying schema components should be developed in a reusable way, to minimize effort in developing a consistent engineering schema.

K.5.2  Define a systematic approach for units of measurement.
Develop a systematic approach for associating units of measurement with any element in an XML schema. This approach should provide for use of SI units by default, but should also optionally allow other units of measure to be used.

K.5.3  For simplicity, consider adopting SI units only.
To simplify building application maps, consider adopting SI units only so that applications will not be required to supply any units conversion beyond that required for conversion to and from standard SI units.

K.5.4  Define a systematic approach for handling valid data ranges.
Define a systematic approach for handling valid data ranges associated with any element in a schema.

K.5.5  Support vectors, tables and multi-dimensional arrays.
Define extended base data types that allow the use of vectors, tables and multi-dimensional complex arrays.

K.5.6  Optionally support stored information about user access control.
Provide a systematic approach to optionally support user access control to any data element in the set of engineering schemas through inherited optional descriptive attributes.

K.5.7  Optionally support encrypted binary data for secured data access.
Related to the previous requirement, optionally provide encrypted data and encrypted binary data for secure data access mechanism in a single file through inherited optional descriptive attributes.

K.5.8  Provide standard mechanisms for extending XML schema.
Provide a systematic approach for easily customizing and extending the engineering schemas as needed for non-standard data transactions among consenting parties. One mechanism to accomplish this may be to include references to and use of external catalogues.
K.5.9 **Provide standard approaches to object identification and references.**
Provide a systematic approach for uniquely identifying objects and a mechanism to reference these objects either within the same file, or across multiple files, databases, document management systems, and the web, located on internal or external servers, etc.

K.5.10 **Support embedded binary data.**
Provide a mechanism for embedding binary data, e.g., to contain software-specific data to support iterative round-trip engineering work processes.

K.5.11 **Optionally support storage of associated revision data.**
Optionally provide a systematic mechanism for tracking changes to stored data since previous revision(s) and other data transactions through inherited optional descriptive attributes that describe a data version. (Note this pertains to data versions contained in archived XML files, not revisions to schema versions covered by principle 4.5.6.)

K.5.12 **Provide descriptive data about roles or use of a data element.**
Provide a mechanism to support the data that describes the role or use of the data attribute, e.g., this group of data is typically supplied by the purchaser or by the vendor. On some current equipment datasheets, for example, circles and squares are used to denote these roles.

K.5.13 **Provide a mechanism to determine data type of “any” elements.**
If an “any” data type is used in a user extension to schema, provide a mechanism to enable software applications to know what data type is included.

K.6 **Deployment Requirements and Constraints**

K.6.1 **Complete Documents**
It must always be possible to send a single file or document of XML that is complete and contains no references to other XML. The XSD must support this fully and not require a 'get-round' to achieve it.

K.6.2 **Duplicate XML**
A cfiXML document should never contain more than one copy of the detailed XML data of an object. Even if more than one reference to the same object exists within a document, the XML data of that object must only appear at most once within the document. For example, if the same stream object is connected to one side of a heat exchanger as well as appearing in a table of streams within a data sheet document, then one of the streams objects in the data sheet will refer to the other. This target, referred to, stream object will contain all the detailed information on the stream.

K.6.3 **XML Validation and Test Files**
As mentioned above, schema should be validated often during development and using more than one tool. In addition, part of the aim of the cfiXML schema model is to ensure that resulting xml is succinct and clear. Therefore, it is strongly recommended that
schema developers create a test xml file, based on a real-life usage scenario as they create new schema. As the test xml is written and validated, any errors or additional requirements may be highlighted and the schema can be adapted to produce tidy xml. Example files also help developers conform to element naming best practices, where redundant usage of a label in an element containment hierarchy can be easily seen and reduced or eliminated. Ideally, the schema should be tested in a real application.

K.6.4 XSLT and Standard Parser Processing

It must always be possible to process any cfiXML with XSLT and standard parsers without the need to write any custom software to reformat the information, such as for display on a Web site. As an extension, references to objects must contain sufficient information to be meaningfully reported, e.g., a reference to a company should at least include the company’s name.
L. TYPES AND ELEMENTS

Type definitions are used throughout the Schema and then instantiated as elements. This is known as the Venetian Blind Design Style. When a global type is created, the definition file in which it is placed determines the level of reusability of the type through the schema model.

L.1 Rationale for Types

The Venetian Blind Design maximizes the ability to reuse schema including derivation of more complex types. This is ideal for modeling engineering data, which contains large numbers of items that are based on similar foundation items or which contain much common data. Also, by making types that are not relevant outside of their context, local within a containing type, they are hidden and not available anywhere else.

L.2 Elements

Elements are used for containing virtually all the engineering data, mostly based on complexTypes. The base simpleType “atomic” elements are created as ‘ext’ data types or ‘pq’ data types. SimpleType enumeration lists are also common components of the schema instantiated as elements and/or a limited number of attribute groups.

L.2.1 Rationale for Use of Elements versus Attributes

Original design discussions concluded that elements would be used as the primary schema component for containing virtually all the engineering data.

The rationale for this decision includes:

1. The line between what is data and what is meta-data is blurred in many cases.

2. Meta-data vs. data classification - it may well be difficult to decide which data should be considered the main value and which the meta-data with the possibility that more than one ‘main’ data value may exist.

3. Extension of attributes is difficult but for elements is easy.

(Full original rationale available in ‘The Schema Design Principles.doc’ (available via WinCVS), section 7.2 Element vs. Attribute)

Other disadvantages of using attributes include:

1. Attributes have to be added to each element that may require the 'qualification' as opposed to providing an element within a sequence and multiplicity on the parent.

2. An attribute can only be used once on the same element.

Overall, the use of elements supports clearly defined schema development, though there is the noted added complexity for implementations.
L.2.2 Creating Base ‘Leaf’ Elements – use of ext and pq namespaces

A simpleType element should be used when the data item is “atomic,” e.g. is a single value or string etc. The element data type should be an “ext” base data type (e.g. use ext:String and not xsd:String). This is to include the additional items provided in the ext namespace to support functionality through the use of attributes, such as change flags and comment. These are required in all engineering data.

Similarly, for physical properties, a base “leaf” element should be a “pq” data type, in order to automatically include the valid physical quantity units of measure symbols in addition to the “ext” base attributes.

L.2.3 Optional by default

All elements in the schema should be optional by default. A few exceptions are allowed for specific purposes (e.g. transactionType to support object transactions).

**Rationale for Elements Being Optional By Default**

1. The same schema is being used for multiple usage scenarios. Using a required element to enforcing compliance to one usage scenario may result in the same element to be required to have a null value under a different usage scenario.

2. The cfiXML schema supports the use of “delta” data transactions, i.e., transmitting only changed data in the context of making revisions to documents. Having required elements would force files to contain large numbers of required element tags with null values, thereby significantly increasing file size for no added value.

L.3 Creating Complex types

If when developing content models and extensions, a reusable group of related data items is recognized, then a complexType should be created to contain the sequence of the associated elements. This is to ensure that sets of elements that are common to more than one section of schema are available to be reused where required. Thus, the same elements always have the same name, annotation and order and the chance for error is reduced. The complex types may be extended from objects, include objects in a sequence, or be a sequence of non-objects.

A global element is then created with its type declared as the complexType, so as to be able to reference this element as required [L.7].

The decision whether to create a complexType should involve trying to minimize the number of levels in a sequence hierarchy and thus reduce the complexity of the schema.

**Example 1** – A simple reusable sequence of elements is shown in Figure 14. The complex type etl:Coordinate, is taken from the etl namespace.
Example 2– When creating schema for new facility items, most will be based on equipmentItem, bulkItem or fabricatedItem elements.

```
obj:BaseObject
  ⇓ obj:Object
  ⇓ eq:FacilityItem
  ⇓ eq:EquipmentItem (or eq:BulkItem)
  ⇓ eq:MyNewFacilityItem
```

L.4 Creating SimpleType Enumeration Lists

When a pick-list of data is required, a simpleType enumeration list is created. A global element is then created by extending the simpleType to include the attribute group reference of ‘ext:BaseEnumerationAttributeGroup’. This element is therefore available to be referenced in schema. [L.7]

L.4.1 Enumeration list order and format

The main pick-list items should be in alpha order, with the first letter in upper case and in a sentence type format, as appropriate (e.g. ‘Automatic’; ‘High temperature’; ‘Care, install and spares’). Abbreviations should not be used without including equivalent text in full. However, standard codes are valid. Consideration should also be given to whether the list should be ‘grouped’, to provide an easier way to find related data items (e.g. ‘Load – full’, ‘Load – partial’, Load – rated’).

There are also the following exceptions:

1) **The pick-list item is a known acronym**, which should be spelled out and provided in the following type of format:
   ‘Process Flow Diagram (PFD)’
   ‘Request For Quotation (RFQ)’
2) The simpleType forms part of what was originally a derivable pair of elements and now **lists elements which may form part of a substitution group** [N.1.4]. In this case, the contents which can be substituted are in CamelCase, and those which are not available as substitution group members are in lower case, normal text:

- ‘ElectricMotor’
- ‘Engine’
- ‘GasEngine’
- ‘gas expander’
- ‘other’
- ‘unspecified’
- ‘Nil’
- ‘custom’

The final items in all enumeration lists should be:

- ‘Other’
- ‘Unspecified’
- ‘Nil’
- ‘custom’

*(Note lower case for non-substitute group members, as above)*

The ‘Other’ and ‘custom’ items support schema flexibility and extensibility requirements. Only if there is some very specific reason (e.g. there can be no other possible contents), should the ‘Other’ and/or ‘Unspecified’ items not be included.

The ‘Nil’ enumeration is required to allow the recording of a change of a data value to no value and also potentially for use with referenced properties [L.9.2].

Further information on naming conventions is provided [M].

**L.5  Format of complexType and simpleType Enumeration Names and Associated Elements**

By convention, complex and simple types are capitalized and the associated element is given a lower case first letter. For example, type ctx:Organization has a corresponding global element of ctx:organization. SimpleType enumeration lists always have an additional prefix of ‘E’ to indicate enumeration (e.g. ext:EUsageContext).

**L.6  Location of Types and Elements in Schema**

An element created with its type declared as the complexType (and this includes those associated with the enumeration lists) is a global type. However, due to the inheritance structure of cfiXML, the location of an element determines to what extent it becomes available throughout the schema model. If the data are only applicable to the particular equipment item, then the element should be located in the subject schema file. If the data could be used by different equipment items in other namespaces, then the element should be placed accordingly. For example, if the item is only associated with centrifugal
pumps, it should be located in eqRotCpmp.xsd, or, if the item is associated with rotating equipment in general, it should be placed in eqRotBase.xsd. If associated with non-specific equipment items, it should be in eqBase.xsd.

Note that a complexType, followed by the associated element, should be placed in the upper part of an xsd file, in alpha order. The simpleTypes (and elements and associated attributes) are placed under the complexTypes, also in alpha order. Exceptions are when additional elements based on an existing type are added to provide, for example, a synonym to suit a specific engineering domain. These are kept with the associated simpleType.

L.7 Use of Global Element References v Global Types

In defining content models, it was the recommendation of the NIST AEX test bed reviewers to use references to global elements in preference to global types and this is the predominating practice in the cfiXML schemas. However, occasional problems in software implementation (parser problems) have occurred. Also, in certain circumstances for schema readability reasons, applying the more commonly used object-oriented modeling principle of using the global type can be preferable, especially if the number of global element “synonyms” for the complex type becomes large.

L.8 Content Models

L.8.1 No use of ‘All’

Unfortunately, the use of the “all” content model, which has the desirable property of not enforcing an ordered sequence of elements in the instance file, is generally not considered practical for engineering information models. This is because the elements must all be of simple type, and most elements in cfiXML are of complex types. So the “all” content model is not used in cfiXML.

L.8.2 No use of ‘Choice’

The use of the “choice” model is discouraged. Software implementations of this content model are significantly more difficult than alternative approaches in that developers must define each choice fully in code. This may not be an issue for XML serialization tool users with cfiXML LT, but more sophisticated applications can be better supported by other schema structures, such as substitution groups. Until the development of cfiXML LT, the alternate ‘choice’ requirement was efficiently met by specifying the ‘xsi:type’ attribute through the use of derivable type polymorphism. As of cfiXML Public Release v4.0, substitution groups have replaced derivable types [N.1].

L.8.3 Use of ‘Sequence’ and Ordering Conventions

The cfiXML schemas make extensive use of the ‘sequence’ model for logically related data groups. Some of these lists can become quite lengthy. The element/group order is important to consider from a usability and maintenance point of view. For long lists of items, alphabetical order is always the most usable order for locating terms. From the schema maintenance viewpoint, alphabetical order can be maintained from one schema version to the next, because virtually all elements are optional and so additional elements
can be inserted at the appropriate location without invalidating existing XML files. For
the sake of usability there is some valid argument about using ‘natural or logical order’
versus alphabetical order in some cases (e.g., the order of elements in a mailing address).
Accordingly, the following rules have been applied to the sequences and enumeration
lists in the cfiXML schemas.

1) Use alphabetical order for elements in a sequence list, ignoring any namespace
prefixes that occur in elements.

2) If there is a defensible, logical natural order to an element group, such as the
elements that comprise a mailing address, then use the logical natural order in
preference to alphabetical order.

L.9 Attributes and Qualifiers
Generally, attributes are used to provide additional supplementary information about the
data contained in the element, e.g., for describing metadata as types of qualifiers, setting
usage context, an indication that the data has changed, instructions for recipients, and
comments or commentReferences.

In addition to the attributes provided for all base elements and simple types in cfiXML
[D.1.2], there are various additional attributes which have been created to simplify
schema and improve functionality as a “recommended practice” Examples of these are
discussed below.

L.9.1 Using Conditions or Qualifiers
The use of attributes has been found to be effective for particular metadata. Key sets of
attributes are located in etl.xsd and provide a means of reducing the repetition of similar
elements, thus reducing the size and complexity of the resultant schema, and the length of
element names. As an example, the attribute propertyType includes enumerations such
as “Average”, “Maximum”, “Rated”, “Normal” etc.

Most enumerations in etl are available as either elements or attributes. This has been
found to be particularly important for etl:propertyType and etl:conditionType. These are
commonly used to qualify data property values in cfiXML. The principle by which they
are used as either an attribute or an element depends on the number of elements within
the associated sequence to which they apply. If only a few elements of a sequence
require a conditionType and/or propertyType qualifier, then these elements have the
associated attribute groups added to them and are made unbounded (e.g. ‘ElementB’ in
Figure 13). This multiplicity allows for the multiple use of the same element, but with
different attributes. If numerous elements within the same sequence require the use of a
conditionType and/or propertyType qualifier, then these are added as elements to the
sequence and the parent element to the sequence must have multiplicity on it.

L.9.2 ReferenceProperty and ReferencePropertyID
This referencing mechanism works by allowing a property within a single document to be
referenced to another property, on which the primary property value is dependent.
For example, consider the flow volume of a pump which is related to the rated impeller. The flow volume element includes the ext:ReferencePropertyAttributeGroup. The impeller includes the ext:ReferencePropertyIDAttributeGroup, and impeller can be qualified as ‘Rated’. The referencePropertyID on the rated impeller will be equal to the referenceProperty attribute value on the flow volume. It is recommended that the identifier is in hexadecimal format, or is a meaningful string associated with a supporting data dictionary data item name. (The referenceProperty identifiers were constrained in schema to be in hexadecimal format before the development of cfiXML LT in Public Release V4.0 [B]. The constraint has been removed.)

As data items include the option of a ‘Nil’ value, this can allow for the referenced property value to be absent, but the specifications of the property to still be provided. For example, the value of the viscosity associated with the ‘Normal’ temperature of a fluid is required, but the value of the ‘Normal’ temperature is not known at this stage in the work process. The ‘Normal’ temperature element can still be defined in XML, but with a value of ‘Nil’ and the referencePropertyID hexadecimal or open string added. The viscosity referenceProperty can then be linked to the ‘Normal’ temperature ID. The value of the temperature can be added when available.

**L.9.3 usageContext Attribute**

The ext:usageContext attribute is used to define whether the data of an element is associated with conceptual, construction, design or operating performance specifications (see definitions in J.2.7). This removes the need for the repetition of similar elements. For example, in preliminary design processes, a conceptual operation of the unit may include an element which is also used in a design specification of the item. Following the modeling concept outlined [G.2], this element would therefore be included as a child of the unit itself, and the attribute usageContext used.

Certain elements are only associated with one usage context. These elements should be mapped as children of a suitably named parent (e.g. See Figure 13 unit/designSpecification, axialFan/operatingPerformance).

Usage context is available to all elements based on ext, simpleTypes and object. This covers all pq physical properties, ext:Strings, Booleans Doubles etc. – i.e. the elements in which ‘real’ data are defined, and of course the objects.

For a sequence element, usageContext can be made available by adding the usageContext attribute group.

UsageContext is specified for an element within schema by using the “fixed” attribute. It is specified within XML by giving the usageContext attribute or the element a value.
Rules for the use of usageContext in schema and xml
1) When usageContext is specified on an element, then all elements below (found by recursive decent from this element) will be assumed to be of the same usageContext. All the elements below this element are considered in the ‘scope’ of the usageContext.

2) If usageContext is to be specified, then it should be done at the highest appropriate level within the schema or XML (i.e. bearing rule 1 in mind, all descendant elements will be of the specified usageContext).

3) XML Processing Rule: If an object with objectState = “fullXML” is referred to or contained by an element within the scope of a specified usageContext, then the usageContext element of this object must match that of the scope. For example, if a stream contained within a pump, references a fullXML material stream stored elsewhere, then both usageContexts must be the same.

4) If an element is used in XML with a usageContext, then any other occurrences of this element must have the usageContext specified (even as 'Unspecified'). If there is a filter on the same property then all properties must have filters to distinguish them.

Along with definitions of usage context options, example XPaths and XML are provided [J.2.7].

L.10 Creating an object
A complexType may be extended from any of the following existing objects:

- BaseObject
- Object
- BulkItem
- EquipmentItem
- FabricatedItem

Use the following criteria to decide which option to use.

1. If there is a need to create and track a unique name or identifier for the complexType element, because other content models elsewhere in the schema may wish to make reference to it, then it should be at least extended from BaseObject.

2. If the complexType element data requires version management, then extend it from Object.

3. If the data associated with the complexType element will require manufacturer, tag, cost, weight and other data similar to other Facility Items then should be at least extended from BulkItem. If the item is pre-manufactured and mass produced then this provides further indication that it may well be a BulkItem.

4. If there are piping connections, it is custom-built or not a totally standard off-the-shelf item, extend the complexType from EquipmentItem.
5. If it is the type of item that is purpose-built or a one-off special from multiple parts, and commonly built on site, extend it from FabricatedItem.

Always ensure that the complexType does not exist in schema already, bearing in mind possible synonyms.

**M. CONVENTIONS – NAMING ELEMENTS AND FILES, ANNOTATION, VERSION NUMBERS**

**M.1 Naming Elements**

**M.1.1 Use of Case**

Use name case (camel case), i.e. upper case letters as separators, lower case for the rest of the word. No underbars “_”, e.g., hereIsOneName and not here_is_one_name or here_is_one_name.

Simple and Complex types should use an upper case first letter. All other element names and attribute names should start with a lower case letter.

If there are widely used and understood acronyms that use capitalized letters, then they should be used within the name of an element, but if the element name starts with the acronym, then all the letters should be lower case e.g., temaType.

Special characters are not available for use as element names in XML. Certain special characters can, however, be used in enumerations (for details of which see http://www.w3.org/documentation).

**M.1.2 Prefix and Suffix Codes**

Prefixes should only be used for XSD types and not for element or attribute names. Suffixes are not used in general.

*Rationale for Prefix and Suffix Codes*

The prefix codes make the XML harder to read and distract from its meaning.

**M.1.3 Use of Standard Abbreviations and Acronyms**

While fully spelled-out terms are preferred for schema element, attribute and type names, commonly used acronyms and abbreviations are permitted in a few cases, provided they are broadly recognized within the subject domain, e.g., API, max, MAWP, NPSH. Also, there should be no potential for the abbreviation to be misinterpreted and abbreviations should always be used the same way. These are maintained in an abbreviation table to ensure consistency of abbreviation usage [Table 14].

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>A</td>
<td>abstract type suffix</td>
</tr>
<tr>
<td>AC</td>
<td>alternating current</td>
</tr>
<tr>
<td>API</td>
<td>American Petroleum Institute</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>BOD</td>
<td>biological oxygen demand</td>
</tr>
<tr>
<td>CO2</td>
<td>carbon dioxide</td>
</tr>
<tr>
<td>COD</td>
<td>chemical oxygen demand</td>
</tr>
<tr>
<td>CODEN</td>
<td>CODEN identification for journals</td>
</tr>
<tr>
<td>coef</td>
<td>coefficient</td>
</tr>
<tr>
<td>cp</td>
<td>heat capacity at constant pressure</td>
</tr>
<tr>
<td>cs</td>
<td>heat capacity at saturation</td>
</tr>
<tr>
<td>cv</td>
<td>heat capacity at constant volume</td>
</tr>
<tr>
<td>DC</td>
<td>direct current</td>
</tr>
<tr>
<td>E</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Fx</td>
<td>force in the x direction</td>
</tr>
<tr>
<td>Fy</td>
<td>force in the y direction</td>
</tr>
<tr>
<td>Fz</td>
<td>force in the z direction</td>
</tr>
<tr>
<td>GUID</td>
<td>globally unique identifier</td>
</tr>
<tr>
<td>H2</td>
<td>hydrogen</td>
</tr>
<tr>
<td>H2S</td>
<td>hydrogen sulfide</td>
</tr>
<tr>
<td>hex</td>
<td>hexadecimal</td>
</tr>
<tr>
<td>HRSG</td>
<td>heat recovery steam generation</td>
</tr>
<tr>
<td>HTML</td>
<td>hypertext markup language</td>
</tr>
<tr>
<td>ID</td>
<td>identifier</td>
</tr>
<tr>
<td>IP</td>
<td>I (current) to P (pressure)</td>
</tr>
<tr>
<td>IR</td>
<td>infrared</td>
</tr>
<tr>
<td>ISBN</td>
<td>international standard book number</td>
</tr>
<tr>
<td>ISO</td>
<td>international standards organization</td>
</tr>
<tr>
<td>ISSN</td>
<td>international standard subscription number</td>
</tr>
<tr>
<td>JT</td>
<td>Joule-Thomson</td>
</tr>
<tr>
<td>LMTD</td>
<td>log mean temperature difference</td>
</tr>
<tr>
<td>log</td>
<td>logarithm</td>
</tr>
<tr>
<td>MAWP</td>
<td>maximum allowable working pressure</td>
</tr>
<tr>
<td>max</td>
<td>maximum</td>
</tr>
<tr>
<td>min</td>
<td>minimum</td>
</tr>
<tr>
<td>MTD</td>
<td>mean temperature difference</td>
</tr>
<tr>
<td>Mx</td>
<td>moment force in the x direction</td>
</tr>
<tr>
<td>My</td>
<td>moment force in the y direction</td>
</tr>
<tr>
<td>Mz</td>
<td>moment force in the z direction</td>
</tr>
<tr>
<td>NPSH</td>
<td>net positive suction head</td>
</tr>
<tr>
<td>NPSHR</td>
<td>net positive suction head required</td>
</tr>
<tr>
<td>NRTL</td>
<td>non-Random Two Liquid method for predicting fluid properties</td>
</tr>
<tr>
<td>O2</td>
<td>oxygen</td>
</tr>
<tr>
<td>P</td>
<td>pressure used as a physical quantity</td>
</tr>
<tr>
<td>PNA</td>
<td>Parafinic-Naphthenic-Aromatic</td>
</tr>
<tr>
<td>ppm</td>
<td>parts per million</td>
</tr>
<tr>
<td>PR</td>
<td>Peng-Robinson equation of state method for predicting fluid properties</td>
</tr>
<tr>
<td>QA</td>
<td>quality assurance</td>
</tr>
<tr>
<td>RPM</td>
<td>revolutions per minute</td>
</tr>
<tr>
<td>RTD</td>
<td>resistance temperature detector</td>
</tr>
<tr>
<td>SI</td>
<td>Systeme Internationale</td>
</tr>
<tr>
<td>SMILES</td>
<td>SMILES is a way of expressing chemical formulas</td>
</tr>
<tr>
<td>spec</td>
<td>specification</td>
</tr>
<tr>
<td>SRK</td>
<td>Soave-Redlich-Kwong equation of state method for predicting fluid properties</td>
</tr>
</tbody>
</table>
### Table 14 Abbreviations and acronyms accepted in schema

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub</td>
<td>subordinate</td>
</tr>
<tr>
<td>T</td>
<td>temperature</td>
</tr>
<tr>
<td>TEMA</td>
<td>Thermal Equipment Manufacturer's Association</td>
</tr>
<tr>
<td>TOD</td>
<td>theoretical oxygen demand</td>
</tr>
<tr>
<td>UMI</td>
<td>University Microfilms Incorporated</td>
</tr>
<tr>
<td>UPS</td>
<td>uninterruptible power supply</td>
</tr>
<tr>
<td>URI</td>
<td>universal resource indicator</td>
</tr>
<tr>
<td>URL</td>
<td>universal resource locator</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>XML</td>
<td>extensible markup language</td>
</tr>
<tr>
<td>XPath</td>
<td>XPath</td>
</tr>
<tr>
<td>XPointer</td>
<td>XPointer - a URI plus an XPath</td>
</tr>
</tbody>
</table>

### M.1.4 Use of Units of measurement

Units of measurement should be avoided in element names as they are specifically included in the pq schema and may vary for the same physical property as different parties require. Application software can use the units of measure names to generate the appropriate unit symbols in user displays and reports. There are, however, a few valid exceptions to this rule when the units of measurement are effectively the physical property name. This includes ‘Amperes’ and ‘Voltage’, for which the units are always some level of current or voltage (e.g. milli-volt or volt, milli-amp or amp).

### M.1.5 Company names, associations, non-international standard cataloguing or identification systems

As far as possible, no company names, associations, non-international standard cataloguing or identification systems should be included as elements or attributes in the XML Schema Definition (XSD) files. However, it has been found that certain association standards are required to fully describe particular properties, and though not international, are well recognized in the industry. For example, material standards are commonly used (e.g. American Society of Mechanical Engineers or American Society of Testing and Materials standard material names), as well as other standards from organizations and equipment testing or listing agencies. When appropriate or possible, annotation in the schema should give the details of the organization. It is envisaged that a compilation of well-known and well-used standards will be developed over time in the schema model as driven by business requirements and industry usage scenarios.

**Rationale for Company names, association etc.**

*This will ensure quick and non-contentious approval of cfiXML and maximum applicability. Enumerations may be used for such items since they can easily be extended.*

### M.2 Annotation – xsd:appinfo and xsd:documentation

Annotation should be included throughout the schema to enable it to be used for automatically supporting user interfaces and auto-generated documentation.
The documentation element should be used for full descriptions of the element, attribute or type and may contain several lines. This documentation should explain any aspects of the element’s purpose that are not obvious.

The appinfo element provides information to applications using the schema and can contain both data to be used in the application and additional validation information. appinfo can contain any elements.

M.2.1 appShortName and appLongName

At present, the only elements used in appinfo are appShortName and appLongName which provide text to be displayed by the application. appShortName is intended for use in captions or narrow columns, while appLongName provides a longer name for use in reports and full-width lists of data. The recommended text should be a maximum length of 20 and 50 characters respectively, including spaces between words (or abbreviations, as necessary) and no camel case. This is not enforced, but adhering to these will ensure that the text should display correctly in cfiXML-based applications.

Both elements are identified as belonging to the appcfi (http://www.cfixml.org/appcfi) namespace. No schema file exists for this namespace, but it is more convenient to locate the elements in this namespace than to leave them belonging to the default namespace, e.g. http://www.cfixml.org/obj. The appinfo elements can then always be identified as appcfi:longName rather than having to specify a different fully qualified name for each schema.

Example:
<xsd:element name="elecClassifArea" type="etl:ElecAreaClassif" minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>Electrical classification area for the equipment item</xsd:documentation>
    <xsd:appinfo>
      <appcfi:longName>Electrical classification area</appcfi:longName>
      <appcfi:shortName>Elec Class Area</appcfi:shortName>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>

M.3 File Headers

Each capital facilities industry XML Schema file contains a standard header of the copyright and legal notices as given in www.cfiXML.org/Copyright.htm and the “IP Notice.doc” file. This is contained in comment rather than an annotation element to reduce the load on the parser. Annotation at the global level includes information on the current version, author and date of release. A copy of the full open source license agreement from Fiatech is available [cfidev documents].

M.4 Version Numbers

Each capital facilities industry XML Schema file has a version number, an attribute which is set in the root <schema> element. This should be incremented in an appropriate
manner whenever changes are made to the schema file. Applications can then use this to ensure that software is compatible with the corresponding schema.

**M.5 Schema Files**

The convention for naming schema files is illustrated [C.5.1].
N. SPECIAL MODELING CONCEPTS

N.1 Use of Substitution Groups
In Public Release V 4.0, polymorphism using xsi:type was replaced. A detailed review of which options would provide the same ‘choice’ functionality concluded that substitution groups would provide the best option for both cfiXML LT and cfiXML Full users [N.1.5].

Example of use of substitution group - attached drivers:
Elements of a substitution group must be of the same type as the ‘head’ element of the group or of a type derived from the ‘head’ element’s type. For example, if the head element is of type [eqRot:DriverItem] all members of the substitution group must be of type [eqRot:DriverItem] or of a type derived from [eqRot:DriverItem].

N.1.1 Schema development considerations
When using substitutionGroup, it must be noted that the attribute can only occur once on the same element. This means that if an element is required to be part of a substitution group in more than one context in the schema, then a new element will have to be created.

N.1.2 XML content
The xsi:type attribute is not to be used on the head element (as in derivable types) otherwise there is more than one way to map to the same data elements (via a substitution or via xsi:type) and the resulting XML differs. Use of the xsi:type attribute cannot be blocked in schema while still supporting substitution groups.

N.1.3 Application support
Previous use of derivable types [N.2] in schema provided some constraint on applications in that the namespace defined by the XXXType element attribute xsiTypeNS was the same as the namespace defined by the XXXItem xsi:type. However, further constraint was provided by the application when more than one derived type occurred in the same namespace. The original XXXType elements with xsiTypeNS attributes are currently retained in schema.

Applications could also identify the base object common to all of the members of a derived type ‘group’ to ensure objectIDs could be maintained if a user of the application altered their selection. (For example, if a user refines their choice from a turbine to a steam turbine, the common base object is the eqRot:Driver.) The “head” element structure of substitution groups should allow similar application support.

N.1.4 Enumeration format for substitution groups
An “elementNameType” enumeration list may include data items which are included in a substitutionGroup and data items which are not. This is because the enumeration list may be a required primary data element. The enumerations, which represent substitution group members, are in camel case format. The enumerations, which do not represent substitution group members, are lower case, normal text. [L.4]
N.1.5 **Rationale**

XML serialization tools, such as XML Beans, do not support xsi:type polymorphism use in schema and are unable to automatically cast associated Java objects. The cast can be made explicitly in the code for each occurrence of a derivable type, but this significantly increases development time and complexity. To support LT schema, an alternative ‘choice’ mechanism is a better option.

**Derivable type rationale and structure in full schema**

Details of derivable type polymorphism in full cfiXML are outlined below [N.2]. The derivable type use in schema was for the selection between elements which have a common base and this can be a useful means of identifying members of the same ‘group’ when developing applications. Also, the collection of elements can be verified automatically by a schema validator checking the type against the base type.

**Re-assessment of Choice**

The use of ‘Choice’ as a suitable replacement option for derivable type polymorphism has significant implications for application developers using cfiXML Full, in that each choice must be defined fully in code. This may not be an issue for XML serialization tool users, but more sophisticated applications can be better supported by other schema structures, such as substitution groups.

There is no requirement for each element associated with a choice to have any common type base. An external authority would be required to confirm whether the collection of elements is valid, that is, human input is needed to choose which elements are allowed.

**Substitution Groups**

A schema based option for applications which do not support the use of polymorphic derivable types is the use of the substitutionGroup attribute on elements in separate namespaces. All members of the same substitutionGroup can be searched for automatically throughout schema as they must be extended from the same base type. The use of xsi:type is problematic for XML serialization tools and as the xsi:type attribute is not required when elements are substituted, this resolves the issue.

N.2 **Use of Derivable Types (xsi:type and xsiTypeNS) prior to Public Release V 4.0**

(*Information provided for users of previous versions of cfiXML.*)

As the preferred mechanism to using a “choice” content model, an element can be made to represent one of a number of possible options by adding the element to the schema with the xsi:type attribute identifying it as a derivable type. The element is created as a base complex type, which may include data elements covering ‘general’ data that are applicable to all the derived types. To include information about the types that are available, a pair of similarly named elements is used, as shown in the following example:

**Example:** The following are child elements of ‘endDescription’
endConnectionItem  an element based on type pipingComponent from which multiple connection items types are derived.

endConnectionType  an enumeration element containing the types that are expected for endConnectionItem

The “elementNameType” (e.g. endConnectionType) includes an attribute xsiTypeNS of type xsd:string, which is used to store the namespace prefix used in the xsi:type attribute in “elementNameItem” (e.g. endConnectionItem). (Note that the xsi:type attribute is part of W3C Standard Schema XML.)

In schema, documentation on derivable type element pairs has been standardized to ensure that similar, clear information is always provided (see Figure 15). The full list of derivable type options available from endConnectionItem are found in endConnectionType, but documentation under the “-Item” gives the user an indication of key “-Type” enumerations available.

![Diagram of endConnectionItem and endConnectionType]

Figure 15 Example of derivable types - end connections

N.3  Use of AlternativeOption attribute

The attribute ‘alternativeOption’ in the BaseAttributeGroup [D.1.2] has two functions, as follows.

1) One use is to be able to pick more then one enumeration from the same simpleType enumeration list (as shown in the following example). The element in the schema to which this attribute is attached must have multiplicity on it.

Example: XML example wherein a block valve requires that the gateWedgeType is either Solid or Flexible:
<eq:blockValve>
  <eq:blockValveTrim>
    <eq:closureMember>
      <eq:gateWedgeType alternativeOption="true">Solid</eq:gateWedgeType>
    </eq:closureMember>
  </eq:blockValveTrim>
</eq:blockValve>
2) The other purpose of alternativeOption is to allow different child elements to be considered as alternative options. In Figure 16, the parent element ‘material’ based on complexType ‘Material’, has multiplicity on it and the attribute alternativeOption has been added. This allows for different materials, based on child elements of material, to be described. For example, a material could be stainless steel (materialType) with a filling of another material (auxiliaryMaterial), or alternatively, just carbon steel that has been cast (processType). Example XML is provided [J.2.10].

![Diagram of material elements]

**Figure 16** Example use of alternativeOption on a parent element (mtrl:material)
N.4 key and keyref
The XML Schema key/keyref mechanisms are used within a single document where they enforce unique values for the key elements and references to these unique values. They are used in cfiXML to provide the association of a value in a cell to the column and header information within a table.

The cfiXML does not make extensive use of these mechanisms as extensive cross value validation is often too dependent on the usage scenario. In particular they are not used to enforce object validation: The cfiXML schema is designed to provide an integrated multi-resource network of related information. This includes referential integrity of objects across multiple resources (see description of the use of cfiXML objects [E]). The rationale is that to be able to support containment and referencing of sub-objects it is necessary to allow multiple occurrences of the same identifier where one may be the full XML and the others references to it. However, since this is an XML level option and not fixed at the XSD level, key cannot be used globally on objectID as it enforces unique identifiers within the file.

The key and keyref mechanism is used directly within a file for tables and has been used to enforce validation within the mtrl:PropertyDataTable complex type.

Also key and keyref are used where a single large structure or table contains master lists of items that are referred to from multiple other locations in the structure or table. An example of this is the mtrl:PropertyDataTable. However, if the items referred to may ever be located in another file, the references should be defined as full cfiXML objects which support the contained or referenced mechanism, including references to other files or objects via a registry.

N.5 Modeling Reference Relationships
When modeling the relationship between two objects, only one of the two objects should reference the other if possible. This maintains the principle of storing information in only one location to ensure that inconsistencies are not possible. For object relationships, the relationship should be maintained in only one of the two or more objects. If complex multi-directional relationships are required it is better to record this information in a completely separate location such as a connection table.

N.6 Material Streams
Consider a plant facility where a process stream will pass from one equipment item via piping into a valve then via piping into another equipment item. The means of modeling this situation is provided by having a material stream object external to any equipment item, so that the same stream can be referred to by multiple items (e.g. valve, or piping). This material stream would be of objectState ‘fullXML’.

When defining specifications for a particular equipment item, the associated stream properties may need to be defined (e.g. inlet temperature). This situation is handled by having a stream object element contained within the equipment item itself. As a cfiXML object can be referenced, this contained stream can be objectState ‘refLocal’ and
therefore reference to the main (external to the equipment item) fullXML material stream.

In a conceptual specification, piping is assumed to be ‘perfect’, so that there are no losses (e.g. of temperature or pressure). In this case, the material streams can be considered as passing directly from one key equipment item into another. In all other usage contexts, the full details of the connecting pipeline etc. would need to be included.

The schema model can therefore meet whatever level of detail a user may wish to define. This schema design is illustrated by the outline of key schema components for heat exchangers and centrifugal pumps etc.[Appendices S-Y].

The effects of this modeling on XML are best illustrated by viewing one of the provided XML example file, such as …/examples/ CentrifugalPumpTest2.xml. If the user searches for uo:materialStream, the first occurrence of this object in the example file will be found to have an objectState of fullXML. If the associated objectID is then searched for, a stream object will be found to have the same objectID, but objectState of refLocal. This is the contained stream object [E.4].

Parties involved in data exchange or storage can determine what level of detail they may wish to use when representing material streams. In contrast to the example XML provided, it is, of course, possible to set the contained stream to fullXML and not refer to any higher level uo:MaterialStream.

Note also that the materialStream is the head element of a substitutionGroup [N.1], so that access to utility streams is also possible.

**N.7 Order Element or Order Attributes for Lists and Tables**

Two principle methods are used to describe array lists and simple tables in schema. The level of complexity of the data associated with the list or table properties is the key factor determining which method is used.

**N.7.1 Order Element**

When properties in an array list or in a simple table are potentially associated with relatively detailed additional specification or metadata information then a parent element of type “order” is used.

For example, when considering a material stream passing through a heat exchanger, details of the mass or mole fractions of each component within the material component list may be required, and the associated critical temperatures and pressures. In schema, mtrl:materialComponentList/mtrl:orderedComponent/mtrl:order provides the structure to identify each material component by an order integer. In the same sequence as the order element is mtrl:materialComponent, which provides access to detailed molecular, structural and identification specifications, as well as material properties and their associated context data.
The object mtrl:materialComponentList is provided in existing schema files such as eqHxDoc.xsd, external to the equipment item as well as contained within the equipment item via uo:materialStream/mtrl:materialProperty/mtrl:material/mtrl:materialComposition. This allows for details of material components to be stored externally to equipment items. Details of the material components within the stream passing through the equipment item can be referred to as required via the objectID and objectState mechanism. The mtrl:materialComposition/mtrl:composition element (a sibling of the contained mtrl:materialComponentList) provides access to material component amounts, masses, volumes etc.

In summary, the use of the order element in this example allows each component in a material component list to be flagged, and all the detailed specifications, including identifiers, properties and relative quantities, to be associated to each component via the order integer.

**N.7.2 Order Attribute and Order Attribute Group**

When lists and tables fundamentally consist of variable values, perhaps with a few data constraints or qualifiers, then the order attribute method is used. For example, in mtrl:property, all the property elements have multiplicity and include the ext:OrderAttributeGroup so they can all be used in array lists or tables. Similarly, eqRot:CentrifugalPump performance curve data properties include the order attribute (see example XML [J.2.11]).

To enable a value to be associated with one or more other values, as in an array list and simple tables, an order attribute or the ext:OrderAttributeGroup is used. Both include ‘order’, a positive integer which is used sequentially starting from 1. The OrderAttributeGroup also includes index1 and index2. When using the index1 and index2 attributes within the order attribute group, the indexes must refer to elements in their alpha-numeric order.

An example of the simplest case for using the order attribute is when one variable is associated with only ONE other variable:

<table>
<thead>
<tr>
<th>Temperature</th>
<th>123</th>
<th>134</th>
<th>145</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viscosity</td>
<td>40</td>
<td>50</td>
<td>60</td>
</tr>
</tbody>
</table>

The temperature elements would be:

```xml
<t order="1">123</t>
<t order="2">134</t>
<t order="3">145</t>
```

```xml
<viscosity order="1">40</viscosity>
<viscosity order="2">50</viscosity>
<viscosity order="3">60</viscosity>
```
However, if a variable is associated with the value of 2 other variables, the orderAttributeGroup index1 and index2 are also required, such as in this example:

<table>
<thead>
<tr>
<th></th>
<th>Pressure 1: 10</th>
<th>Pressure 2: 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature 1: 123</td>
<td>kValue1 (P1 &amp; T1): 10</td>
<td>kValue2 (P2 &amp; T1): 20</td>
</tr>
<tr>
<td>Temperature 2: 134</td>
<td>kValue3 (P1 &amp; T2): 30</td>
<td>kValue4 (P2 &amp; T2): 40</td>
</tr>
</tbody>
</table>

The temperature elements would be:

\[
\text{<t order="T1">123</t>}
\]

\[
\text{<t order="T2">134</t>}
\]

The pressure elements would be:

\[
\text{<p order="P1">10</p>}
\]

\[
\text{<p order="P2">20</p>}
\]

The kValue elements would be:

\[
\text{<kValue index1="1" index2="1">10</kValue>}
\]

\[
\text{<kValue index1="2" index2="1">20</kValue>}
\]

\[
\text{<kValue index1="1" index2="2">30</kValue>}
\]

\[
\text{<kValue index1="2" index2="2">40</kValue>}
\]

Note: The kValue index1 attribute contains the associated order from the pressure elements and the index2 the associated order for the temperature. That is, the indexes on the kValue are given in the order according to the alphabetic order for P and T.

### N.8 Use of PropertyTable

If complex data tables are required, such as heat release curves for process material streams, then the structure to represent columns, with full header details, rows and cells is available. An outline model of this structure is provided in ext:PropertyTable and a developed example is found in mtrl:propertyDataTable/mtrl:propertyTable. Note that the column and cell elements include the compulsory ‘order’ attribute, which is used to link each cell to the correct column. Some use of key and keyref is used to enforce column definition and cell and column order links. Also, mtrl:propertyDataTable includes key and keyref constraints ensuring, for example, that specific material properties are correctly defined with their associated phase references.

### N.9 Equipment Connections

Any piece of equipment that is defined as an object extended from the base complex type eq:EquipItem, contains the element eq:equipmentConnection. This object includes the child element of eq:endDescription, leading into the elements of eq:endConnectionItem and eq:endConnectionType. As an example of a substitution group, in this case, the endConnectionItem is the ‘head’ element of type eq:PipingComponent. Any element extended from the same base type and with the “endConnectionItem” substitutionGroup attribute is available. Thus, details of the
connections available on the main item of equipment under consideration can be described.

However, this main equipment item is attached to other connections on items of equipment with which it is associated. The element eq:equipmentConnection also contains the child elements of eq:attachedConnection and eq:attachedEquipment. The former is based on eq:EquipmentConnection and therefore provides all the elements in this parent. Hence, details of the attached connections can be described. The element eq:attachedEquipment is based on eq:EquipmentItem, so information on the attached equipment can also be included.

In addition, it has been found that description of associated equipment items as a child of a data sheet may also be necessary. For example, in the control valve data sheet, a sibling of the main unit, i.e. the eq:controlValve, is eq:EquipmentItem. This is clearly documented as the equipment item to which the valve is related. If it is found necessary to describe an equipment item related to the main unit, in detail, it is therefore available.

**N.10 Life Cycle Status**

Figure 4 illustrates work processes involved in the capital facilities industry. The cfiXML schema provides a combination of elements and attributes which can be used to define the life cycle status of an object at any stage through this process. The level of detail that is required is up to the parties involved in the data exchange and storage process.

1) The dx:DataSheetHeader element allows a ‘document’ or effectively a package of data, to be tagged with the dx:dataSheetType of ‘Budget quotation’, ‘Request For Quote (RFQ), ‘Care, install and spares’ etc. This provides metadata to describe the compiled package of data to be exchanged.

2) The obj:Object transactions provides information on who created, modified, approved or issued data and when. This information also includes a “completedBy” element that provides the role of the person filling in the data.

3) The obj:Object element lifeCycleStage includes enumerations such as “Front End Engineering Design (FEED)”, “Operation and maintenance” and “Construction”. This describes the stage in the complete life cycle of the process plant or facility that the object and associated data represent. As a data sheet is an object, this can be used to identify the life cycle stage to which the data sheet applies. For example, the “Detailed design” data will be substantially different from the “Operation and maintenance” stage data.

**N.10.1 lifeCycleStage vs usageContext**

The usageContext attribute indicates the perspective or view of the data being described. It is related to how the data are obtained or specified. For example, “Conceptual specification” data are created by completing heat and material balance calculations, often by running simulation programs. The equipment items are modeled as unit
operations that transform the streams that connect them. Contrasted to this is “Operating performance” data that are measured on an actual working plant.

The lifeCycleStage attribute only indicates the stage in the life cycle of the plant or equipement, for which the information applies. A “Detailed design” lifeCycleStage may include “Conceptual specification” (usageContext) information that is then multiplied by a design factor to provide “Design specification” (usageContext) data.

**O. MISCELLANEOUS GUIDELINES FOR SCHEMA DEVELOPERS**

Do not use magic numbers. Using a negative number, 0, very large or other number to have special meaning confuses the design and is likely to be overlooked or not understood at a later date. Rather use additional elements of Boolean or enumeration data types in a sequence to serve the same purpose. In particular, XML does have a “Nil’ value.

Use easily searchable keywords and comments to guide future schema developers who will be working on the Schema to extend or fix problems. Use the following keywords at the beginning of such comments:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TODO</td>
<td>Further work required here.</td>
</tr>
<tr>
<td>:BUG</td>
<td>There’s a known bug here – include an explanation of the bug or a bug ID.</td>
</tr>
<tr>
<td>:KLUDGE</td>
<td>Something ugly here that could be improved at a later date. Explain the problem.</td>
</tr>
<tr>
<td>:TRICKY</td>
<td>The following section is tricky and anyone making changes in it should do so with caution because…</td>
</tr>
<tr>
<td>:WARNING</td>
<td>Beware of something</td>
</tr>
<tr>
<td>:ATTRIBUTE</td>
<td>Further user defined keywords</td>
</tr>
</tbody>
</table>

The initials of the schema developer and the date should follow the keyword.

Use default values as much as possible to ensure that schema is initialized to a sensible state when it is created. Undefined values can cause lots of problems. Even setting a default of Nil may be more meaningful than not defined at all.

Use of small schema files that provide specific features is a good idea and helps to support reuse.

**O.1 Important XML Schema Features**

These are W3C.org standard requirements and provided here as a reminder.

**O.1.1 Occurrences**

Attributes   Must occur once or not at all.
Element    The default is once.

O.1.2 Defaults
Default attribute values apply when attributes are missing
Default element values apply when elements are empty.

O.1.3 FormDefaults
Regardless of the elementFormDefault flag, global elements must ALWAYS be qualified
with the namespace prefix (unless there is a default namespace specified in the instance
document.)
P. Extending cfiXML Schemas

Extensibility is an important part of the schema design. The XML files validated against a schema must be able to include elements that were not included in the original design. It must also be possible for users to extend the schemas on local systems. The cfiXML framework supplies schema that can be used as is, or can be readily extended as required by two parties who need to exchange data that are not contained in the current schemas without having to edit the base standard schema files. Examples of how to undertake schema extensions are provided [S.5.2; S.5.3].

Extension of existing standard elements in the schema by editing the published xsd files is not recommended at any time, regardless of how the original XSD was structured or how the extension is done.

A robust system to allow schema developers to report issues and request changes to existing cfiXML schema is available. Details can be found on the http://www.cfixml.org web site and the cfidev project on SourceForge. Opening an account with SourceForge will allow access to the development web site to be able to report issues.

P.1 Extensibility mechanism without adding schema definitions

Extension mechanisms are provided in the schema design so that XML instance files may be extended without having to provide any additional schema definitions, as follows.

3) Each element “sequence” and “enumeration” list in the schema allows the user to insert additional custom elements, provided they are in “well-formed” XML syntax, further described below. This is the “customXXX” element method.

4) The enumeration list for each enumeration data type in the schema can be extended by using the “otherValue” mechanism, described below.

The advantage of these approaches is that they are easy to use and do not require any schema change at all.

The disadvantage is that data validation is the responsibility of the receiving and exporting application software. No automated data validation by the XML parser is available.

P.1.1 Extending elements in schema using the “customXXX” element

User elements may be added to any content model in the schema. The last element in each sequence throughout the entire schema is, by convention, named “customXXX” where “XXX” is the name of the parent element of the contained sequence. The “customXXX” element is specified to be of the type “ext:Custom” which is based on “ext:(any).” This allows any user-supplied additional elements to be appended to the sequence and the XML instance files will be treated with ‘lax’ validation by the parser. This means the parser will effectively ignore the extension elements for validation processing as long as the XML is “well-defined,” i.e., follows basic XML syntax rules. This leaves the entire responsibility for data validation to the applications that use the extensions. See the tutorial S.5.2 for example XML.
The custom (any) mechanism is provided to enable extension of sequences and enumeration lists in such a way that the XML produced will still be valid against cfiXML schema whether or not the recipient has the extension schema. If a user attempts to redefine elements within a sequence, i.e. not using the recommended extension mechanisms, then it is unlikely that the resulting XML will successfully validate against the original schema.

P.1.2 Extending enumeration elements using the “OtherValue” method

The attribute group “ext.BaseEnumerationAttributeGroup”, which is applied to all elements associated with simpleType enumerations, allows for an “otherValue” attribute to be added when the enumeration list does not include the value required. Most enumeration lists include the final 3 enumeration values of “Other”, “Unspecified” and “custom” in this order as a standard (i.e. they do not occur in alpha order as part of the full list of enumerations). If a value of “Other” is chosen from the enumeration list, then the attribute “otherValue” can optionally be used if the data are available. This is illustrated by an example provided [S.5.3].

P.2 Extensibility mechanism through user-provided schema definitions

Extension mechanisms are provided in the schema design so that XML instance files may be extended through user-provided schema extensions. This captures the benefit of XML parser validation for user extended schemas. These mechanisms are:

1) Each element “sequence” or “enumeration” list in the schema provides the ability for the user to insert additional custom elements and arbitrarily complex element structures by adding these schema elements to a separate user custom schema file. This allows users to extend schema while still getting the benefit of parser validation yet not affecting the standard published schema.

2) The enumeration list for each enumeration data type element in the schema can be extended by users who provide “custom” schema extensions in a custom user schema definition file.

3) New engineering or other data transaction documents to support specific data transactions can be created by simply defining new and separate document schema files. These document schemas are essentially custom collections of engineering objects that are needed to support a specific usage scenario.

We recommend that user schema extensions be contained in separate user extension namespaces and that these are named as follows:

“http://www.cfixml.org/custom/xxx-com/yyyy”

Where “xxx-com” is an organization’s domain name (substituting hyphen ‘-‘ for the period ‘.’ in the URL) and “yyy” is defined by the user organization however they choose to provide as many namespaces as needed. For multiple user namespaces, simply use different “yyy” designations as needed.
The custom namespace for the ‘eq’ customization namespace provided by cfiXML.org is “http://www.cfixml.org/custom/cfixml-org/eq”

Further, we recommend, that the user-defined schemas be placed in the folder structure provided by cfiXML under the “custom” subfolder of the root cfiXML folder that uses the same naming convention. That is, in a separate subfolder called “schema/custom”

[cfiXML]
  [documentation]
  [examples]
  [schema]
    [custom] (Implementer specific customized schemas)
      [bechtel-com]
      [cfixml-org]
      [alarsoftware-com]
      [htri-net]
      [nist-gov]
        abc.xsd - implementer specific schema
        [xrom1]
        pqr.xsd - site specific schema
        (other organization-specific schema folders)

[document]

No attribute extension mechanism is included other than adding additional enumeration list items.

**P.2.1 Adding an Additional Element**

If you want to use the XML parser to validate new elements (advisable for larger extensions), then the way to do this is by defining a custom namespace schema and custom schema definition file using the namespace and file naming conventions described previously. See the tutorial for an example [S.5.2].

This extension mechanism is very powerful for defining extensions that are not already in the schema. It is advantageous to applications, because data validation is done directly by the XML parser, not the application code.

**P.2.2 Extending an Enumeration List**

Each enumeration in the schema is specified as a combination of a simpleType and an element definition based on the simpleType. The simpleType contains the values for the enumeration strings. One of these choices is always “custom.” The advantage of using this method for extending enumeration types is that the XML parser will provide validation of the user extension choices instead of passing this responsibility to the application software. See the tutorial for an example [S.5.2; S.5.3].
Q. APPLICATION DEVELOPERS RESOURCES

A full cfiXML-based application developer should be familiar with the contents of this document.
The design of cfiXML LT is to enable developers with limited XML experience to be able to develop data exchange using XML serialization tools and XPaths provided in a data dictionary. LT features are described in [B], and more details available in cfiXML Light (LT) Specifications.

Q.1 The basic steps for setting up an XML import and export interface application software

1) Determine what data need to be read into or written from the software application to support one or more usage scenarios. Interview users about the data they require transferred to and from the application.

2) Review the existing XML ‘document’ schemas to see if there are existing XML document definitions that can be used, e.g., see the eqRotDoc:centrifugalPumpDataSheet element which is defined in the ‘document/eqRotDoc.xsd’ file.

3) If existing exchange documents do not meet requirements, consider defining a custom document schema definition file. If the scope of data exchange is narrow, (e.g., contained within a single global element) then consider using one of the global elements defined in any of the schema definition files.

4) Write any additional custom user schema required or if the application will be widely used, work with the cfidev project to extend the standard published schemas as needed.

5) Establish how the information is stored in the application and what programming interfaces are required to put information into application storage on import, and to get information out of application storage for export.

6) Design and document a mapping table between the data structures in the application for the data to be transferred. In one column define the storage location and function calls required to get or put information from and to the application storage. In a second column define the Xpath to the element in the cfiXML schemas that match the data items to be transferred. Xpath query language also used by XML Stylesheet Language Tranformations (XSLT) provides a unique path definition to data contained within an XML instance file.

7) Write application code for “import” and “export” that uses an XML parser and the DOM or SAX application programming interface. Use the parser to get or put information into an XML instance file and your application storage according to the mapping table you defined in step 4.

While XML schema files can be reviewed in native form, we highly recommend that XML schema files should be reviewed in an XML schema interactive development environment. However for the purpose of this discussion, we will present the XML files in native text format.
Q.2 Tried and Tested Application Development

The cfiXML schemas have been tried and tested by application developers over the last few years including both pilot not-for-profit trials and commercial implementations. Some of the projects include:

- A Centrifugal Pump pilot project involving a large chemical company representing the buyer, pump software vendors and pump suppliers and manufacturers. The results of the pilot project were demonstrated to the American Petroleum Institute (API).\[A.5.1\]
- An interface between cfiXML and the Heat Transfer Research Institute (HTRI) XIST program.
- Microsoft Excel equipment data sheets with cfiXML import and export capability.

The AEX project and the Hydraulic Institute have been collaborating to define the data requirements associated with two commercial transactions, or usage scenarios, between users and suppliers of centrifugal pumps. These two transactions are the Bid Request for Quotation (BidRFQ) and Bid Quotation (BidQuote).

Q.3 Development Environments and Tools

Details of a comprehensive list of development environments and tools that have been tried and tested are available on the Fiatech web site (www.fiatech.org).

Q.4 The cfiXml.xsd File for Loading Schema into Software

The schema/document/cfiXml.xsd file has been created to include all schema document files and therefore all namespaces. This single file can therefore be loaded into applications rather than having to use multiple files. This should help developers to build software relatively quickly and simply and improve performance.

*For cfiXML LT:* Loading the full schema via the cfiXml.xsd file for an LT application is probably not required as a simple application using LT will only use a small piece of the cfiXML.

Q.5 Messaging Usage and Business Protocol Containers

The cfiXML engineering documents do not directly include the messaging and business process protocol schemas, which are being developed in other industry XML efforts such as OAGIS, UBL and RosettaNet.

The cfiXML documents are intended to be the “payload” documents contained within, or attached to any appropriate e-business messaging and business process architecture that is used by the industry.

The ctx namespace contains information that will be similar to or the same as the messaging and business protocols, including complex types for organization, person, location, and project information.
Q.6 Creating and Managing cfiXML Objects

Uniquely identifiable cfiXML objects enable both transient exchange of information and the maintenance of information in object repositories, ranging from a single file or small collection of files to large databases. The schema design allows any object element to contain the full XML data of the object or simply a reference to the object that resides locally or elsewhere. This allows application users to accommodate a broad range of usage scenarios for applications.

The most basic use of cfiXML is to support external data transactions, in which the XML files are totally self-contained, i.e. objects will only make use of a local referencing mechanism. In contrast, large indexed database systems may be used by organizations with access to object data maintained via some object registry or direct referencing. The following outlines application requirements to cater for these usage scenarios.

cfiXML application software needs to be able to use object referencing both within a file and across files, including files that may be accessed over the Internet.

For how to support revision management, versions and change tracking, see [Q.9].

Q.6.1 ObjectIDs

Format: namespacePrefix.type.idOrName
Example: proj.Project.BrownfieldUpgrade209515930

Application Implementation Recommended Practice: Neither the value of the namespacePrefix nor the type name is enforced by the schema. The pattern of three names separated by periods can be enforced if the application is only to support cfiXML Full [E.3.1]. Applications should provide the correct values for the namespacePrefix and the type name as other applications may rely on this information. User input should only be required for idOrName, and in most circumstances this is all that needs to be displayed in the application user interface. The application should ensure that the namespace and type are correct. Applications should ensure that the idOrName is indeed unique for every object. The recommended way to do this is to append either a 9-digit random number or a 9-digit CRC code\(^{11}\) to the end of the user-supplied idOrName string as illustrated in the above example.

Q.6.2 Editing Referenced Objects – Processing Required

Restrictions: This functionality ONLY applies to version tracked registered objects, i.e. referenced objects with objectState = “refViaReg”.

When an application allows a referenced object to be edited (by being ‘checked-out’ from an object resource), then it is necessary to create a new version of the object with the changed data. This is done by changing the objectState to “fullXml” and inserting the full XML into the containing object. The processing software will create a new index entry for the new version of the object, which then has the same objectID, but a new version.

\(^{11}\) http://en.wikipedia.org/wiki/Cyclic_redundancy_check
Example XML – fullXml

```xml
<eqHx:egContainedObj objectID="eqHx.ShlTubeUnit.H101" version="A107B777" objectState="fullXml">
[The full XML here.]
</eqHx:egContainedObj>
```

Q.6.3 Registries and Data Services

The use of Object and its attributes supports the use of XML files distributed in many resources, files or databases and accessible via a large range of possible implementations of index, directory or registry. The semantics of the Object type and how it is used is described in detail [E]. This will ensure that XML generated by different processing software will be fully compatible. However cfiXML schema does not specify how registries or data services should be implemented.

Rationale for Registries and Data Services

The hierarchy of names used and associated information for any registry is highly likely to be different for different companies or industries. Also, it is likely that the most efficient way of providing this type of functionality will be via a high performance multi-user database operating on a central server. The combination of contextURL, version and objectID are sufficient to uniquely identify an Object resource and to implement the software to retrieve it.

Q.7 Units Support and Issues with XSLT and Direct XML Access

Q.7.1 Converting Engineering Values from XML

Physical Quantities (pq: namespace) [D.2], provides details on how engineering values with associated units of measure are stored in cfiXML. To display an engineering figure in an application user interface or on a report, the source and target symbol for the value is required. Within the XML the element containing the value can have a ‘symbol’ attribute making the source units of measure obvious. If no symbol attribute is present, the application must locate the ‘unitSet’ attribute of the containing object. The pq.xsd schema file, or a custom user schema file, will provide the definition of the unit set specified by the unitSet attribute. This gives the unit symbol for every physical quantity. The physical quantity of the value can be obtained from the definition of the underlying pq type used. This combined with the definition of the unit set in use is used to determine the unit symbol for the value. If no unitSet attribute is specified for the containing object, then the SI standard unit set is used. The target symbol is dictated by the application. Once the source and target symbol have been determined, the value can be converted from the source to the target units by applying standard unit conversion factors and, in the case of temperatures, offsets.

It is convenient to include in the application interface, a table of all the conversion factors for all symbols relative to the SI symbol for each physical quantity. Such a table can be used to quickly convert between any 2 unit symbols.
Q.7.2 Using XSLT or Directly Accessing XML Values

As cfiXML does not store all values in a single unit set, the units for a value have to be obtained, as given in the procedure above. When using XSLT or directly obtaining values from XML, the value must remain with its associated symbol. The XSLT, or any mechanism which directly accesses XML engineering values, must include special processing to obtain the unit symbol for the value. In XSLT this can be done by embedding a scripting language routine to do this job.

Q.8 Populating XML structures – Processing Rules

Q.8.1 Element Order

The implication of the extensive use of the sequence content model in cfiXML schemas is that application export interfaces must ensure that all elements are placed in the correct order for the XML to be valid. Creating elements in a sequence that has elements out of order will not pass XML parser validation against the schema. One deployment practice that has been found useful is to export a ‘blank frame’, populate the blank frame with data from the application, and then remove any tags that have null values.

Q.8.2 Creating Objects

[Not applicable for cfiXML LT]

Even though a cfiXML object is an element like any other XML element, it must also include a minimum set of attributes. An export application must recognise each element that is an object and ensure that at a minimum the objectID and contextURL attributes are specified with valid strings. See ‘Creating and managing cfiXML objects’ [Q.6] for details on the special requirements for objects.

Q.8.3 Parent Element Creation

When exporting a value, particularly when using an application interface that directly uses XPaths, all element nodes (ancestors) in the XPath before the element that contains the data, must be created in the XML. If any of these element nodes are objects, then they must be treated specially as given in the previous section. Also, if any filters or qualifying attributes are required in the XPath or to correctly specify meta-data associated with the value, these must be included in the XML.

Q.8.4 Supporting the alternativeOption Attribute

If multiple alternative option elements are to be used in the XML, the application must recognise such elements as alternatives when importing data and must include these attributes when exporting. In particular, if only a single element instance exists, then the alternativeOption attribute should not be used. If there is more than one instance of exactly the same element, with no other or the same attribute qualifiers, then all such elements must include the alternativeOption attribute [N.3].

Q.8.5 Maintaining Existing XML Data

When an application uses an XML resource that already exists or was created by another program, it is essential that the application does not corrupt or delete any data that existed before the application used it. Also, any data that the application does not import, export
or manage, as given in the application’s published specification, should not be changed in any way.

These processing rules ensure that a single XML resource can flow through any number of applications with the assurance that each application only uses and changes data applicable to their purpose.

**Q.8.6 Access to Individual Elements Where Multiple Instances Exist**

cfiXML elements and attribute values must always be accessible through the use of a standard W3C XPath which does not include ‘not’. This means that all multiple instance elements within XML created by an application, must be qualified with attributes or the use of alternativeOption so that they can be uniquely identified using an XPath without having to resort to using an index or the ‘not’ logical operator.

Example of invalid XML:

\[
\begin{align*}
E[@\text{Attribute}='A'] \\
E[@\text{Attribute}='B'] \\
E & \text{has no attribute qualifier.}
\end{align*}
\]

In this example, it is not possible to obtain the value of the third element E without using a ‘not’ filter to exclude the first 2 E elements. The solution, for example, is to assign Attribute='Unspecified’ for the third element.

**Q.9 Supporting Revisions, Versions and Change Management**

cfiXML LT does not support detailed revisions, versions and change management. cfiXML LT includes support for all data found in a typical procurement data sheet revision table. For example, this would consist of the document revision identifier, date, description and the person who did the modification, checking or issuing.

**Q.9.1 Versions and Transactions**

[Not applicable for cfiXML LT]

All objects in AEX cfiXML can contain a table of transaction information where each transaction describes the reason for the change in status or data of the object. A transaction element includes the person responsible for the change, the date and time of the change and the type of change, e.g. Created, Modified, Issued (Figure 17). The transaction element also includes a version element that uniquely identifies the specific version of the object associated with the transaction. Processing software must be able to recognize that when a date is not included in a transaction, the transaction has been left "open" to be closed and time/date stamped when committed by the user.
Figure 17 Transaction elements provided for all objects in cfiXML

A new version of the object is only created when the data of the object is changed in any way. Changing the status of the object by adding a Checked, Approved or Issued transaction, which does not change the engineering data in the object itself, does not require a new version to be created.
When supporting versions, an application must copy the current object and make it available in the new version object via the obj:previousVersion element. Depending on how objects are being managed, the previousVersion element may contain full XML or solely a reference to the old version object via a registry. In the latter case, the use of versionIndex provides a means for an application to label old version XML file names with humanly readable numbers, starting with a version index of ‘1’ and continuing in sequential order.

When a copy of an object is made and a transaction of type “Modified copy” is created, a completely new object is created. The new object is a copy of the original but will have a different objectID. The new object can maintain the audit trail to the source object using the “sourceObjectReference” element.

Q.9.2 Revisions
[Not applicable for cfiXML LT]
Each object, as well as each transaction element, can include a revision string. This is a user defined identifier for the revision of an object, such as a document. The revision element is an unformatted string to support any revision numbering system required. Multiple object versions can make up a single revision. For example, a document may be created and then committed several times, each of these changes creating a new version of the object and new transaction element, but all corresponding to a single revision. The following sections of cfiXML illustrate how a program may use the provisions in cfiXML to record different versions of the file as it is developed. How different software handles a revision cycle does not matter provided that a transaction is added for every time a new version of the object is created and that the version id in that transaction corresponds to the version in the new version object.

Example: The centrifugal pump data sheet object includes the attributes revision and version. The file is revision “a”. When first created the version was “4C643DCA”, which was recorded as an attribute of the data sheet object and as part of the transactions type “Created” and “Modified” and “Checked”. These changes were committed by the particular processing software. Following further modification of the file, the changes were again committed and so the current version of XML file, revision “a” is “6BFC333C”.

<centrifugalPumpDataSheet objectID="document/eqRotDoc.CentrifugalPumpDataSheet.test2" revision="a" version="6BFC333C" versionIndex="2" etc….

<obj:transaction>
  <obj:dateTime>2007-03-16T06:25:11.000-00:00</obj:dateTime>
  <obj:order>1</obj:order>
  <ctx:person objectID="ctx.Person.Makt2">
    <ctx:shortID>MAKT</ctx:shortID>
  </ctx:person>
  <obj:revision>a</obj:revision>
</obj:transaction>
<obj:transactionType>Created</obj:transactionType>
<obj:version>4C643DCA</obj:version>
<obj:versionIndex>1</obj:versionIndex>
</obj:transaction>

<obj:transaction>
<obj:dateTime>2007-03-16T06:28:59.000-00:00</obj:dateTime>
<obj:order>2</obj:order>
<ctx:person objectID="ctx.Person.Makt2">
<ctx:shortID>Makt2</ctx:shortID>
</ctx:person>
<obj:remark>new data added</obj:remark>
<obj:revision>a</obj:revision>
<obj:transactionType>Modified</obj:transactionType>
<obj:version>4C643DCA</obj:version>
<obj:versionIndex>1</obj:versionIndex>
</obj:transaction>

<obj:transaction>
<obj:dateTime>2007-03-16T06:32:53.000-00:00</obj:dateTime>
<obj:order>3</obj:order>
<ctx:person objectID="ctx.Person.Makt2">
<ctx:shortID>Makt2</ctx:shortID>
</ctx:person>
<obj:remark>ok</obj:remark>
<obj:revision>a</obj:revision>
<obj:transactionType>Checked</obj:transactionType>
<obj:version>4C643DCA</obj:version>
<obj:versionIndex>1</obj:versionIndex>
</obj:transaction>

<obj:transaction>
<obj:order>4</obj:order>
<ctx:person objectID="ctx.Person.Makt2">
<ctx:shortID>Makt2</ctx:shortID>
</ctx:person>
<obj:revision>a</obj:revision>
<obj:transactionType>Modified</obj:transactionType>
<obj:version>6BFC333C</obj:version>
<obj:versionIndex>2</obj:versionIndex>
</obj:transaction>

Q.9.3 Change Tracking
[Not applicable for cfiXML LT]
Every element in cfiXML includes the optional Boolean attribute “changed” through the base attribute group. A value of true indicates that the element has been changed from the previous version. The processing software is required to reset these attributes from
one version to the next when a value has not been changed, i.e. if the element data changed from version 1 to version 2, but not from version 2 to 3.

In addition to tracking changes from one version to the next, cfiXML also supports tracking the changes made from one revision to the next revision. Standard engineering practice normally requires tracking revision changes. However, for full audit trails and legal needs, tracking version changes is needed. Revision tracking is done using the revChanged attribute which is available all all data elements. Application software must clear the refChanged attribute when creating a new revision.

**Q.9.4 Full Change Tracking, Revision and Versions System**

[Not applicable for cfiXML LT]

As illustrated by the example cfiXML above, version and revision control is supported by cfiXML. A full change tracking system can be developed by using the “changed” and revChanged Boolean-type attributes on each element and combining this information with the version and revision data. When changes are committed, the processing software can compare the content of the existing version of the cfiXML file with the “modified” content. A new version of the file can be saved by merging the content of the existing file with the changed file. This ensures that if the cfiXML file contains data from other sources than the particular program which is being run, no data are lost. The “existing” version then becomes the previousVersion, for example:

```xml
<obj:previousVersion xsi:type="CentrifugalPumpDataSheet" objectID="document/eqRotDoc.CentrifugalPumpDataSheet.test2" objectState="fullXMLReadCopy" version="4C643DCA">
```

The previous version therefore remains available if changes between versions need to be checked.

To summarize, to support full change tracking and a revision and versions system, data created/updated in an application needs to be compared and merged with the original XML data imported from another application. Alternatively, every change that the application makes must be marked as changed.

**Q.10 Providing Pick-lists and Allowing for User Values**

cfiXML enumeration values can be provided to application users via ComboBoxes or other controls to aid fast specification of accurate data. In more sophisticated applications, mapping tables between the cfiXML standard strings for enumeration values and familiar or local strings that are more easily recognised by users can be included.

[Not applicable for cfiXML LT]

cfiXML includes the otherValue attribute on most enumeration elements. Applications can include a mechanism to allow the user to specify any value for the enumeration element and then create valid XML by setting the enumeration value to “Other” and adding the user specified string as the value of the otherValue attribute. Similarly, such
an application needs to recognise and display or print any otherValue strings that exist in
XML being imported.

**Q.11 Optional Software Implementation Features**

*Not applicable for cfiXML LT*

**Q.11.1 User Access Control**
Access control is not supported by cfiXML.

*Rationale for Access Control*
Access control can be implemented in many ways at many different levels of granularity. However, using an ACL (access control list) which includes XPath expressions to specify chunks of the schema, right down to element level if required, access control can be implemented easily without any additional flags or data contained in the schema.

**Q.11.2 Read and Write Status**
The objectState attribute on all objects contains only 2 states that contain all the XML: “fullXML” and “fullXMLReadCopy”. All other states are references (or the deltaXML state which is likely to be withdrawn). The reference states do not provide the data of the object and are not accessible to an application.

Of the 2 objectStates which include all the data, one is ‘read and write’ and the other is ‘read only’. An application that supports access control needs to ensure that all data obtained from a “fullXMLReadCopy” object displayed in the application cannot be changed by the user. The application must not change this object in any way.

In a larger system that includes management of objects in a database or other resource manager, the “fullXMLReadCopy” will be provided by the system when an application requests a copy of a referenced object for read-only use. If an application requires a writeable copy, then the resource manger will issue the object with status “fullXML” and ensure that the object is “checked-out” or otherwise flagged as being in use.

**Q.11.3 Mappings to Other Schemas**
No mapping information to other standards is included in capital facilities industry XML, either at the XSD level or even the capability to include such information in XML.

*Rationale on Mappings to Other Schemas*
Multiple methods for mapping XML data to other Schemas or even other formats exist. The mapping information is more appropriately held in formats associated with the processing software, e.g. XSLT or POSC-CEASAR/EPISTLE/ProcessML implementation maps.

**Q.11.4 Default Values and Ranges**
Default and range information has not been included in the schema or even slots for the inclusion of such data in the XML.
Rationale for Default Values and Ranges

Any object can be used for storing default values and upper or lower bounds. Since such data will most often be corporate standard data it has been excluded from the schema. Also there are multiple implementation mechanisms for default and range values. One example of how such data can be implemented would be to maintain in a corporate ‘template’ resource, 3 copies of a pump XML object. Default values would be contained in the first pump object, which would be copied to an engineer’s environment on creating a new pump. The second and third pump XML objects would contain the lower and upper absolute range values and would be used by validation software in user interfaces.
R. KEY FOR FIGURES

The following is the key for figures showing relationships among cfiXML schema components.

- **A** indicating a diamond shape:
  - Indicates inheritance, or extends relationship; B inherits characteristics from A, or B extends A.

- **A** indicating a double arrow:
  - Indicates uses or dependency relationship; B uses A.

- **A** indicating a single arrow:
  - Indicates contains relationship; A contains B.

- **A** indicating a single arrow with a diamond shape:
  - Indicates may contain or reference relationship; A may contain B, A may contain a reference to B.

In addition to above, figures outlining key schema components may also indicate which items are objects and which are non-objects. This key is applicable to Figures 11 and 12 and for all equipment items from Figure 18.

- **Non-object**
- **Object**
S. CENTRIFUGAL PUMPS

S.1 Key Schema files
Collection-container Schema: ..schema/document/eqRotDoc.xsd
Subject Schema: ..schema/eqRotPmp.xsd
..schema/eqRotBase.xsd

The CentrifugalPumpDataSheet combines core and subject-specific engineering objects to describe context and detailed pump specifications as required by common standard data sheets used in industry for centrifugal pump procurement. As Figure 18 shows, the data sheet includes access to site and material stream information as well as the centrifugal pump.

Also available are CentrifugalPumpEquipmentList and CentrifugalPumpOrder ‘documents’ for describing details of pumps with different specifications (i.e., represented by using a number of data sheets) and an order list type document.

Figure 18 Outline of key schema components of the Centrifugal Pump Data Sheet
(See [R] for key)

S.2 Modeling of centrifugal pumps
Rotating equipment are complex assemblies of a driver, fluid mover, connecting shaft and various equipment parts including bearings and lube oil systems, seals and...
The schema was largely set up as a generalized model for any item of rotating equipment. So, the eqRot:CentrifugalPump object is extended from eqRot:Pump, which is extended from eqRot:RotatingEquipmentItem and eqRot:BaseRotatingEquipmentItem. The data types common to rotating equipment items are all available to the pump (e.g. shaft; bearing assembly, coupling etc.).

In addition, substitution groups [N.1] form a key part of the model. These provide links to mechanical and other seal data in eqRotSeal.xsd via the eqRot:seal element; attached drivers (e.g. electric motor in eqElecMtr.xsd or turbines in eqRotBase.xsd), and via stream connections, material stream data in uo.xsd. (See N.6 for details on material streams.)

**S.3 Description of some key data types**

- **CentrifugalPump**: A CentrifugalPump is a Pump whose function it is to raise the pressure of a liquid using kinetic energy.

- **Pump**: A Pump is a RotatingEquipmentItem whose function it is to raise the pressure of a liquid.

- **RotatingEquipmentItem** and **BaseRotatingEquipmentItem**: A RotatingEquipmentItem is extended from BaseRotatingEquipmentItem, an EquipmentItem, which is an assembly of machinery equipment items that impart rotational mechanical energy from a driver to the driven equipment item to accomplish work, typically in the transport of fluid materials.

- **Baseplate**: A Baseplate is an EquipmentItem upon which the driver and the driven equipment items for a RotatingEquipmentItem are mounted.

- **Gear**: A Gear is an EquipmentItem that changes the rotation speed between the driver and the driven equipment for a RotatingEquipmentItem.

- **Seal**: A Seal is an EquipmentItem that prevents internal fluid leakage to the environment for a RotatingEquipmentItem.

- **Shaft**: A Shaft is an EquipmentItem that transmits rotational mechanical energy between the driver and the driven equipment for a RotatingEquipmentItem.

- **Casing**: A Casing is a component part of a RotatingEquipmentItem that contains the driven equipment rotating element, e.g., impeller and the process fluid which is being raised in pressure.

- **Impeller**: An Impeller is a BulkItem and is the mechanical component that imparts kinetic energy to the fluid in a RotatingEquipmentItem.

**S.4 Example XML files**

The schema for centrifugal pumps currently covers all data items required for the Hydraulic Institute data dictionary for Bid Quote and request for quote (RFQ) work processes and for the API610 10th and 11th standard data sheet editions. The following lists example XML files which include real user data for both full and LT versions of schema. The key difference between these examples is the LT version does not include attributes such as objectIDs and any defining units of measurement [B].

AEX cfXML Schema Reference Guide 127
1. **examples/CPumpQuote_R_full.xml and CPumpQuote_R_LT.xml** include 171 ‘Required’ data items to describe the minimum technical information needed by the pump purchaser to evaluate the pump performance and make a qualified purchasing decision (i.e. Bid Quote data).

2. **examples/CPumpQuote_RDS_full.xml and CPumpQuote_RDS廖.xml** include 317 ‘Required’, ‘Desired’ and ‘Supplementary’ (RDS) data items to describe typical technical information used by the pump purchaser to evaluate the pump performance and make a purchase decision (i.e. Bid Quote data).

3. **examples/CPumpRFQ_R_full.xml and CPumpRFQ_R_LT.xml** - include 59 ‘Required’ data items to describe fundamental operating conditions, properties and characteristics of pumped fluid and main pump components for the candidate pump, and the minimum technical information required by the pump supplier to provide a pump selection and technical scope-of-supply (i.e. Bid RFQ data).

**Other example cfiXML Full test files include:**

*examples/CPumpProcurementDemo_DSandSizing.xml*

This contains data from initial stages of pump procurement. The data were exported from a data sheet into a sizing program and then the resulting data exported from the sizing program.

*examples/CentrifugalPumpTest2.xml*

The XML file includes data items required to fill in an industry standard data sheet (i.e. API610 8th edition). The values of the data are not from a real example of a procurement cycle, but are provided purely as a means to demonstrate the layout and structure of the XML.
S.5 Pump Tutorial – Using the cfiXML Schemas

In this section, we illustrate how to use an existing equipment item, a centrifugal pump, to navigate through the schemas, and extended them if needed.

Consider a centrifugal pump, which has the following characteristics:

1. equipment tag: P-101
2. dry weight: 435 lbs
3. normal capacity: 35 gallons/minute
4. manufacturer: XYZ Flow Equipment Company
5. model: A-1
6. type: OH3
7. operating environment: outdoor

The pump tutorial shows:

- how to use existing schemas as is
- how to use built-in schema extensions for data not in the schema

The tutorial is carried out in stages:

Part 1 - Using Existing schemas

- Using the built-in schema for values we know and are already in the schema
- Using Units of Measurement
- Defining user extensions to enumeration values without extending the schema definitions

Part 2 – Extending/customizing the schemas

S.5.1 Pump Tutorial Part 1 – Using the existing schemas

From a basic understanding of XML, we know that we can assign each part of the data to an element enclosed in tags and even without knowing anything about the schema, we would expect the XML data file for our pump to look something like the following:

```
<centrifugalPump>
  <tag>P-101</tag>
  <dryWeight>435.0</dryWeight>
  <normalCapacity>35</normalCapacity>
  <manufacturerCompany>XYZ Flow Equip Company</manufacturerCompany>
  <model>A-1</model>
  <type>OH3</type>
  <operatingEnvironment>outdoor</operatingEnvironment>
</centrifugalPump>
```
How do we describe this pump using the cfiXML core schema framework? First we need to look for schemas that already exist. As illustrated in Figure 18, the centrifugal pump datasheet contains the equipment item, ‘centrifugalPump.’ It also contains other information such as material properties and site conditions.

The schema structure for the centrifugal pump is defined as the bottom layer extension of a multiple layer inheritance hierarchy object data model, summarized as follows:

```
objc:BaseObject
  |objc:Object
  |  |eq:FacilityItem
  |  |  |eq:EquipmentItem
  |  |  |  |eqRot:BaseRotatingEquipmentItem
  |  |  |  |  |eqRot:RotatingEquipmentItem
  |  |  |  |  |  |eqRot:Pump
  |  |  |  |  |  |  |eqRot:CentrifugalPump
```

The structure of the object inheritance hierarchy schema is simultaneously powerful and confusing, at least until you gain some familiarity in working with it.

It is powerful, because common attributes of objects can be reused, so there is no need to repeatedly specify the same things when creating schema extensions to new equipment types. Similarly any data item defined in an ancestor or base object within the hierarchy is available in the more derived objects. For example, when developing an interface for a centrifugal pump, many data items that are relevant to all pumps will already be available in the eqRot:Pump object.

Looking at the native XML schema definition files is difficult, but it is made much easier by using an XML development environment with a tree view. The general nature of the content of this inheritance hierarchy can be summarized as follows:

1) objc:BaseObject holds common data about all objects, e.g., the unique ObjectID, the name and description
2) obj:Object holds common change and revision data about objects, e.g., the revision number, the revision date, etc.
3) eq:FacilityItem holds common data about any facility item, e.g., id information, weight or cost
4) eq:EquipmentItem holds common data about any engineered equipment item, e.g., design or documentation and certification requirements
5) eqRot:BaseRotatingEquipmentItem holds common data about any item of rotating equipment, e.g., the various pieces and parts of rotating equipment like bearings, baseplates, etc.
6) eqRot:RotatingEquipmentItem holds common data on seals and impellers for rotating equipment
7) eqRot:Pump holds common data about pumps.
8) eqRot:CentrifugalPump holds data specific to centrifugal pumps, e.g., information about vertically suspended pumps, applicable centrifugal pump standard, etc.

The way XML works is that each of these layers in an inheritance hierarchy shows up in the schema definition as a separate element sequence.

The ordering of elements in an XML instance file starts with the BaseObject data, then follows with ObjectData then FacilityItem data, etc. to CentrifugalPump data. Also, in looking at an XML instance file, the use of the namespace prefix is needed to say which namespace the element is located in.

The practical impact of this inheritance structure is that, when you are doing mappings to your software application, and you don’t immediately find the information element you’re looking for, you need to keep looking throughout the inheritance hierarchy (start either at the top or bottom of the hierarchy) until you find what you need.

After working with the schemas for a while, you become familiar with which level to start looking first. In a graphical XML editor, this task is relatively easy, because you can start with eqRot:CentrifugalPump and all levels of the hierarchy are shown at once, in the correct order and with the namespace prefixes.

Let’s consider our pump data items listed above (without regard to the actual schema) to see where we might begin looking:

```xml
<centrifugalPump>
  <tag>P-101</tag>
  <dryWeight>435.0</dryWeight>
  <normalCapacity>35</normalCapacity>
  <manufacturerCompany>XYZ Flow Equip Company</manufacturerCompany>
  <model>A-1</model>
  <type>OH3</type>
  <operatingEnvironment>Outdoors</operatingEnvironment>
</centrifugalPump>
```

**Steps in discovering the schema and instance file structure**

The following is a series of steps that you can take to discover the schema and instance file structure.

**Step 1 – Start with the container document**

Find the eqRotDoc:CentrifugalPumpDataSheet object in the `..//schema/document/eqRotDoc.xsd`. The datasheet contains siteFacility information, material information, and centrifugalPump information (Figure 18). Looking at the
schema structure against our current set of data, you would expect an instance file that looks something like the following:

```xml
<centrifugalPump>
  <siteFacility>
    <operatingEnvironment>Outdoors</operatingEnvironment>
  </siteFacility>
  <tag>P-101</tag>
  <dryWeight>435.0</dryWeight>
  <normalCapacity>35</normalCapacity>
  <manufacturerCompany>XYZ Flow Equip Company</manufacturerCompany>
  <model>A-1</model>
  <type>OH3</type>
</centrifugalPump>
```

**Step 2 – Map data items to the pump object structure and reorder**

Next, look at the centrifugalPump object structure in ../schema/eqRotCpmp.xsd. Now, the centrifugalPump object inherits from Pump which inherits from rotatingEquipmentItem object, which inherits from equipmentItem, etc. This is the initially confusing object hierarchy previously discussed. Looking for data items above, the following is found:

“operatingEnvironment” is associated with the “site” namespace (site.xsd). A SiteFacility has one or more SiteSubAreas.

“tag” is underneath an element called “id” and is located in the element sequence associated with the FacilityItem object in the ‘eq’ namespace.

“dryWeight” is called “weight” with an attribute of “weightType” which is an enumeration variable, and one of the weightTypes is “Dry” and is located in the element sequence associated with the FacilityItem object in the ‘eq’ namespace.

“normalCapacity” is the volumetric flow of the inlet (by default) stream. actualVolumetricFlow is available under stream in the uo namespace. This can be reached via the streamConnection on the centrifugalPump element. Qualifying attributes have to be specified to indicate that this is an “Inlet” flow, that the stream is a MaterialStream (not energy or heat) and that the flow type required is “Normal”. The usageContext is set to “Operating performance” on the streamConnection to indicate the context or perspective of the data required.

“manufacturerCompany” is underneath an element called id” and is located in the element sequence associated with the FacilityItem object in the ‘eq’ namespace.

“model” is underneath an element called id” and is located in the element sequence associated with the FacilityItem object in the ‘eq’ namespace.
“type” is called “pumpTypeClassification” and is located in the element applicableStandard associated with Pump.

Reordering these to be in proper sequence order (and this is an essential requirement), we should expect our XML file to look something like the following:

```xml
<centrifugalPumpDataSheet>
  <siteFacility>
    <siteSubArea>
      <characteristic>
        <operatingEnvironment>Outdoors</operatingEnvironment>
      </characteristic>
    </siteSubArea>
  </siteFacility>
  <centrifugalPump>
    <id>
      <manufacturerCompany>
        <name>XYZ Flow Equip Company</name>
      </manufacturerCompany>
      <model>A-1</model>
      <tag>P-101</tag>
    </id>
    <weight weightType="Dry">435.0</weight>
    <applicableStandard>
      <pumpTypeClassification>OH3</pumpTypeClassification>
    </applicableStandard>
    <streamConnection usageContext="Operating performance">
      <normalFlowDirection>Inlet</normalFlowDirection>
      <materialStream>
        <actualVolumetricFlow>35</actualVolumetricFlow>
        <flowPropertyType>Normal</flowPropertyType>
      </materialStream>
    </streamConnection>
  </centrifugalPump>
</centrifugalPumpDataSheet>
```

At this point we have found all the relevant elements in the schema, and we have a well-formed XML file specified in the correct order. But, we’re not done in creating a cfiXML file, for several reasons:
(a) We have not yet set up the root element, which is an object, declaring the necessary imported namespaces and included files.
(b) We have not defined the required attribute “objectID” (for cfiXML Full), and other qualifiers for the objects in the XML.
(c) We have not yet put the correct namespace prefixes on the element names.
(d) We have not yet handled the units of measurement for the numerical physical quantities.

**Step 3 – Set up root object, objectIDs, namespaces and units of measurement**

The easiest way to handle the root object declaration is to look at the top lines of an existing example file in the ‘examples’ folder. The following root object declaration came from the CentrifugalPumpTest2.xml example file. Then you add the required objectID attributes (for cfiXML Full), namespace prefixes to element names, and units of measurement, all of which are shown below in bolded blue.

```xml
<centrifugalPumpDataSheet
objectID="document/eqRotDoc.CentrifugalPumpDataSheet.MRA-MPCG-00001-r000"
document\eqRotDoc.xsd" contextURL="http://www.cplanndata.com/defaultContext"
language="en" xmlContent="cfiXML" objectState="fullXML" currencyCode="USD"
contentType="text/xml" unitSet="USEngineering"
xmlns="http://www.cfixml.org/document/eqRotDoc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlobj="http://www.cfixml.org/objb" xmlobj="http://www.cfixml.org/pq"
xmlox="http://www.cfixml.org/ctx" xmllox="http://www.cfixml.org/dx"
xmlobjects="http://www.cfixml.org/eq" xmlns=site="http://www.cfixml.org/site"
xmlobjects="eqRot="http://www.cfixml.org/eqRot"
schemaRelease="02.00.0000">
  <site:siteFacility
    objectID="site.SiteFacility.Brownfield" language="en" xmlContent="cfiXML"
    objectState="fullXML" unitSet="USEngineering" contentType="text/xml"
    currencyCode="USD">
    <site:siteSubArea
      objectID="site.SiteSubArea.CentrifugalPumpP10"
      language="en" xmlContent="cfiXML" objectState="fullXML"
      unitSet="USEngineering" contentType="text/xml" currencyCode="USD">
      <site:characteristic>
        <site:operatingEnvironment>Outdoors</site:operatingEnvironment>
      </site:characteristic>
    </site:siteSubArea>
  </site:siteFacility>
  <eqRot:centrifugalPump
    objectID="eqRot.CentrifugalPump.P10"
    language="en"
    xmlContent="cfiXML" objectState="fullXML" unitSet="USEngineering"
    contentType="text/xml" currencyCode="USD">
    <eq:id>
```
The root element of the document, centrifugalPumpDataSheet, includes other attributes that first define the required namespaces, the schema location and adds the objectID attribute. One thing to notice is that the centrifugal pump data sheet document contains elements from several namespaces (obj, site, ctx, eq, and eqRot) because:

1) Data Sheet is an arbitrary container document for other objects – some from site description data and some from rotating equipment, and

2) Even for centrifugal pump, the inheritance hierarchy described above spans three namespaces – obj, eq and eqRot. So, the elements contained within the data sheet document that come from namespaces other than the containing datasheet document are then always prefixed with their namespace prefix
The default units, if not specified at the object level or at the element level, are SI. In this example, note that the “unitSet” was set for the data sheet object to be USEngineering. What this means is that no unit symbols are required inside the containing object as long as the values conform to the default US Engineering units. To see what the definition of the US Engineering unit set is, in terms of physical quantities, consult the pq.xsd schema definition file. In these definitions, the default unit symbols are defined for each physical quantity. If the unit symbol is defaulted, there is no need to include the unitSymbol attribute. In the case of uo:actualVolumetricFlow the default is set to ft³/hr and the desired unit is US gallons per minute. This is accommodated by adding the unitSymbol=”US_gal/min” to the actualVolumetricFlow element.

**S.5.2 Pump Tutorial Part 2 – Extending/Customizing the schemas**

What if we wanted to add an “inventory code” to further identify the pump? We have two options:

1) add the required element directly to the XML instance file using the ‘customID’ tag at the end of the “id” sequence under FacilityItem.
2) create a new XML schema file, extending the “id” element with the required element.

Finally, what if the ISO Classification team adds a pump designated as OH5, which we will assume is not in the centrifugal pump ISO Classification enumeration list, for this exercise (though in fact it is).

We will examine each of these situations in this part of the pump tutorial.

**Part 2a – Adding custom data directly through custom elements**

The simplest way to provide extra data elements is to use the extensibility built into the schema using the ‘custom’ element found at the end of the appropriate sequence in the schema. In this example, we will add the custom element “inventoryCode” as an extension of the “FacilityItem/id” content sequence.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Pump Tutorial - Part 2a - extend existing schema in instance files by using built-in custom elements -->
<centrifugalPumpDataSheet objectID="eqRotDoc.DataSheetCentrifPump.Doc1012"
<!same as part 1 -->
<eqRot:centrifugalPump objectID="eqRot.CentrifugalPump.P-101">
<eq:id>
<ctx:manufacturerCompany
objectID="ctx.Organization.XYZFlowEquipmentCompany">
<obj:name>XYZ Flow Equipment Company</obj:name>
</ctx:manufacturerCompany>
<eq:model>A-1</eq:model>
<eq:tag>P-101</eq:tag>
<eq:customID>
<inventoryCode>P-XYZFlow00A1</inventoryCode>
</eq:customID>
</eq:id>
</eqRot:centrifugalPump>
</centrifugalPumpDataSheet>
```
In this example, all we did was add the new element directly under the ‘eq:customID’ element that was already built in to the base schema for FacilityItem.

This approach is easy. Just put data in the instance file. No schema definition changes are needed.

The shortcoming of this method is that the data are not validated. The additional custom element is a string value and can literally be anything you want. As long as the XML is “well formed” with data enclosed within valid XML tags no further validation is made by the XML parser. This is effective provided that when you build your software application to use the data, you take on the additional burden of ensuring the data are valid.

In the next step, we show you how you can add data elements of a particular data type and have the XML parser validate against this data type. This has the effect of increasing work in providing custom user supplied schema files, but saving work in your application code, because XML parser validation is now available.

**Part 2b – Adding elements by defining a custom extension schema**

In this example, we show how to create a user extension using a separate custom schema file. This is very powerful for defining extensions that are not already in the schema. Here, the relatively simple task of adding an additional element to the site schema is described. This “inventoryCode” is an extension under FacilityItem/id and can be fully validated as a String data item by the XML parser. This is advantageous to applications, because data validation is done directly by the XML parser, not the application code. Also, the change tracking features are available through the additional attributes that the ext:String provides, such as “changed” or “comment” etc.

The custom user schema file, custom.xsd is as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Pump Tutorial - Part 2b - custom schema extension file that adds additional custom characteristic to FacilityItem -->
<xsd:schema targetNamespace="http://www.cfixml.org/custom/cfixml-org/custom"
 xmlns="http://www.cfixml.org/custom/cfixml-org/custom"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmllns:ext="http://www.cfixml.org/ext"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.00.0000">
  <xsd:import namespace="http://www.cfixml.org/ext"
              schemaLocation="../../schema/ext.xsd"/>
  <xsd:element name="inventoryCode" type="ext:String"/>
</xsd:schema>
```

We first declare a custom namespace according to cfiXML usage convention as “http://www.cfixml.org/custom/cfixml-org/custom.” Second we use XML’s import feature to import the “ext” namespace, which defines the “ext:String” data type. Finally, we define an element for ‘inventoryCode’ in this namespace which is has the data type of “ext:String.”
In the XML instance file, virtually everything is the same as in Part2a, except we declare the ‘custom’ namespace and substitute “custom:inventoryCode” in place of the unvalidated ‘inventoryCode’ element.

```xml
<?xml version="1.0" encoding="UTF-8"?><!-- Pump Tutorial - Part 2b - extend existing schema using a user-supplied custom schema extension file -->
<centrifugalPumpDataSheet objectID="dxDsCpmp.DataSheetCentrifPump.Doc1012"
xmlns="http://www.cfixml.org/document/eqRotDoc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:obj="http://www.cfixml.org/obj"
xmlns:ctx="http://www.cfixml.org/ctx"
xmlns:custom="http://www.cfixml.org/custom/cfixml-org/custom"
xmlns:site="http://www.cfixml.org/site"
xmlns:eq="http://www.cfixml.org/eq"
xmlns:eqRot="http://www.cfixml.org/eqRot"
http://www.cfixml.org/custom/cfixml-org/custom ...
../schema/custom/cfixml-org/custom2b.xsd">
<!--same as part 1-->
<eqRot:centrifugalPump objectID="eqRot.CentrifugalPump.P-101">
  <eq:id>
    <ctx:manufacturerCompany objectID="ctx.Organization.XYZFlowEquipmentCompany">
      <obj:name>XYZ Flow Equipment Company</obj:name>
    </ctx:manufacturerCompany>
    <eq:model>A-1</eq:model>
    <eq:tag>P-101</eq:tag>
    <eq:customID>
      <custom:inventoryCode>P-XYZFlow00A1</custom:inventoryCode>
    </eq:customID>
  </eq:id>
</eqRot:centrifugalPump>
</centrifugalPumpDataSheet>
```

This solution, though not as easy as the method in Part 2a, is much more powerful, because it offers the advantage of full parser validation of the data and the ability to build on existing cfiXML schema types such as ext:String. The benefits of using ext: or pq: data types ensures that the attributes provided to support change tracking or units of measurement are all available to the custom elements. This maybe not so important a consideration for a one-element extension, but is very useful if a large number of elements or complex structure is needed.

**Part 2c – Adding extensions to enumeration variable values**

Each enumeration in the schema is specified as a combination of a simpleType and an element definition based on the simpleType. The simpleType contains the values for the enumeration strings. One of these choices is always “custom”, which can be used if there is a need to extend the enumeration lists.

*(Note: There is an additional, simple method by which an additional enumeration can be added, as described at the end of the pump tutorial.)*

For example, let us assume that the pump ISO Classification of OH5 is not in existing schema for this exercise. To add this choice to the enumeration list, we will show how to
extend the ‘custom’ schema extension from part 2b, where we added the new element ‘inventoryCode’. In this case we simply add a new element as a standard enumeration variable (restriction of the xsd:string data type). Our new ‘custom’ schema now becomes:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Pump Tutorial - Part 2c - custom schema extension file that extends an enumeration variable -->
<xsd:schema targetNamespace="http://www.cfixml.org/custom/cfixml-org/custom"
xmns="http://www.cfixml.org/custom/cfixml-org/custom" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.00.0000">
  <xsd:import namespace="http://www.cfixml.org/ext"
schemaLocation="../../../schema/ext.xsd"/>
  <xsd:element name="pumpTypeClassification">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="OH5"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="isMarine" type="ext:Boolean" nillable="true"/>
</xsd:schema>
```

Using this custom schema, the XML instance file changing the pumpTypeClassification from OH3 to OH5 is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Pump Tutorial - Step 2c - extending an enumeration variable using a user-supplied custom schema -->
<centrifugalPumpDataSheet objectID="eqRotDoc.DataSheetCentrifPump.Doc1012">
  <!-- same as in part 1 -->
  <eqRot:centrifugalPump objectID="eqRot.CentrifugalPump.P-101">
    <eqRot:applicableStandard>
      <eqRot:pumpTypeClassification>custom</eqRot:pumpTypeClassification>
      <eqRot:customApplicableStandard>
        <custom:pumpTypeClassification>OH5test</custom:pumpTypeClassification>
      </eqRot:customApplicableStandard>
    </eqRot:applicableStandard>
  </eqRot:centrifugalPump>
</centrifugalPumpDataSheet>
```

Note that the built-in schema element choice has the choice of custom, and then the next element comes from the ‘custom’ namespace defined in our user-supplied custom schema definition file.

**S.5.3 Extending enumeration elements using the “OtherValue” method**

In addition to the method of extending enumeration lists described in Part 2c above, there is a simple method which allows a user to add another enumeration to the provided list. Most enumeration lists include the final 3 enumeration values of “Other”, “Unspecified”
and “custom” in this order as a standard. If “Other” is chose, then the attribute “otherValue” can optionally be used to enter a string, if the data are available. This is illustrated by the following XML and XPath:

**Example XML** of “Any type” of a centrifugal pump seal, gland connection, tap type is required, i.e. an “Other” value from enumeration ETapType.

```xml
<eqRot:centrifugalPump>
  <eqRot:seal>
    <eqRot:mechanicalSeal objectID="eqRot.Seal.MechanicalSeal">
      <eqRot:glandConnection>
        <eqRot:isRequired>true</eqRot:isRequired>
        <eqRot:tapType otherValue="Any type">Other</eqRot:tapType>
      </eqRot:glandConnection>
    </eqRot:mechanicalSeal>
  </eqRot:sealType>
</eqRot:centrifugalPump>
```

**Example XPath** to be able to add a string for the otherValue attribute to describe the type of tap, which isn’t included in the enumeration ETapType:

```xml
eqRot:centrifugalPump/eqRot:seal[eqRot:sealType="MechanicalSeal"]/eqRot:sealItem/eqRot:glandConnection[eqRot:tapType="Other"]/eqRot:tapType/@otherValue
```
T. CENTRIFUGAL FANS

T.1 Key Schema files

Collection-container Schema: ..schema/document/eqRotDoc.xsd
Subject Schema: ..schema/eqRotFan.xsd
..schema/eqRotBase.xsd

The CentrifugalFanDataSheet combines core and subject-specific engineering objects to describe context and detailed fan specifications as required by common standard data sheets used in industry for centrifugal fan procurement. As Figure 19 shows, the data sheet includes access to site and material stream information as well as the centrifugal fan.

Also available are CentrifugalFanEquipmentList and CentrifugalFanOrder schemas for describing details of fans with different specifications (i.e., represented by using a number of data sheets) and an order list type document.

Figure 19 Outline of key schema components of the Centrifugal Fan Data Sheet
(See[R] for key)
T.2 Modeling of fans
When compared with centrifugal pumps, it can be seen that the key difference between
the schemas is that centrifugal fans are extended from eqRot:Fan (as opposed to
eqRot:Pump), which is extended from eqRot:RotatingEquipmentItem.

In addition, substitution groups [N.1] form a key part of the model. These provide links
to mechanical and other seal data in eqRotSeal.xsd via the eqRot:seal element; attached
drivers (e.g. electric motor in eqElecMtr.xsd or turbines in eqRotBase.xsd), and via
stream connections, material stream data in uo.xsd. (See N.6 for details on material
streams.)

T.3 Description of data types
The key data types in Centrifugal Fan are the same as for Centrifugal Pump. The only
difference is that Centrifugal Fan is defined as a RotatingEquipmentItem whose function
it is to raise the pressure of a gas using kinetic energy.

T.4 Example XML file
..examples/CentrifugalFanTest1.xml

The file provides XML illustrating data items required to fill in an industry standard data
sheet. The values of the data are not from a real example of a procurement cycle, but are
provided purely as a means to demonstrate the layout and structure of the XML.
U. SHELL AND TUBE HEAT EXCHANGERS

U.1 Key Schema files

Collection-container Schema: ..schema/document/ eqHxDoc.xsd
Subject Schema: ..schema/eqHxSt.xsd
..schema/eqHxBase.xsd

The ShellTubeExchangerDataSheet combines core and subject-specific engineering objects to describe context and detailed shell and tube heat exchanger specifications. As Figure 20 shows, the data sheet includes access to site and material stream information, detailed thermodynamic parameters and extended material properties data, as well as the equipment item.

Also available are ShellTubeExchangerEquipmentList and ShellTubeExchangerOrder schemas for describing details of exchangers with different specifications (i.e. represented by using a number of data sheets) and an order list type document.

Figure 20 Outline of key schema components of the Shell and Tube Heat Exchanger Data Sheet
(See [R] for key.)

U.2 Modeling of Shell and Tube Heat Exchangers

For heat exchangers, the schema was set up as a generalized model for any item of heat exchange equipment, e.g., the heat exchanger equipment and tube objects. These are
extended with detailed specializations that are appropriate only for shell and tube exchangers. In this way, the model is readily extensible and reusable for air coolers, plate and other heat exchanger equipment items.

So, the eqHx:ShellTubeExchangerUnit is extended from eqHx:HeatExchangerEquipment, which is in turn extended from eq:EquipmentItem. Each unit can comprise numerous assemblages. Specifications for the unit as a whole, for each assemblage and components are all covered.

The schema has been compiled to meet the requirements for design and rating, data sheets and other usage scenarios.

**U.3 Description of some key data types**

- HeatExchangerEquipment: A HeatExchangerEquipment is an EquipmentItem that describes basic heat exchanger function regardless of type.
- ShellTubeExchangerUnit: A ShellTubeExchangerUnit is a HeatExchangerEquipment item and consists of one or more ShellTubeExchangerAssemblies.
- ShellTubeExchanger: A ShellTubeExchangerConceptualSpecification describes process performance information used to design the shell and tube heat exchanger.
- ShellTubeExchangerAssembly: A ShellTubeExchangerAssembly is an EquipmentItem describing the physical arrangement of component parts of the shell and tube heat exchanger. It consists of shell, ends, tube bundle, etc.
- ExchangerNozzle: A HeatExchangerNozzle is a FabricatedNozzle through which the process fluids pass into and out of the heat exchanger.
- mtrl:PropertyData: A PropertyDataTable (from the ‘mtrl’ namespace) is an Object that contains the tables of fluid physical properties necessary to design the heat exchanger.

**U.4 Example XML file**

../examples/ShellTubeHeatExchangerTest5.xml

The example XML includes data sheet specifications with detailed heat release property curves. The values of the data are not from a real example, but are provided purely as a means to demonstrate the layout and structure of the XML.
V. AIR COOLED HEAT EXCHANGERS

V.1 Key Schema files

Collection-container Schema: ..schema/document/eqHxDoc.xsd
Subject Schema: ..schema/eqHxAc.xsd ..schema/eqHxBase.xsd

The AirCooledHeatExchangerDataSheet combines core and subject-specific engineering objects to describe context and detailed air cooled heat exchanger specifications. As Figure 21 shows, the data sheet includes access to site and material stream information, detailed thermodynamic parameters and extended material properties data, as well as the key data for the equipment item.

Also available are AirCooledHeatExchangerEquipmentList and AirCooledHeatExchangerOrder schemas for describing details of exchangers with different specifications (i.e. represented by using a number of data sheets) and an order list type document.

Figure 21 Outline of key schema components of the Air Cooled Heat Exchanger Data Sheet
(See [R] for key.)
V.2  Modeling of Air Cooled Exchangers
As described under shell and tube heat exchangers, some key data items for a generalized heat exchanger are extended to provide the air cooled heat exchanger specifications. For example, the eqHx:AirCooledHeatExchangerUnit is extended from eqHx:HeatExchangerEquipment, which is in turn extended from eq:EquipmentItem.

Each unit can comprise of numerous assemblages. Specifications for the unit as a whole, for each assemblage and components are all covered.

V.3  Description of some key data types
AirCooledHeatExchangerUnit  An AirCooledHeatExchangerUnit is a HeatExchangerEquipment item and consists of one or more AirCooledHeatExchangerAssemblies.

AirCooledHeatExchanger As an AirCooledHeatExchangerConceptualSpecification describes process performance information used to design the shell and tube heat exchanger.

AirCooledHeatExchangerAssembly An AirCooledHeatExchangerAssembly is an EquipmentItem describing the physical arrangement of component parts of the shell and tube heat exchanger. It consists of shell, ends, tube bundle, etc.

Bundle A Bundle is an EquipmentItem containing a collection of Tubes used to provide heat exchange surface area between two non-contacting fluids.

HeatingCoil A HeatingCoil is a coiled Tube used to provide heat exchange surface area between two non-contacting fluids.

Louvre A Louvre is a BulkItem that is used to adjust air flow across the heat exchange

V.4  Example XML file
..examples/ AirCooledHeatExchangerTest1.xml

The example XML includes data sheet specifications. The values of the data are not from a real example, but are provided purely as a means to demonstrate the layout and structure of the XML.
W. ELECTRIC MOTORS

W.1 Key Schema files

Collection-container Schema: ..schema/document/eqElecDoc.xsd
Subject Schema: ..schema/eqElecMtr.xsd
..schema/eqElecBase.xsd

The ElectricMotorDataSheet combines core and subject-specific engineering objects to describe context and detailed motor specifications. As Figure 22 shows, the data sheet includes access to site data as well as the electric motor.

Also available are ElectricMotorEquipmentList and ElectricMotorOrder schemas for describing details of motors with different specifications (i.e. represented by using a number of data sheets) and an order list type document.

W.2 Modeling of Electric Motors

The eqElec namespace defines the equipment types relevant in the electrical and power equipment subject domain. The electric motor has been developed in sufficient detail to describe the data found on industry standard datasheets. Additional items have been included in the eqElec namespace, which have standard equipment summary information available and can be used in lists of electrical equipment.

W.3 Description of some key data types

Cable A Cable is a BulkItem that is used to provide electrical power.
ElectricMotor An ElectricMotor is an EquipmentItem that typically provides the function of an “equipment driver” turning a shaft that is connected to an item of rotating equipment such as a centrifugal pump.
ElectricMotorEnclosure An ElectricMotorEnclosure is a Housing used to contain the motor to protect it from the surrounding environment.
ElectricalSupplyEquipment An ElectricalSupply is an Object that supplies electrical energy to the ElectricMotor.
MainSupplyConductor A MainSupplyConductor is a Cable that provides electrical power through the ElectricalSupply.
SpaceHeater A SpaceHeater is an EquipmentItem that provides heat to the TerminalBox if needed.
TerminalBox A TerminalBox is an EquipmentItem that connects the ElectricalSupply to the ElectricMotor.
tewacHeatExchanger A tewacHeatExchanger, or Totally Enclosed Water-Air Cooled Heat Exchanger is a BulkItem that provides water cooling.
W.4 Example XML file

..examples/EMotorQuote_full.xml and EMotorQuote_LT.xml – includes data required for a quote for the procurement of an electric motor in full and LT schema-based versions. The key difference between these examples is the LT version does not include attributes such as objectIDs and any defining units of measurement [B].

..examples/EMotor250HPandOverTest1.xml and /EMotorUpTo600VTest1.xml – include data items required to fill in industry standard data sheets (based on API 541). The values of the data are not from a real example of a procurement cycle, but are provided purely as a means to demonstrate the layout and structure of the XML. These are both in cfiXML Full.
X. BLOCK AND CONTROL VALVES

X.1 Key Schema files

Collection-container Schema: ..schema/document/eqPvfDoc.xsd
Subject Schema: ..schema/eqPvfValve.xsd
..schema/eqPvfBase.xsd

The eqPvfDoc.xsd schema includes valveCatalog and valveCatalogLibrary. These have been developed to represent industry standard block valve catalogues in cfiXML and provide a tested example of these forms of ‘document’.

The eqPvfDoc.xsd file also includes ControlValveDataSheet. This combines core and subject schema objects to describe context and detailed linear or rotary control valve specifications as required by common industry standard data sheets for procurement. Also available are ControlValveEquipmentList and ControlValveOrder schemas for describing details of control valves with different specifications (i.e. represented by using a number of data sheets) and an order list type document.

A similar set of schema ‘documents’ are available in eqPvfDoc.xsd for describing basic specifications for the procurement of pressure relief valves.

X.2 Modeling of Valves

The valve schema structure was created for the Fiatech GVCC\textsuperscript{12} project supporting the Process Industry Practices (PIP) standard reference catalogues for block valves and the AEX project, supporting control valve equipment data sheets.

Figure 23 outlines the key components of schema for a block valve catalogue and a control valve data sheet. The valve schema provides a relatively involved example of reusing complex types. As there are sets of elements common to both types of valves, complex types have been developed to combine these base sets. Each sequence of shared elements is then extended to provide the specifications for either the block valves or the control valves. This system ensures that data elements are consistent wherever used in the schema and the chance for error is reduced. The resulting sequences of elements may seem strangely ordered. However, by viewing the complexTypes that have been extended, the structure becomes clearer.

Overall, the blockValve schema comprises a simpler model than the controlValve model. Examination will highlight that some of the key additional sequences of elements found in controlValve schema are to describe associated monitoring and control systems.

\textsuperscript{12} http://www.fiatech.org/projects/idim/gvcc.htm
X.3 Description of some key data types

BlockValve  A BlockValve is a Valve that regulates flow in pipes, and is often used in services where the valve is completely open or closed, depending on the operational mode (normal, maintenance, shut down, etc.). Block Valves are also used to regulate flow in a partially open position where continuous
valve positioning is not required.

BlockValveBodyAssembly A BlockValveBodyAssembly is a ValveBodyAssembly that has been extended for characteristics specific to block valves.

BlockValveOperator A BlockValveOperator is a set of characteristics specific to block valves for operation of the valve.

BlockValveSeat A BlockValveSeat is a ValveSeat that has been extended for characteristics specific to block valves

BlockValveTrim A BlockValveTrim is a ValveTrim that has been extended for characteristics specific to block valves

Controller A Controller is an EquipmentItem that provides control functions.

ControlValve A ControlValve is a Valve that regulates flow in pipes in a continuous control mode where the valve position is adjusted continuously by a Valve Actuator in response to a controlled process variable, e.g., desired flow rate.

ControlValveBodyAssembly A ControlValveBodyAssembly is a ValveBodyAssembly that has been extended for characteristics specific to control valves.

ControlValveSeat A ControlValveSeat is a ValveSeat that has been extended for characteristics specific to control valves.

ControlValveTrim A ControlValveTrim is a ValveTrim that has been extended for characteristics specific to control valves.

Valve A Valve is an EquipmentItem that provides the function of regulating fluid flow in pipes. The following types of valves are modeled: Ball, Butterfly, Check, Diaphragm, Gate, Globe, Needle, Plug and others.

ValveController A ValveController (includes Positioner) is a Controller which is part of a ControlValve

ValveBodyAssembly A ValveBodyAssembly is a reusable set of elements that describe common valve body and bonnet characteristics shared between BlockValve and ControlValve

ValveSeat A ValveSeat is a reusable set of elements that describe common valve seat characteristics shared between BlockValve and ControlValve

ValveTrim A ValveTrim is a reusable set of elements that describe common valve trim characteristics shared between BlockValve and ControlValve. This includes parts which come into contact with the process fluid.

X.4 Example XML files

..examples/LinearControlValveTest1.xml
This file provides XML illustrating data items required to fill in an industry standard data sheet. The values of the data are not from a real example of a procurement cycle, but are provided purely as a means to demonstrate the layout and structure of the XML.

..examples/PReliefValve_BidQuoteTest1.xml
This file includes real example data of Bid Quote specifications for pressure relief valves.
Y. HVAC EQUIPMENT

The most recent development of the AEX cfiXML schema model has been for Heating, Ventilation and Air Conditioning Equipment. The eqHvac.xsd and eqHvacDoc.xsd namespaces have been added and include data for Air Handling Units and Chillers.

Y.1 Air Handling Unit Key Schema files

Collection-container Schema: ..schema/document/eqHvacDoc.xsd
Subject Schema: ..schema/eqHvacAhu.xsd  ..schema/eqHvacBase.xsd

The eqHvacDoc.xsd file includes an AirHandlingUnitDataSheet to combine core and subject-specific objects to describe context and detailed Air Handling Unit (AHU) specifications. As Figure 24 shows, the data sheet includes access to site and material stream information as well as the key data for the AHU.

Also available are AirHandlingUnitEquipmentList and AirHandlingUnitOrder elements for describing details of AHUs with different specifications (i.e. represented by using a number of data sheets) and an order list type document.

Figure 24 Outline of key schema components of the Air Handling Unit Data Sheet
(See [R] for key.)
Y.2  **Modeling of Air Handling Units**

The schema was developed to cover example data for the procurement of AHUs. The cfXML schema files already included key components required for the description of AHUs.

The hot and cold coil components of an AHU are finned tube heat exchangers.

Axial and centrifugal fan data are accessible via eqHvac:fan as part of the substitution group with the ‘head element’ eqRot:FanItem (see [N.1] on substitution groups). An outline of the data elements available for the description of centrifugal fans is given in Figure 19.

The new element eqHvac:Casing includes casing component information, including uFactor, material types and thickness. The casing is extended from eq: FabricatedItem and therefore all elements associated with this are available.

Specifications associated with fluids flowing through the coil components (i.e. finned tube heat exchanger tubes) and the air side flows over the coils are covered by streamConnection elements within the schema. Similarly, data on air flows through the associated fans are also covered by the streamConnection element. (See [N.6] for information on material streams.)

Y.3  **Description of some key AHU data types**

- eqHx:HeatExchangerEquipment: A HeatExchangerEquipment is an EquipmentItem that describes basic heat exchanger function regardless of type.
- eqHx:FinnedTubeHeatExchangerUnit: A FinnedTubeHeatExchangerUnit is a HeatExchangerEquipment item and consists of one or more assemblies.
- eqHx:finnedTubeExchangerAssembly: A finnedTubeExchangerAssembly is an EquipmentItem with component parts of the heat exchanger and in the AHU domain, are the ‘Coil’ elements. It includes a bundle, tubes, and air side specifications.
- eqRot:AxialFan: An AxialFan is a type eqRot:Fan, which is extended from eqRot:RotatingEquipmentItem. (See Centrifugal Fan for details)
- Casing: The eqHvac:Casing is extended from eq: FabricatedItem.

Y.4  **AHU Example XML file**

..examples/AirHandlingUnit-BidQuoteATest1.xml

The data in this xml file are a set of specifications provided by a vendor as part of a Bid Quote. Details of the data included are found in ..examples/AHUBidQuoteExampleXMLLog.txt.
Y.5 Vapor Compression Chiller Key Schema files

Collection-container Schema:  ..schema/document/eqHvacDoc.xsd
Subject Schema:  ..schema/eqHvacChiller.xsd
..schema/eqHvacBase.xsd

As with the AHU, eqHvacDoc includes a VaporCompressionChillerDataSheet, EquipmentList and –Order.

Figure 25 Outline of key schema components of the Vapor Compression Chiller Data Sheet
(See [R] for key.)

Y.6 Modeling of Chillers

In Figure 25, the condenser is shown as including attachedFan and and attachedPump elements. These provide access to substitution groups [N.1]. It is therefore possible to access details of a fan or a pump depending on whether the condenser is air-cooled or fluid-cooled.

The compressor element contains another substitution group allowing the user to select which type of compressor, if this is required. A couple of examples of compressors are shown. Schema cover for centrifugal and reciprocating compressors is the most developed.
The evaporativeCooler includes the substitution group with the ‘head element’ eqHx:evaporativeCoolerItem, which is extended from eqHx:HeatExchangerEquipment, the base data set for all heat exchangers. If the ‘evaporator’ in the chiller system is air-cooled, then the finnedTubeHeatExchangerUnit can be accessed. If fluid-cooled, then the shellTubeHeatExchangerUnit can be accessed.

One important point in the modeling of chillers involves the material streams associated with the system. There is the refrigerant fluid stream loop, the plant material stream which is being cooled as it passes through the evaporator and the air or fluid stream cooling the condenser. The present schema model allows specifications for all these material streams to be covered. (See N.6 for information on material streams.) As with all the other data elements associated with the chiller, the level of detail required needs to be determined by the parties involved with the data exchange or storage processes.

**Y.7 Description of some key Chiller data types**

See also the AHU set of data types.

- eqHvac:vaporCompressionChiller
  A vaporCompressionChiller is an EquipmentItem. It comprises various components which are also equipment items.

- eqHvac:evaporativeCoolerItem
  An evaporativeCoolerItem is an eqHx:HeatExchangerEquipment type. It is the head element for a substitution group which includes finned tube or shell and tube heat exchangers.

- eqHx:Condenser
  A Condenser is an eqHx:HeatExchangerEquipment item.

- eqHvac:compressorItem
  A compressorItem is an eqRot:CompressorItem type and forms the head element for a substitution group so the user can select which compressor is required.

**Y.8 Chiller Example XML file**

..examples/Chiller-BidQuoteExample.xml

This file includes data specifications which would be required as part of a Bid Quote. The values of the data are not from a real example, but are provided purely as a means to demonstrate the layout and structure of the XML.
INDEX

Air Cooled Exchangers
  Data types, 146
  Key Schema files, 145
  Modeling, 146
  XML example file, 146
Air Handling Unit (AHU)
  Data types, example, 153
  Key Schema files, 152
  Modeling, 153
  XML example file, 153
alternativeOption attribute, 101
  Example XML, 75
Annotation in schema
  appShortName and appLongName, 97
documentation, 97
Attributes and Qualifiers. See also by name
  Use, 91
BaseAttributeGroup (ext: ), 38
BaseEnumerationAttributeGroup (ext: ), 38, 88
BaseObject (objb: ), 93
BaseObject attributes (objb: ), 46, 54
Block and Control Valves
  Data types, 150
  Key Schema files, 149
  Modeling, 149
  XML example files, 151
Block Valves. See Block and Control Valves
BulkItem (eq: ), 59, 93
Centrifugal Fans
  Data types, 142
  Key Schema files, 141
  Modeling, 142
  XML example file, 142
Centrifugal Pumps
  Data types, 127
  Key Schema files, 126
  Modeling of, 126
  XML example files, 127
cfiXML Full
  Applications
    Data Requirements, 15
    Development Environments and Tools, 15
    How to Build, 15
    Setting up interface - a check list, 15
    Choosing Between LT and Full cfiXML, 5
cfiXML Light (LT)
  Applications
    Data Requirements, 12
Development Environments and Tools, 12
How to build, 12
Setting up interface - a check list, 12
Choosing Between LT and Full cfiXML, 5
Features, 30
Introduction to, 5
Nillable not supported, 33
Objects, identifiers, referencing, 6, 30
Point-in-time exchange, 5
Revision support, 6, 32
Schema simplification, 33
Unit support, 6, 33
Change Management, 54, 121
Collection-container Schemas, 10, 63
ComplexTypes. See Types and Elements
ConditionType element and attribute (etl: ), 43, 91
Content Models
All, no use of, 90
Choice, no use of, 90
Sequence and Ordering Conventions, 90
Context. See ctx: namespace
Control Valves. See Block and Control Valves
Core Data Type Schemas, 7, 38
Core Object Schemas, 9, 56
Creating an object, 93
ctx: namespace (Context), 9, 56

Data dictionary, 12, 15, 65, 66, 78, 113, 127
Default Values and Ranges, 123
Derivable Types, 100
Design Features
Flexibility and Extensibility, 4
Inheritance, 7, 61
Content beyond requirements, 62
Sequences, 62
Layered structure, 7
Reusability, 4
Units of Measurement, 5
Versions and Revisions System, 5
Directory structure, 2
Document. See dx: namespace
DOM or SAX parser, 12, 15, 16, 113
Downloads
Available files, 1
Web sites, 2
dx: namespace (Document), 9, 56
Example data XPaths, 68

Electric Motors
Data types, 147
Key Schema files, 147
Modeling, 147
XML example files, 148
Elements. See Also Types and Elements
Creating Base ‘Leaf’ Elements, 87
Optional by default, 87
Use for primary data, 86
Elements, naming
Abbreviations and Acronyms, 94
Company names, associations, 96
Prefix and Suffix Codes, 94
Units of measurement, 96
Use of Case, 94
Engineering Type Library. See etl: namespace
Enumeration list Order and format, 88
eq: namespace (Equipment), 10, 59
Example equipment data XPaths, 69
Example equipment properties XPaths, 70
Equipment Connections, 106
Equipment inspection tests, 74
Equipment items. See eq: namespace
EquipmentItem (eq: ), 59, 93
etl: namespace (Engineering Type Library), 8, 43, 91
ext: namespace (Extended Data Types), 7, 38, 87
Extended Data Types. See ext: namespace
Extending Schema
customXXX method, 20
Data Dictionaries, 19
Data Requirements, 19
Extending Standard cfijXML - a check list, 20
otherValue method, 20
Representing Equipment Items, 19
User-provided schema definitions, 20
FabricatedItem (eq: ), 59, 93
File folder structure, 2
File Headers, 97
File names (XSD) and namespaces, 7
fixedProperty (mtrl: )
Example XML, 76
Form Default Qualification, 67
Full Change Tracking, Revision and Versions System, 122

HVAC Equipment. See Air Handling Unit and Vapor Compression Chiller

Inheritance. See Design Features
Interoperability, 24

key and keyref, 103

Life Cycle Status, 107
lifeCycleStage vs usageContext, 107
Lists and Tables, complex. See PropertyTable
Lists and Tables, simple. See Order element and attribute
Mappings to Other Schemas, 123
Materials and associated properties. See mtrl: namespace
mtrl: namespace (Material), 9, 57
Example construction materials XPaths, 70

Namespaces
  targetNamespace definition, 67
  Use in Schema, 7
  W3C declaration, 7
  XML Schema Namespace definition, 67
Nillable, 39

objc: and obj: namespaces (Objects), 8, 45
Objects, 8, See also obj: and objc: namespaces
  Basics, 44
  Contained and Referenced Objects, 48
  contextURL - Object registration, 47
  Definition and purpose, 8
  objectID, 46
  objectState, 50
  Example XML, 73
  organizationID – ANSI registration, 48
Objects, Non-cfiXML or Customized
  Alien objects, 53
  External (non-cfiXML) Objects, 52
  Hybrid objects, 52
  Object Contents, 51
order attribute (ext: ), 105
  Example XML, 75
Order element, 104

Physical Quantity Data Types. See pq: namespace
pq: namespace, 40
pq: namespace (Physical Property DataTypes), 87
pq: namespace (Physical Quantity Data Types), 8
proj: namespace, 56
proj: namespace (Project), 9
Project. See proj: namespace
PropertyTable, 106
PropertyType element and attribute (etl: ), 43, 91

Read and Write Status, 123
Reference Relationships, 103
ReferenceProperty and ReferencePropertyID, 91
Revisions, 54, 120

Schema Content Summary, 2
Schema File Naming, 98
schemaLocation attribute, 67
Shell and Tube Heat Exchangers
  Data types, 144
  Key Schema files, 143
Modeling, 143
XML example file, 144
SimpleType enumerations. See Types and Elements
Site. See site: namespace
site: namespace (Site), 9, 58
Example data XPaths, 69
Streams. See uo: namespace
Streams, Material, 57, 103
Example XML, 73
Subject Schemas, 10, 61
Namespaces, 10
Substitution Groups, 99
Example XML, 71, 73

Transactions, 54
Tutorial, 129
Existing schema, 129
Extending and customizing schema
Custom elements, 136
Custom extension definition, 137
Enumeration custom option, 138
Enumeration OtherValue option, 139
Types and Elements, 86
ComplexTypes
Creating, 87
Creating an object, 93
Location in Schema, 89
Name Format, 89
SimpleType enumerations
Creating, 88
Order and format, 88

Unit Operation. See uo: namespace
uo: namespace (Unit Operation), 9, 59
Usage Scenarios
Demonstration, 26, 28
usageContext attribute (ext: ), 92
Enumeration usage, 72
Example XML, 73
Example XML and XPaths, 72
User Access Control, 123

Vapor Compression Chiller
Data types, examples, 155
Key Schema files, 154
Modeling, 154
XML example file, 155
Version Numbers, 97
Versions, 54
Versions and Transactions, 118

XML
Advantages, 1, 25
XML Schema Features
  Defaults, 109
  FormDefaults, 109
  Occurrences, attributes and elements, 108
XML serialization, 5, 12, 30, 31, 90, 100, 113