PSIG Pipeline Simulation Interest Group

XPSL Language Definition: *Working Recommendation Document*

PSIG 001/2006

Objective: To provide a common structured language for configuration (inputs) and results (outputs) for pipeline simulation applications.

Document Type: A Working Recommendation is a step beyond a Draft Recommendation. The next step is to incorporate comments on this document and prepare a Proposed Recommendation. The Proposed Recommendation is the last draft document. Following review and comments of the Proposed Recommendation, a Candidate Recommendation will be prepared and presented to the PSIG membership for acceptance.

Recommendation Schedule: It is the intention of the PSIG Data Standards Committee to present a Candidate Recommendation to the PSIG membership for approval at the 2006 PSIG Conference

Contents

XPSL Recommendation

1 Introduction

This document defines the XPSL language. XPSL is an acronym derived from "e**X**tensible **P**ipeline **S**imulation **L**anguage", a name which is very descriptive of the purpose of the language. XPSL has been designed to be a common language for describing the inputs and outputs from pipeline simulation applications.

XPSL is a language based on XML, the ubiquitous interface language of modern commerce and applications. XML (e**X**tensible **M**arkup **L**anguage) has become the de facto standard for data interfaces. XPSL is XML and therefore benefits from a wide range of tools that are available for viewing, editing, transforming, formatting, and documenting XML files.

1.1 XPSL Goal

The XPSL language provides a standard format for input and output data with the goal of accomplishing the following:

- To provide users (e.g. pipeline companies) with a structured language for maintaining a description of the physical pipeline assets. XPSL can be used as a "database" to maintain the detailed pipeline configuration data.
- To provide simulation applications with a standard configuration format for describing the physical pipeline configuration.
- To provide simulation applications with a standard format for specifying input variables such as fluid properties, control settings, and setpoints. XPSL provides a mechanism for defining both static and time varying inputs.
- To provide simulation applications with a standard format for outputting application results. XPSL provides a mechanism for defining both static and time varying outputs.
- To define data units consistently so that the data in an XPSL file may be interpreted precisely in the intended system(s) of units or converted to any other system of units.
- To provide precise definition of each XPSL data element so that any XPSL file has a precise meaning, regardless of the user or application.

1.2 XPSL Scope

XPSL is intended to provide a comprehensive language for all inputs and outputs of pipeline simulation applications. In its current (initial) version it primarily focuses on the following:

- Specification of the physical pipeline configuration.
- Specification of the variable pipeline data at a specific point in time. This set of data may include values for physical quantities such as pressures and flows, control modes for pipeline equipment, and limit settings (hi and low limits either for alarms or operational control). XPSL uses the term "Snapshot" to refer to a set of pipeline data at a specific point in time.
- Specification of the most commonly used pipeline fluids.
- Specification of Systems of Units.

Currently, XPSL provides no direct vocabulary to define line fill or compressor or pump curves. However, there are XPSL elements that may be used for those purposes. Also, currently, XPSL provides no mechanism for defining the contents of binary files that are often useful for certain types of inputs and outputs such as trends. The next release of XPSL will address these issues.

1.3 XPSL Schema and Valid XPSL Instances

This document will define the XPSL language in detail. An *XPSL instance* is an XML file that conforms to the XPSL language rules and syntax. XML provides a schema definition language (called XML Schema) that can be used to define the vocabulary of an XML application. XPSL is delivered with a schema referred to as the *XPSL Schema Definition* that is defined using the XML Schema Language. The *XPSL Schema* defines the element names as well as a great deal of the acceptable structure a valid XPSL instance (see below). Conforming to the *XPSL Schema* is a necessary and sufficient condition to be a valid XPSL instance.

A valid *XPSL Instance* is a collection of XML statements that conform to the *XPSL Schema.* Generally, an *XPSL Instance* is defined in one or more xml files that together define a valid *XPSL Instance*. For simplicity, we often refer to an XPSL Instance as an XPSL file. Users and applications create *XPSL Instances*. The PSIG Data Standards Committee maintains *XPSL Schema.*

Note that multiple related or unrelated XPSL files may be used or generated by a single user or application. For instance, the output may be one XPSL file and input may be a different XPSL file.

1.3.1 The XPSL Schema

An XPSL instance must satisfy all the constraints imposed by the XPSL schema. The XPSL schema defines the element names and attributes that may be used in an XPSL instance. The XPSL schema is defined using the W3C XML Schema language. The XPSL schema is defined in five files:

XPSLMain_1_4.xsd

XPSLElemTypes_1_4.xsd

XPSLUnits_1_4.xsd

XPSLValueTypes_1_4.xsd

XPSLGlobals_1_4.xsd

The main schema file is *XPSLMain_1_4.xsd.* The other files are included through this file.

All XPSL elements are defined within an **XPSL namespace** that is named:

http://www.xpsl.org

This name resolves to a specific internet location. The individual XPSL schema files may be retrieved from that location. Prefixes are used within xml files as a shorthand form to reference a specific namespace. The recommend prefix for the XPSL namespace is **xpsl**.

Any xml file that include XPSL elements should reference the XPSL schema and define the xpsl prefix. The following is an example of a minimal XPSL instance:

Figure 1 - Minimal XPSL File

```
<?xml version="1.0" encoding="UTF-8"?>
<xpsl:XPSL
   xmlns:xpsl=http://www.xpsl.org
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:schemaLocation="http://www.xpsl.org ../XPSLMain_1_4.xsd">
</xpsl:XPSL>
```

This first line declares that the file is an xml version 1.0 file. The second line declares an XPSL element. The XPSL element is the root element of an XPSL instance and will be discussed further below. The second line declares an xsi prefix that is part of the XML standard. The fourth line specifies where the http://www.xpsl.org schema is located. In this case, the location (../XPSLMain_1_4.xsd) indicates that the XPSLMain_1_4.xsd file that defines the schema is located in the current directory.

In this example, the XPSL instance consists of only a single root XPSL element.

1.3.2 XPSL Instances

Every XPSL instance consists of an XPSL element (defined in the next section) with zero or more child elements. Every xml file has a single root (uppermost) element. The root element may be an XPSL element as in Figure 1 or may be a child of another element as shown in the following example:

```
<PipelineData>
   <xpsl:XPSL> child elements </xpsl:XPSL>
</PipelineData>
```

In this example, the XPSL element is defined as a child of a PipelineData element.

There are no restriction on how many XPSL elements may occur in a single xml file. However, each XPSL element and its children are considered to be a separate XPSL instance. In the following example, two separate XPSL instances are defined as children of a <PipelineData> element.

```
<PipelineData>
   <xpsl:XPSL> child elements </xpsl:XPSL>
   <xpsl:XPSL> child elements </xpsl:XPSL>
</PipelineData>
```

1.3.3 Creating a single XPSL Instance from multiple XPSL Instances

Often, an application or user will need to combine multiple XPSL Instances into a single XPSL Instance. An example might be the necessity of combining one XPSL instance containing pipeline configuration data with another XPSL instance that contains the output from a simulation of that configuration. XML provides at least two methods of combining multiple elements into a single element. One is the use of XSLT stylesheets. The other is the recently released XInclude specification that allows one XML file to include portions of other XML files. Using XInclude, one could create an XPSL element that is the combination of the XPSL elements from several files.

Alternatively, an XPSL application may choose to consider all XPSL elements contained within a single XML file as belonging to a single XPSL instance. However, this is not a requirement of XPSL.

1.4 XPSL Applications

An *XPSL Application* is an application that uses XPSL Instances for input and/or output. Applications may impose certain constraints of their own on XPSL instances. For example, an application may require that an input XPSL file contain pipeline elements that are properly interconnected. Also, an XPSL application may allow for non-XPSL inputs or outputs as well.

However, an *XPSL Application* should not violate the XPSL Language Definition. The syntax of XPSL is defined by the *XPSL Schema*. However, the use and meaning of the XPSL language elements is further defined in the *XPSL Language Definition*. The *XPSL Language Definition* strives to define each of the XPSL terms precisely. XPSL also defines how the language elements are intended to be used. These definitions must be respected by XPSL applications in order for XPSL instances to be meaningful.

1.5 XPSL Extensibility

The PSIG Data Standards Committee recognizes that applications may require data that is not defined in the XPSL syntax. XPSL explicitly provides for this through an extension mechanism. XPSL provides “hooks” for XPSL extensions at every level of its language hierarchy.

1.6 References

XPSL (Extensible Pipeline Simulation Language) Plan: PSIG 001/2002

Common Language Standard: PSIG 001/2000

2 XPSL Schema Conventions

2.1 XPSL Namespace

- There is a single XPSL namespace
- Name : http://www.xpsl.org
- Resolves to a specific network location
- Individual schema files can be retrieved by users from this location
- Standard prefix is xpsl
- xmlns:xpsl=http://www.xpsl.org
- The XPSL Schema namespace is defined by several individual XPSL schema files that separately define various aspects of the schema.
- The individual XPSL schema files are formally referred to as *XPSL Schema Documents.* These files are authored and maintained by the PSIG XPSL Committee (currently PSIG Data Standards Committee).
- Individual XPSL Schema documents are combined using the XSD include attribute.
- A master XPSL file named XPSL.xsd will be maintained that includes the latest version of all individual XPSL files.
- The target namespace for every XPSL xsd file is http://www.xpsl.org
- Individual XSD files are named with the pattern XPSL*name_v*.xsd where *v* represents the version
- XPSLUnits_1_4.xsd
- XPSLConfig_1_4.xsd
- XPSLOutputs_1_4.xsd
- The version of each XPSL Schema Document will be stated in the <xsd:schema> element and will be consistent with the schema document name.

2.2 XPSL Design Philosophy

- XPSL is intended to provide a common language for describing pipeline objects for applications
- XPSL is intended to ensure common usage and information sharing.
- XPSL is not intended to ensure that the objects described in an XPSL instance file are complete or valid from a simulation perspective

2.2.1 Ramifications

- Except in special circumstances, XPSL will make all inputs optional
- Devices will not need to be connected to other devices
- All elements and attributes are optional
- The minOccurs="0" attribute should be defined wherever possible

2.3 Naming Conventions

- Consistent names are easier to understand, remember, and maintain
- Guidelines apply to XPSL schema – not to user defined objects

2.3.1 Naming Convention for Types and Groups

- Use Mixed Case (first letter of each word capitalized with no underscores or hyphens between words)
- Type names will have “Type” as a suffix
- Group names will have “Group” as a suffix

Examples:

- NodeType
- InlineDeviceType
- NodeGroup

Mixed case will be used for

- Simple Types
- Complex Types
- Attribute and Element Groups

2.3.2 Naming Convention for Elements and Attributes

Use camel case (first letter of each word except the first capitalized with no underscores or hyphens between words).

Examples:

- node
- inlineDevice

2.3.3 Naming Convention for Enumerations

Use mixed case (first letter of each word capitalized with no underscores or hyphens between words).

Examples :

- DifferentialPressure
- APIGravity

2.3.4 Abbreviations

The use of abbreviations in names is discouraged. Any abbreviations that are used should be standard and also documented in the XPSL documentation in a list of standard abbreviations.

2.3.5 Name Length

XPSL names that are longer than 20 characters are discouraged. The shortest descriptive name should be used. However, clarity will be valued above brevity.

2.4 Schema Model Groups

XSD provides three types of “collection” definitions, xsd:all, xsd:sequence, and xsd:choice. The XPSL preferred type is “xsd:all”.

2.4.1 The xsd:all Model Group

This is the preferred XPSL model group.

Example:

```
<xsd:complexType name=“PipeType”>
   <xsd:all minOccurs=“0”>
     <xsd:element name=“OD” type=“xsd:double” minOccurs=“0”>
     <xsd:element name=“WT” type=“xsd:double” minOccurs=“0”>
```

```
</xsd:all>
</xsd:complexType>
```

Advantages

- Not dependent on order
- With minOccurs set to 0, allows 0 or 1 (only) of each of the elements to be defined.

Disadvantages

- Cannot allow the declaration of more than one element of a specific name.

2.4.2 The xsd:sequence Model Group

Use only when element order is both natural and expected

It may also be used when the user may specify more than one element of the same name.

Example

```
<xsd:complexType name="NodeSetType">
  <xsd:sequence>
        <xsd:element name="upNode" type="xsd:token">
        <xsd:element name="downNode" type="xsd:token">
  </xsd:sequence>
</xsd:complexType>
```

2.4.3 The xsd:choice Model Group

- The xsd:choice model group is used to specify a choice between several options.
- When using xsd:choice with maxOccurs > 1, make sure that xsd:all or xsd:sequence is not more appropriate.

2.5 XPSL Element String Content

XSD provides three string types: xsd:string, xsd:normalizedString and xsd:token. XPSL will always use xsd:token as the string type for any string attributes and elements xsd:token does the following string conversions

- Truncates all leading and trailing whitespace
- Collapses all other contiguous white space (tabs, spaces, CR's and LF's to a single space)

2.6 Providing for XPSL Extensibility

Extensibility will be granted through a a global XPSL type called ExtensionType defined as follows:

```
<xs:complexType name="ExtensionType" final="">
   <xs:sequence minOccurs="0">
      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

Attribute extension is not directly allowed. However, user defined types may that are added to an ExtensionType object may have any number of attributes.

The ExtensionType is defined with the following attributes :

- processContents="lax" informs the schema processor to validate any contents for which it can find a declaration. If a declaration is not found, no errors will be reported. This allows users to create extensions without forcing them to define a schema for the extensions.

- final="" allows users to derive types from by extension or restriction from the base xpsl:ExtensionType.

An XPSL:ExtensionType element should be defined at every level of a schema document, starting at the root level (below <xsd:schema>). Every XPSL object should allow extension.

Each ExtensionType element should be declared with the attributes minOccurs="0" and maxOccurs="unbounded".

The XPSL:ExtensionType element will be defined in an XPSLGlobals.xsd Schema Document along with the definition of XPSL units. Every other XPSL schema document must include the XPSLGlobals.xsd document.

2.7 Global versus Local Types

Global types are preferred when a type declaration is expected to be used more than once. Local types are preferred otherwise.

2.8 Unqualified Names Allowed in XPSL Instances

Unqualified element and attribute names will be allowed (elementFormDefault="unqualified" attributeFormDefault="unqualified" />). In other words, it will not be necessary for users to include the xpsl: prefix on each XPSL element or attribute.

2.9 No Constraints Defined in XPSL Schema

XSD provides the capability of defining constraints similar to primary / foreign key constraints in databases. For example, nodes referenced by devices could be constrained to exist in as children of a <Nodes> element.

However, XPSL will not include constraints in its schema. Imposing of constraints and checking for consistency between XPSL elements will be left to the application writers.

2.10 Elements Should Be Nillible

Generally, every XPSL element will be allowed to be take the value of "nil" by using the xsd:nillible="true" attribute setting in the type declaration.

This allows one to use elements in XPSL instance files without defining their values. One use of this is to specifically indicate that data is not available for an elements value.

Example :

```
<xpsl:Pipe>
   <Diameter xsi:nil="true"/>
</xpsl:Pipe>
```

Note: using xsi:nil requires the use of the XSD Schema-instance namespace.

2.11 Elements Have Name and ID Attributes

Each XPSL element will defined to have name and id string attributes. Therefore, any element in a user's XPSL instance document may have a name. Element names are used to cross-reference XPSL objects and form an important part of an XPSL instance. The ID attribute is provided as an additional element identifier. XPSL does not use this identifier directly though XPSL applications may chose to use it.

These attributes are defined in a BaseAttGroup attribute group that is included in the definition of all XPSL types. The group is defined as follows:

```
<xs:attributeGroup name="BaseAttGroup">
   <xs:attribute name="name" type="xs:token" use="optional"/>
   <xs:attribute name="id" type="xs:token" use="optional"/>
</xs:attributeGroup>
```


2.12 Formatting of XPSL Schema Documents

The XPSL Schema Documents will be formatted to allow one to easily read the document in a text editor.

Spaces will be used for indentation.

Tabs will never be used

Indenting will be used to clearly show document structure. Three (3) spaces will be used for each new level of indentation.

When declarations are continued on subsequent lines, all continuation lines will be indented by the same amount and will be indented greater than the amount of the original declaration.

No line in an XPSL Schema Document will be longer than 100 characters.

2.13 Versioning of XPSL Schema Documents

The <schema version="version" attribute will be used to indicate the version of an XPSL schema document.

The name of the schema document will also include the version number.

When a new version of an XPSL schema document is created, the new version will be a complete restatement of that portion of the schema (e.g. units). In other words, it will not import or redefine earlier versions of the document.

Every attempt will be made to make new versions of a schema document backwards compatible with earlier versions. An instance document that validates against an earlier version of the schema should validate against more recent versions.

2.14 Documenting XPSL Schema Documents

Each version of the XPSL Schema will be accompanied by documentation that describes that XPSL Schema.

Documentation should include the following:

- The intended purpose of the schema document.
- A description of all XPSL objects defined in the schema document including a description of each element and content (attributes and child elements)
- A list of all abbreviations used in the schema type names
- Examples of use of the schema

2.15 The XPSL <schema> Element Declaration

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:xpsl="http://www.xpsl.org" xmlns=http://www.xpsl.org
   targetNamespace="http://www.xpsl.org" elementFormDefault="unqualified"
   attributeFormDefault="unqualified" version="1.0">
```

3 Schema Documentation Diagram Symbols and Conventions

The XPSL schema documentation in the following sections makes use of diagrams generated using the Altova XMLSpy Enterprise Suite. A number of graphical symbols and conventions are used in the diagrams. This section describes these symbols and conventions.

3.1 Optional versus Required Elements

Generally, any element, attribute, or group of elements or attributes may be defined in a schema as optional or required. The convention for representing optional components is to use a dashed boundary for the components diagrammatical representation. The convention for representing required components is to use

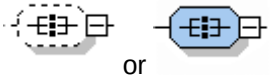
a solid boundary. In the preceding figure, XPSL is a required element and configurations is an optional element.

3.2 XSD Model Groups

3.2.1 The xsd:all Group

Three types of groupings are allowed in XSD schemas. All three of these are used in XPSL.The first, and most commonly used in XPSL, is the xsd:all group. This group is represented by the symbol

Figure 2 - xsd:all Symbol



or

If the symbol has a dashed border, the schema has been defined so that this group is optional rather than required. A solid border indicates that this group of elements is not optional.
The xsd:all group is followed by a list of child elements, ALL of which are allowed no more than one time. The schema writers may define whether or not the child elements are optional. Referring to figure 2, one can see that the *XPSL* element has the following optional child elements: *configurations*, *libraries*, *snapshots*, *extension*, and *comment*.
The xsd:sequence Group
A sequence group of element declarations is used to indicate a set of elements that must appear in a certain order. Unlike the all group, the sequence group allows more than one copy of each element to be included. Generally XPSL avoids defining the order in which element should occur. Instead, XPSL uses the sequence group to define an element that may occur any number of times.

Figure 3 - xsd:sequence Symbol

Figure 3 shows the symbol used to specify the sequence group. This figure also illustrates how the limits on the number of elements that can be included are denoted. The 0..infinity symbol indicates that any number of elements (0 to infinity) are allowed.

Figure 4 - xsd:sequence Example

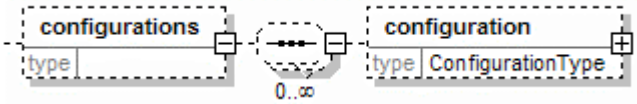


Figure 4 illustrates the use of the sequence group in the XPSL schema. The sequence group is used to define that the *configurations* element can include is zero or more *configuration* child elements.

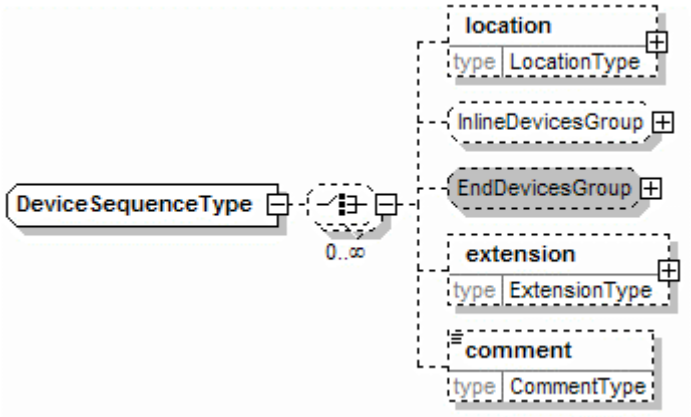
3.2.2 The xsd:choice Group

A choice group of element declarations is used indicate that only one of the child elements may appear. A common use for choice groups is to allow any number of children to appear in any order. This can be accomplished by allowing the choice group to be repeated any number of times. XPSL uses choice groups precisely for this purpose in its definition of the configuration element type.

Figure 5 - xsd:choice Symbol

Figure 5 shows the symbol used for the choice group. Figure 6 shows how a choice group is used to define a device sequence that consists of any number and sequence of location, InlineDevices, EndDevices, extension and comment elements.

Figure 6 - xsd:choice Example

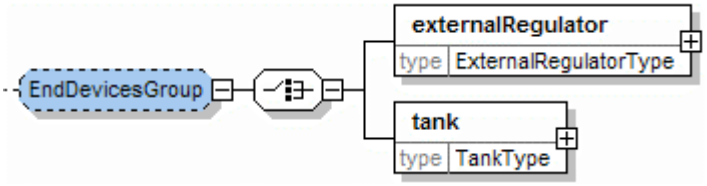


3.3 Element Group Definitions

For convenience and defining a schema, an element group may be defined. This is particularly useful when one wishes to define a group of elements that may be used in different places in the schema. An element group is different from the sequence, all, and choice model groups that are used to define the order and number of child elements that may appear at a particular point in the schema.

Figure 6 above illustrates the use of two element groups named InlineDevicesGroup and EndDevicesGroup. Expanding the EndDevicesGroup, as in Figure 7, one can see that this group consists of a choice of two elements: external regulator or tank. In this particular example, the element group is optional. Therefore it was not necessary to make its child elements optional.

Figure 7 - Element Group Example



3.4 Complex Type Definitions

Generally speaking, any complex element in XPSL that are used more than once are defined as a complex type. A complex type defines an element, its child elements, and its attributes.

Figure 6 illustrates the definition of the DeviceSequenceType. An element of this complex type allows any number of children of the location type, extension type, comment type, or any of the elements defined in the InlineDevicesGroup and the EndDevicesGroup.

4 XPSL Documentation

4.1 The XPSL Root element

Each XPSL instance consists of an XPSL element and its children. The XPSL element and its child elements are shown diagrammatically in Figure 9. Its attributes are shown in Figure 8.

Figure 8 – The XPSL Root Element's Attributes

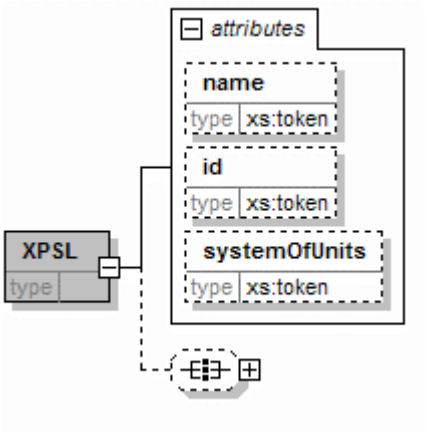
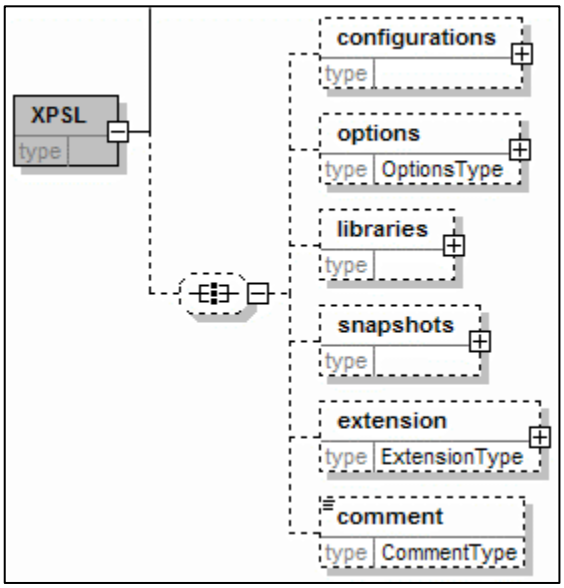


Figure 9 – The XPSL Root Element's Children



4.2 Specifying System of Units, Names, Extensions and Comments

Figure 8 shows the standard attributes that are defined for virtually every element in the XPSL language. The name attributes allows one to name an element. Some elements, such as those in libraries have no purpose unless they are named. It is not an XPSL error to specify duplicate names for the elements of the same type (e.g. two different block valves with the same name).

The id attribute is provided for additional element identification. XPSL does not use this attribute internally for element crossreferencing. It uses the name for that purpose.

The systemOfUnits attribute is used to specify a systemOfUnits. Generally, this should correspond to a systemOfUnits element already defined in a systemOfUnitsLibrary (discussed later). However, it is possible

that application writers may build in to their applications some predefined systems of units that may be specified by name without a separate specification in the systemOfUnitsLibrary.

At virtually every level of the XPSL hierarchy, extension and comment elements may be specified. The extension element is defined to allow any child attributes. When XPSL needs to be extended to meet specific user requirements that are not already met by XPSL, the extension element is the proper place to add the extension. Each extension element may have any number of child elements.

Users and application writers are urged to standardize their extensions and to provide their extensions to the XPSL Data Committee for incorporation into XPSL.

Comment elements may be used to insert comments at most levels of the XPSL hierarchy.

Figure 10 – Simple XPSL Instance with Comment and Extension Elements

```
<XPSL name="My Simulation" systemOfUnits="English">
    <extension>
      <VendorSpecial>anything goes here
      </VendorSpecial>
    </extension>
  <comment>My first XPSL!</comment>
</XPSL>
```

4.2.1 System of Units Overriding

XPSL allows one to specify the SystemOfUnits attribute in most elements in its hierarchy. This means that a SystemOfUnits specification may be entered at a high level (even in the XPSL root element) and then a several lower levels. The convention for resolving SystemOfUnits conflicts is as follows:

An XPSL application attempts to apply the SystemOfUnits specification to the element defining the SystemOfUnits and all element below that element. When a lower level element is found with a SystemOfUnits specification, the new specification applies to all elements below that level.

4.3 Using libReference Attribute to Reference an object in a Library

XPSL provides the ability to archive configuration data in XPSL libraries. Any element already defined in an XPSL library may be referenced when defining a new element. Figure 11 shows the libReference attribute defined for the configuration element. Therefore, when defining a configuration, one my include a reference to a predefined configuration already stored in the XPSL configurationLibrary discussed later.

The following guidelines are provided for the use of the libReference element:

1. The libReference attribute is accepted at almost all levels of a configuration definition
2. libReference is followed by the name of an object in the corresponding library
3. Library references may be nested to any level. An object in a library may reference another object in the library and so on
4. Any subset of an element's content may be defined in a library element
5. When a libReference is used, the element's contents are initialized to referenced element contents. Any local element content is then processes, added to or overriding the content imported from the library element.

These guidelines are part of the XPSL specification. XPSL applications should implement the libReference functionality as specified above.

4.4 XPSL Configurations

The XPSL configurations element is provided to allow one to specify one or more detailed pipeline configurations. The configurations element can have zero or more configuration child elements as shown in Figure 11. Each configuration child element allows up to six attributes. Four have been discussed earlier.

This element also includes a version attribute that is provided for the convenience of a user to identify the version of a configuration.

Figure 11 - configuration Element Attributes

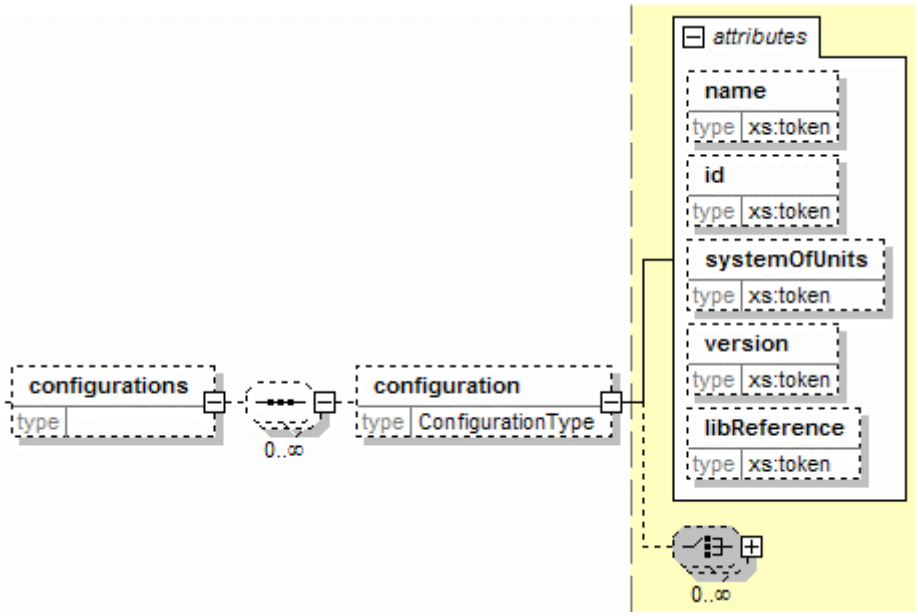
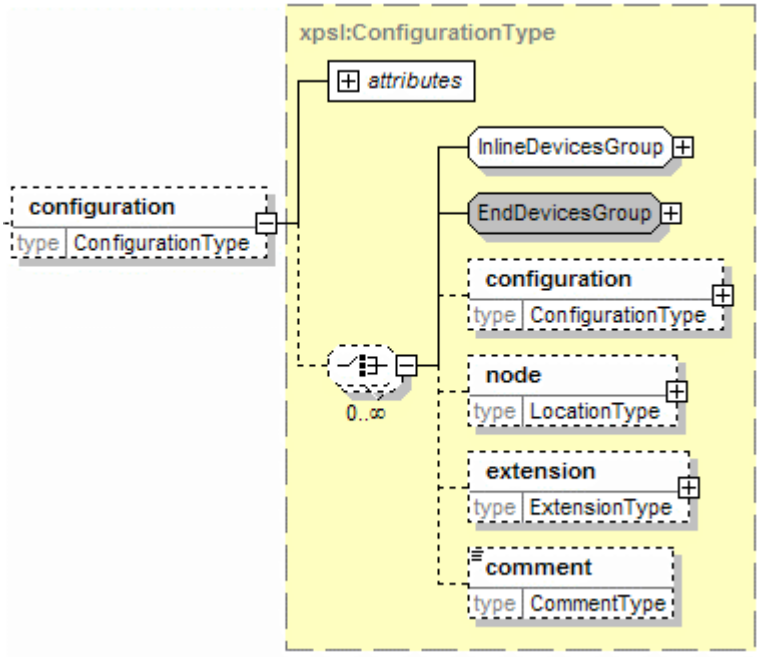


Figure 12 shows a high level view of the configuration element. This element may contain an unbounded number of child elements.

Figure 12 - configuration Element's Child Elements – High Level View



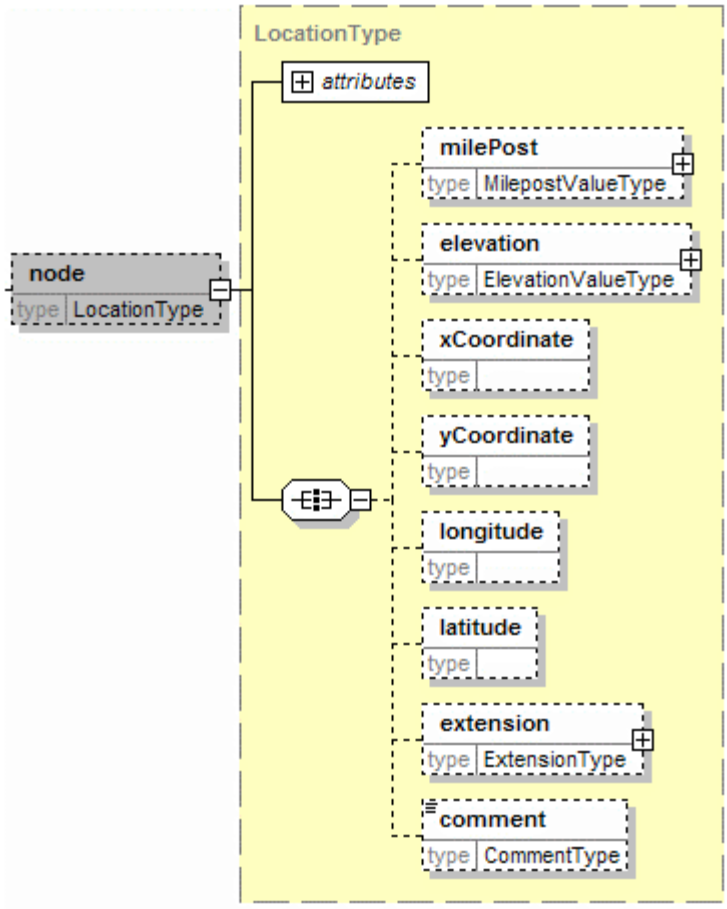
4.4.1 The configuration Element's node Element

The node element is used to specify points of connection between devices in a pipeline configuration. The node element has the typical set of attributes (name, id, systemOfUnits, and libReference) as well as a set

of child elements that provide more detailed information about the node. Most commonly used are the milePost and elevation elements.

The xCoordinate and yCoordinate are used for applications which need to store graphical information about the location of the node in a graphical representation of a pipeline network.

Figure 13 - node Element



A sample node element in XPSL is shown below:

```
<node name="Timbuktu">
   <milepost>7005.735</milepost>
   <elevation>2783.35</elevation>
</node>
```

4.4.2 Strongly Typed Data Values

Note that the elevation and milePost elements are of types ElevationValueType and MilepostValueType. Virtually numerical scalar data value in XPSL is assigned a specific Value Data Type. The Value Data Type (e.g. MilepostValueType) specifies the type of unit conversion that is applied to values of that data type. Also, every data element has child elements that may be used to specify the data quality, data source, and error (uncertainty) associated with that data element.

Figure 14 shows an example of this concept using the elevation element. The label attribute is provided to specify a units label for the data value. In general, this is most useful in output files. The label should be ignored by applications reading in a configuration. It is not possible to specify the units of the data value using the label attribute. The systemOfUnits attribute should be used for that purpose.

The dataQuality element is an enumerated element that allows only the following character content:

Known

or

Unknown.

The source element is used to document the source of the data: It is an enumerated element that allows only the following content:

Scada

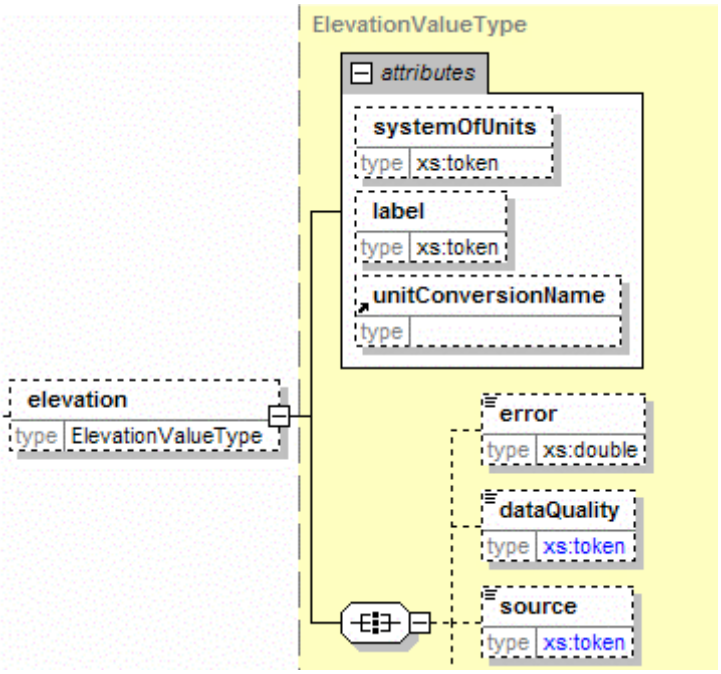
ManualEntry

Simulation

An example of the use of these elements in the XPSL node element is shown below:

```
<node name="Timbuktu">
   <milepost>7005.735</milepost>
   <elevation>2783.35
      <dataQuality>Known</dataQuality>
      <source>Scada</Source>
   </elevation>
</node>
```

Figure 14 - Example Value Data Type: ElevationValueType



4.4.3 Inline Devices

The configuration element may include an unbounded number of Inline Devices in any order. Figure 15 shows the possible inline device elements.

Figure 16 shows the attributes of the inline devices. The figure shows the blockValve device. However, all inline devices have this minimum set of attributes. Pipes have two additional attributes.

A simple (but incomplete) configuration with a block valve element is shown below:

```
<configuration>
   <blockValve name="StopFlow" upNode="up" downNode="down"/>
</configuration>
```

Figure 17 and Figure 18 show the child elements for the blockValve and compressor elements. The key difference is that the blockValve has a single PointDataSetType element, *settings*, element while the compressor has three elements of that type: *upSettings*, *settings*, and *downSettings*.

The PointDataSetType defines the content of these settings elements (attributes and elements). The PointDataSetType will be discussed in a later section. However, for now, accept the fact that this data type is an all-encompassing data type including:

1. A wide range of data values
2. Sets of device limits
3. Control mode specification
4. Ordered data sets suitable for defining compressor maps
5. and more…

Figure 15 - Inline Devices

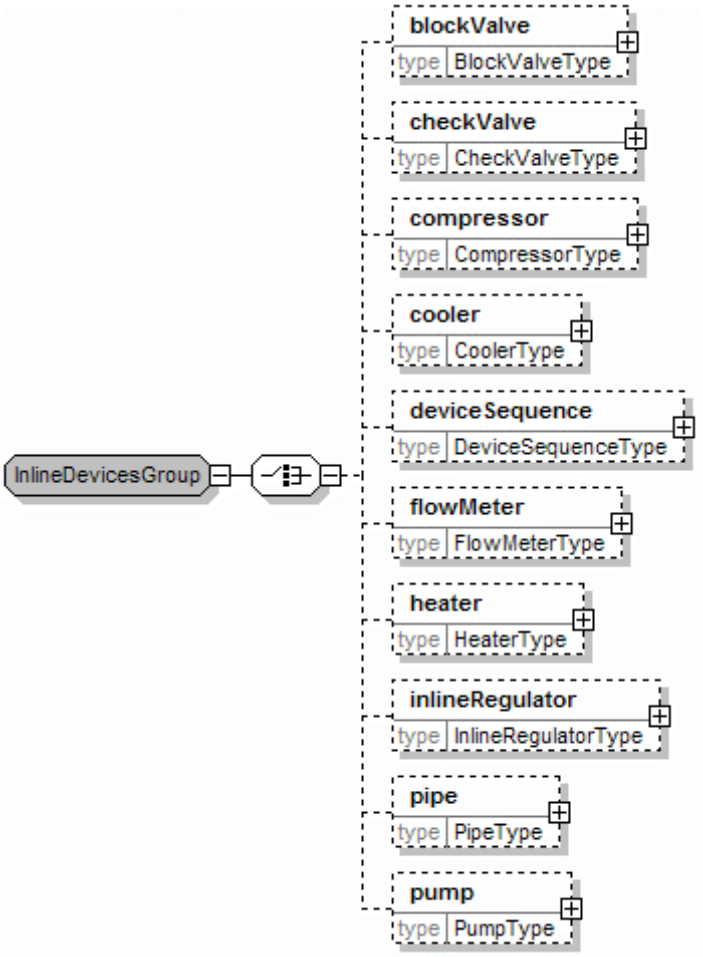


Figure 16 - Inline Device Attributes – blockValve Example

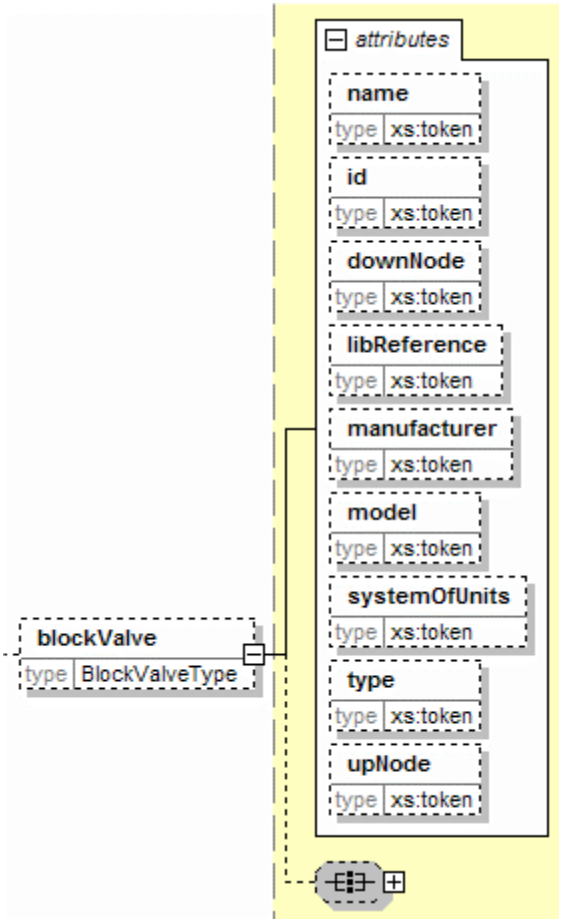


Figure 17 - blockValve's Child Elements

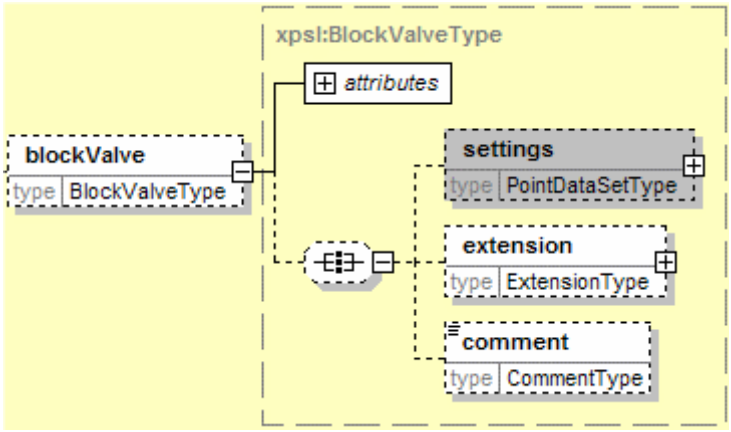
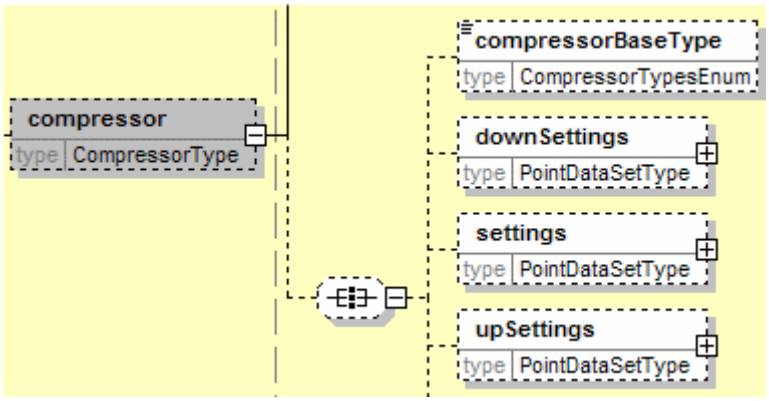


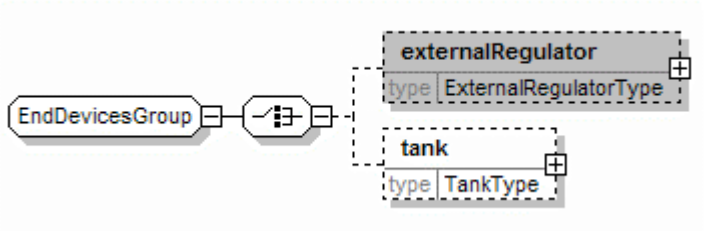
Figure 18 - compressor's Child Elements



4.4.4 End Devices

The configuration element may include an unbounded number of End Devices in any order. Figure 19 shows the possible end device elements. The primary difference between end devices and inline devices is that end devices have a single node reference, rather than an upNode and downNode.

Figure 19 - End Devices



4.4.5 The settings Elements (settings, upSettings, and downSettings)

Figure 17 and Figure 18 show the child elements for the blockValve and compressor elements. The key difference is that the blockValve has a single PointDataSetType element, *settings*, element while the compressor has three elements of that type: *upSettings*, *settings*, and *downSettings*.

The PointDataSetType defines the content of these settings elements (attributes and elements). The PointDataSetType will be discussed in a later section. However, for now, accept the fact that this data type is an all-encompassing data type including:

1. A wide range of data values
2. Sets of device limits
3. Control mode specification
4. Ordered data sets suitable for defining compressor maps
5. and more…

Devices with a single settings element are the following:

a. Block Valve
b. Check Valve
c. Flow Meter
d. Tank
e. External Regulator
f. Pipe

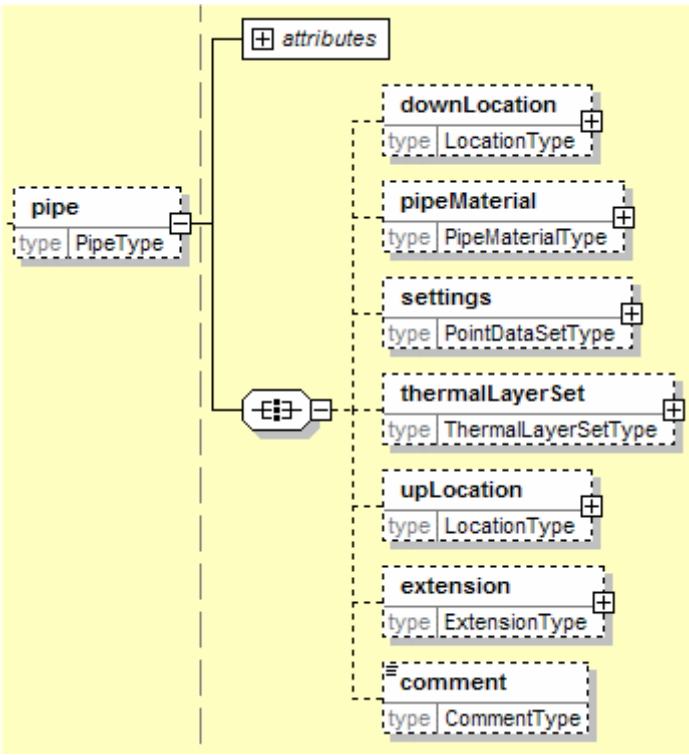
Devices with three settings elements (settings, upSettings, and downSettings) are the following:

a. Compressor
b. Pump
c. Inline Regulator
d. Cooler
e. Heater

4.4.6 The pipe Element

Detailed configuration data is provided for pipe elements. The pipe element has two more attributes than the other inline devices: pipeLoops and pipeEndsConstrained. pipeLoops="x" specifies that the pipeline being defined is actually two identical pipes in parallel. pipeEndsConstrained="false" specifies that the pipe ends are not constrained. The default assumes that pipeLoops="1" and pipeEndConstrained="true".

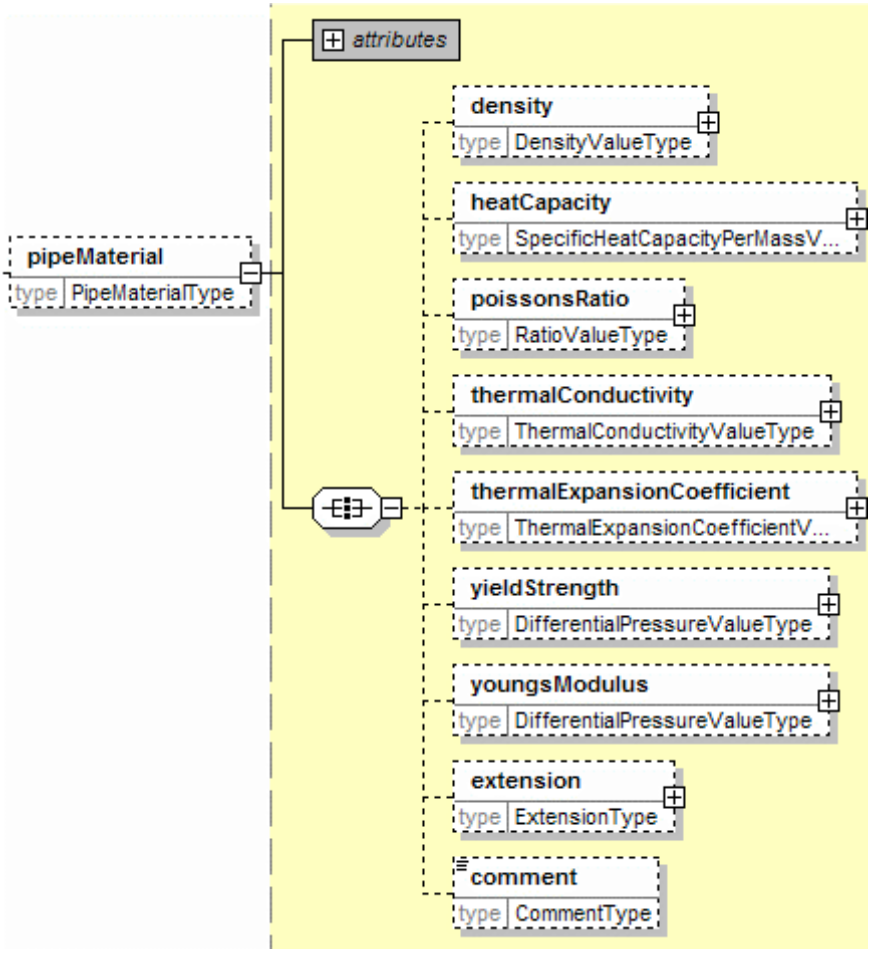
The pipe element also has specialized child elements as shown in the following figure:



Note that pipes have downLocation and upLocation elements. These elements may be used to specify elevation, milePost, and other properties of the pipe ends that are different from the upNode and downNodes specified for the pipe. In this case, while the upNode and downNode attributes define network interconnectivity, any elements or attributes defined in the pipe's upLocation and downLocation elements override those defined for the nodes referenced by upNode and downNode attributes.

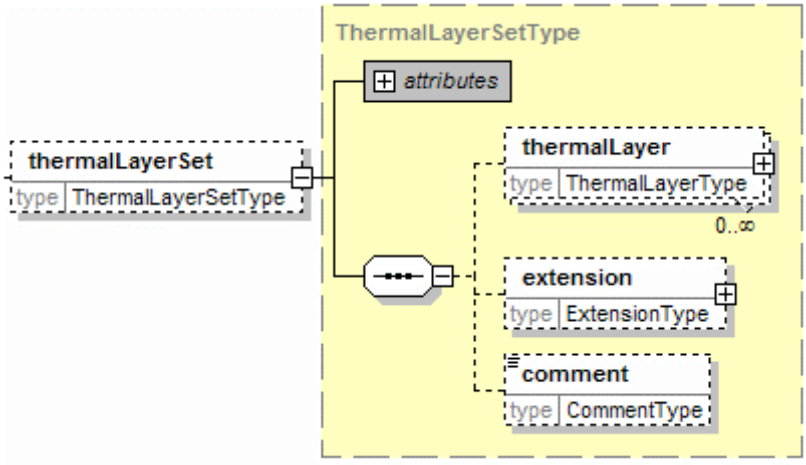
4.4.6.1 pipeMaterial Elements

Pipes have pipeMaterial elements that have child elements that define the pipe material properties. There is a pipeMaterialLibrary in the library element that may be referenced using the pipeMaterial's libReference attribute. Generally this is much more efficient than defining the pipeMaterial properties in each pipe element.

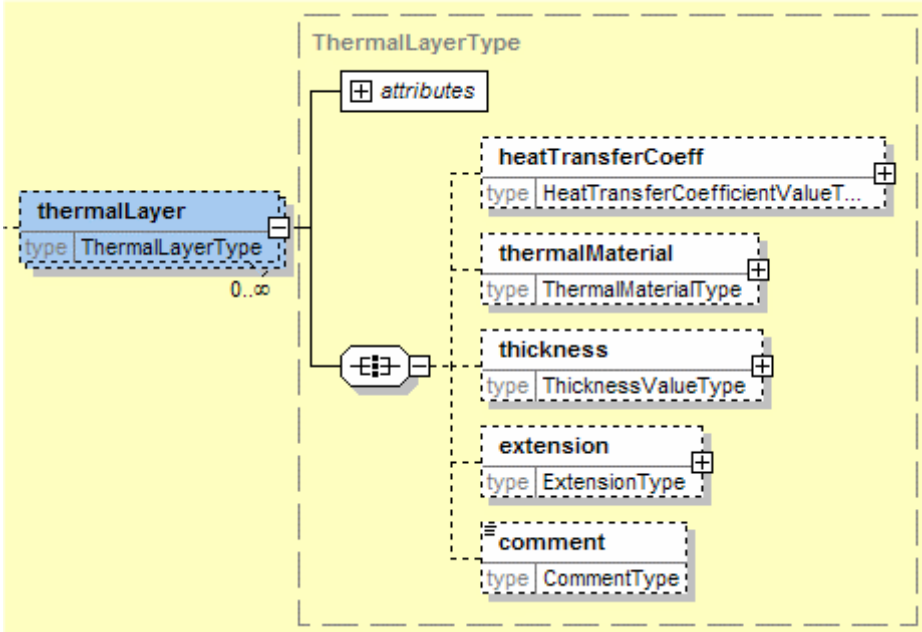


4.4.6.2 thermalLayerSet

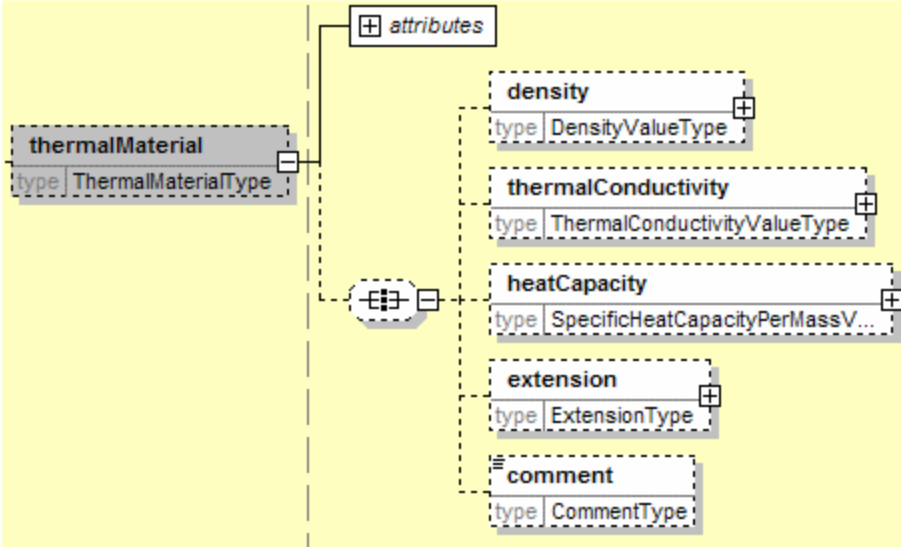
To define the thermal properties of the pipe surroundings, one may define a thermalLayerSet for each pipe. The thermalLayerSet consists of one or more thermalLayers as shown in the following figure.



Each thermalLayer has a thickness and a thermalMaterial and a heatTransferCoeff for some types of thermal models as shown in the following figure:



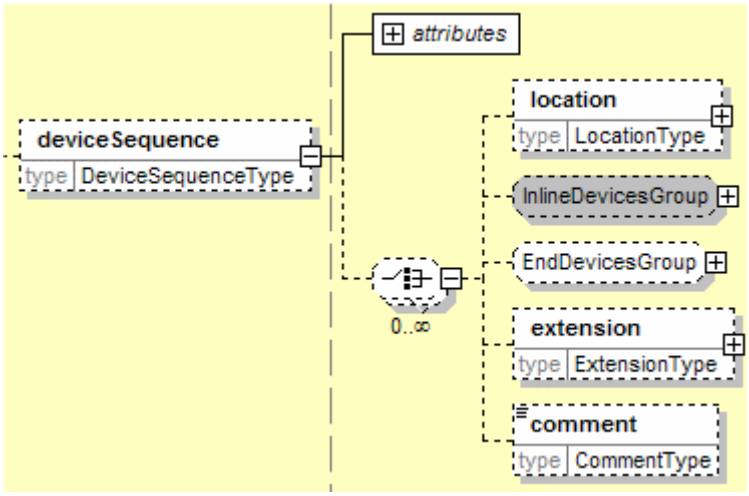
The thermalMaterial element is shown in the following figure:



There is a thermalMaterialLibrary and thermalLayerSetLibrary in the libraries element. Preconfigured library elements may be used instead of specifying this detail with every pipe.

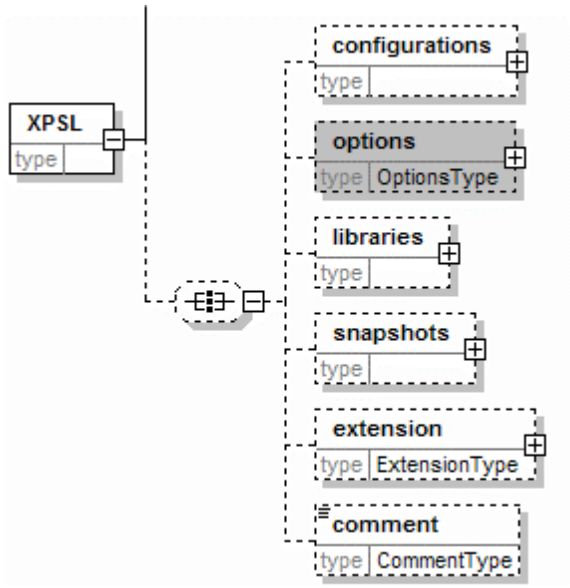
4.4.7 The deviceSequence Device

The deviceSequence element may be used to specify a set of pipeline devices (e.g. pipes, block valves and tanks) that are connected end-to-end. These may be interspersed with location elements. location elements, when used, specify the milepost and elevation of the device immediately following the location element in the deviceSequence.

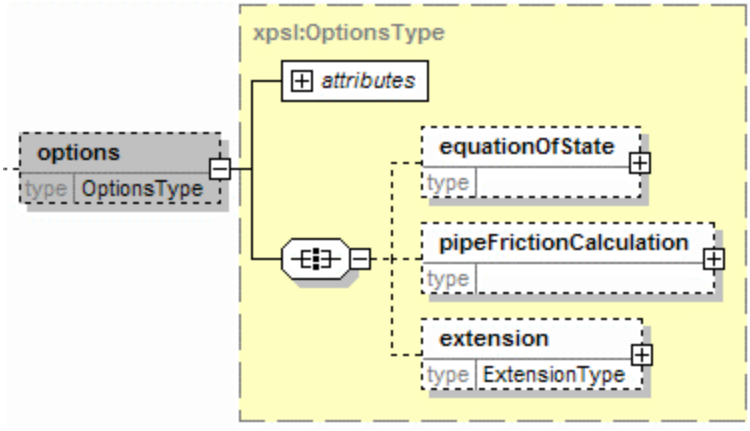


4.5 The options Element

The options element is a child of the XPSL root element.



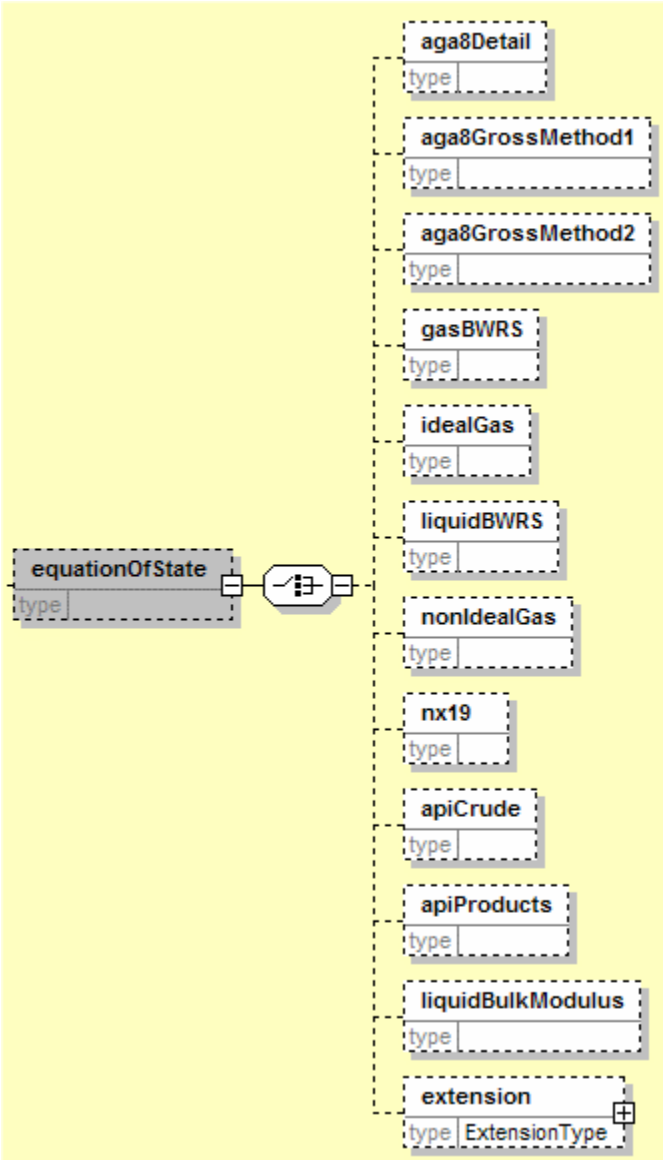
The options element has two child element of interest: equationOfState and pipeFrictionCalculation as shown in the following figure:



4.5.1.1 equationOfState Element

The equationOfState element may have a single child element selected from its possible child elements shown in the figure below. The presence of the child element specifies the equation of state to be used. For instance, to specify the AGA 8 Detail equation of state, one would enter the following into the options element:

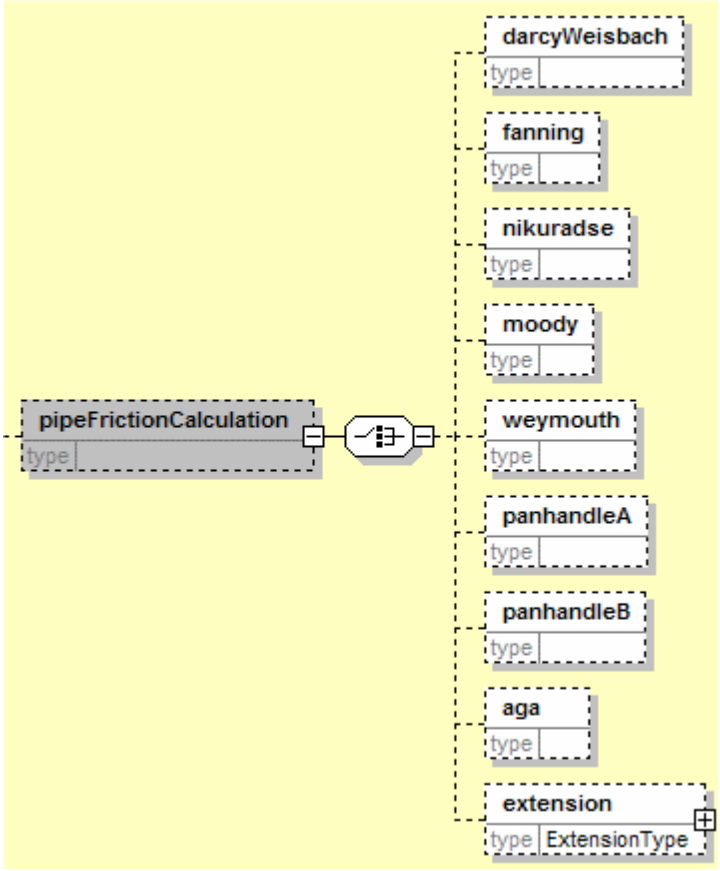
```
<equationOfState>
   <aga8Detail/>
</equationOfState>
```



4.5.1.2 pipeFrictionCalculation element

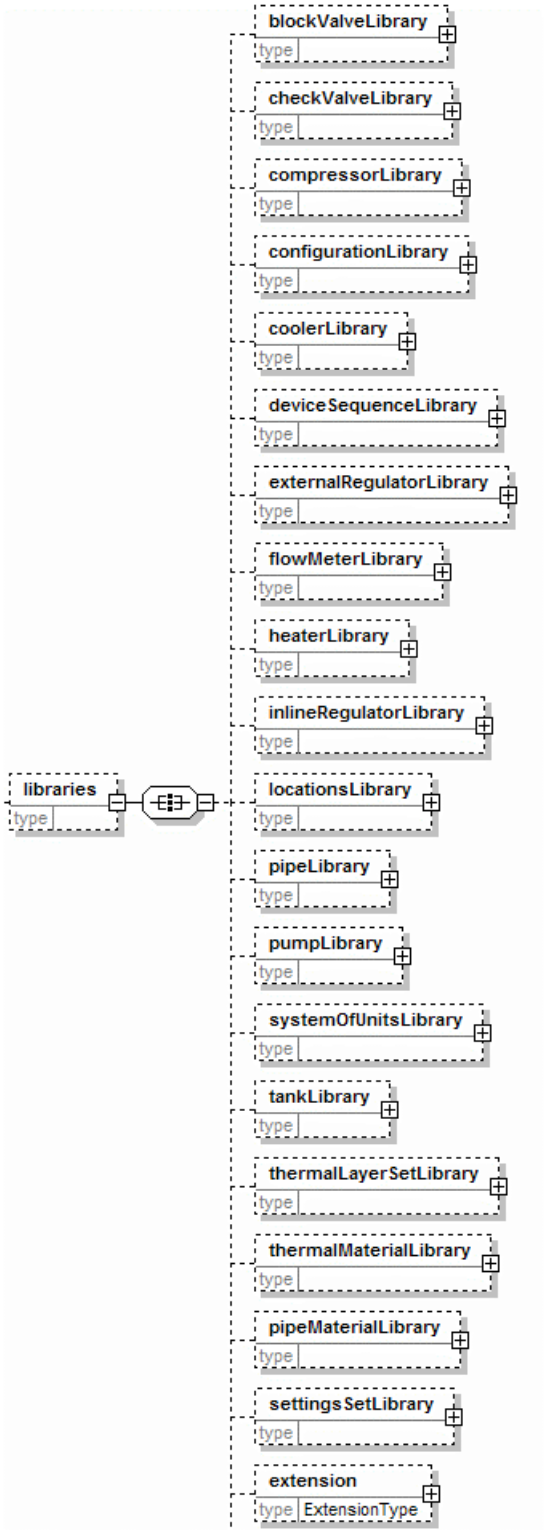
The pipeFrictionCalculation element defines the type of equation used to calculate frictional head loss in pipes. The possible choices are shown in the figure below. To specify the Darcy Weisbach calculation, one would include the following elements in the options element:

```
<pipeFrictionCalculation>
   <darcyWeisbach/>
</pipeFrictionCalculation >
```

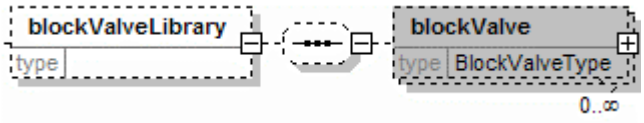


4.6 XPSL Libraries

The XPSL root element contains a libraries element. The XPSL libraries may be entered below this element. Each device type has its own library.



The typical library structure is shown in the following figure:



In this example, the blockValveLibrary consists of an unbounded number of blockValve elements (with there associate attributes and child elements).

Library elements are referenced using the libReference attribute. Every element that can reference a preconfigured library element has this attribute. The libReference is followed by the name of an object in the corresponding library (e.g. libReference=”blockValve1”).

Library objects may also reference other library objects of the same time. Library references may be nested to any level.

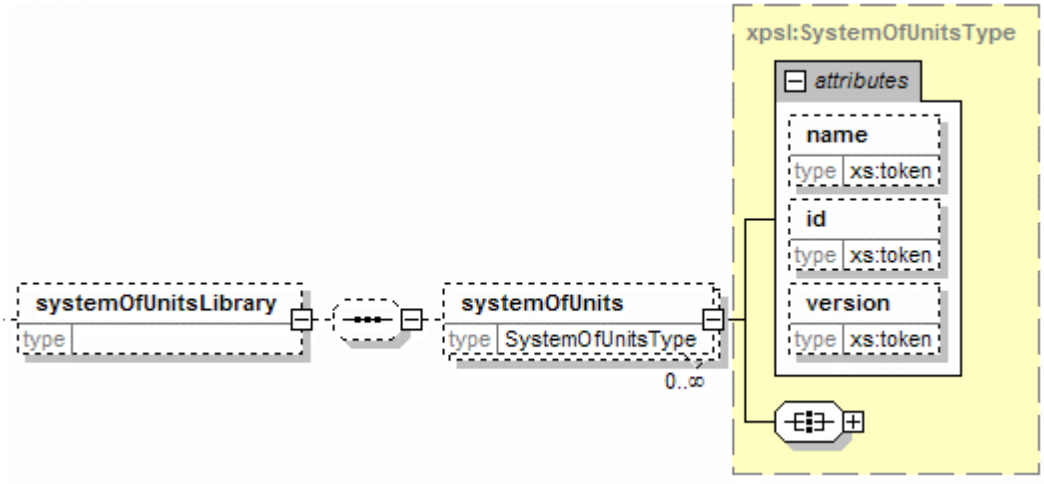
An element in a library may define the complete element content or any subset of an element’s content.

When a libReference is used, the element’s contents are initialized to the referenced element’s contents. Any local element content is then processed, added to or overriding the content imported from the library element.

There are two special libraries: systemOfUnitsLibrary and settingsSetLibrary. These are described in their own sections.

4.7 Systems of Units

- A System Of Units has a name
- Any number of Systems of Units may be defined
- Systems of Units are defined using a systemOfUnits element in the systemOfUnitsLibrary.



Unit conversions are defined in terms of a multiplier of the corresponding SI unit plus an offset in user units. An example of a systemOfUnits element defining only two conversions, area and length, is shown below:

```
<systemOfUnits name="English">
  <xpsl:area label="ft^2">
    <xpsl:linearConversion siMultiplier="0.09290304“
                           userOffset="0.0"/>
  </xpsl:area>
  <xpsl:length label="ft">
    <xpsl:linearConversion siMultiplier="0.3048“
                           userOffset="0.0“/>
  </xpsl:length>
</systemOfUnits>
```

Any or all of the following units conversions may be defined:

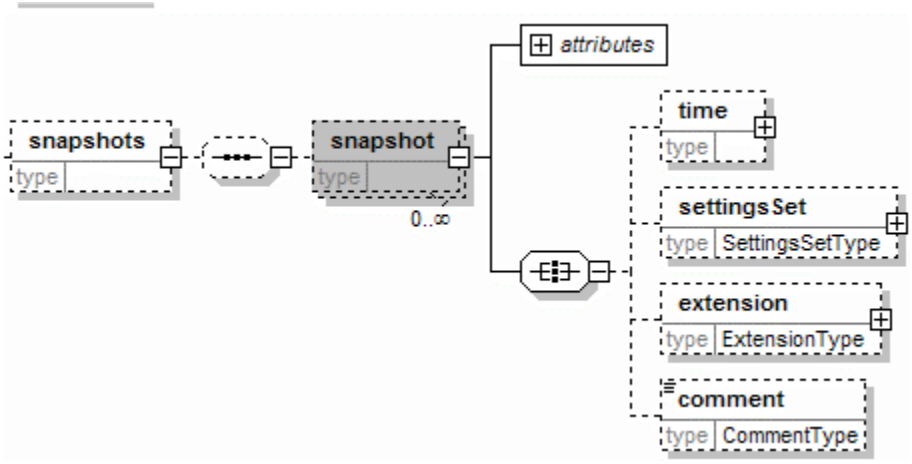
angularVelocity apiGravity area barometricPressure bulkModulus chamberVolume draConcentration cost costPerMass costPerTime costPerVolume days density diameter differentialPressure differentialTemperature dynamicViscosity elapsedTime elevation gasCompressibilityFactorZ energy energyFlow efficiency fluidLevel flow fraction head headGradient heatingValue heatingValuePerMass heatTransferCoefficient heatRate hours hydraulicResistance inverseTemperature isentropicEfficiency kinematicViscosity length linePack longTime mass massFlow massPackRate massPackRatePerLength mediumDistance milepost minutes molecularWeight moleFraction moles nondimensionalFlow nondimensionalHead packRate packRatePerLength pipeLength power pressure rateOfRotation ratio pipeRoughness seconds specificEnthalpy specificGravity specificHeatCapacityPerMass specificHeatCapacityPerMole temperature tensileStrength thermalConductivity thermalEfficiency thermalExpansionCoefficient thermalFlow thermalFlowPerLength thermalFlowPerVolume thickness torque unity valveCv valveCg valveTransitTime velocity volume volumetricEfficiency

4.8 Snapshots and Settings Sets

- A Snapshot is time stamped
- A Snapshot represents the state of the pipeline application at some point in time
- The settingsSet element contains the pipeline state at a point in time

The snapshot element is shown in the figure below. The snapshots element contains an unbounded number of snapshots, each for a specified point it time.

Snapshots may be used both as inputs and outputs from a pipeline simulation application.

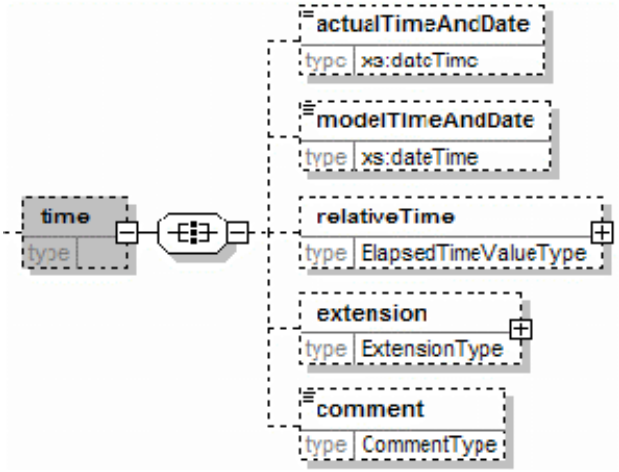


4.8.1 The time element

The time element is used to specify the snapshot time. The actualTimeAndData element specifies the clock time that the snapshot corresponds to. Generally, this is most useful for output snapshots.

The modelTimeAndData stores the simulation time and date.

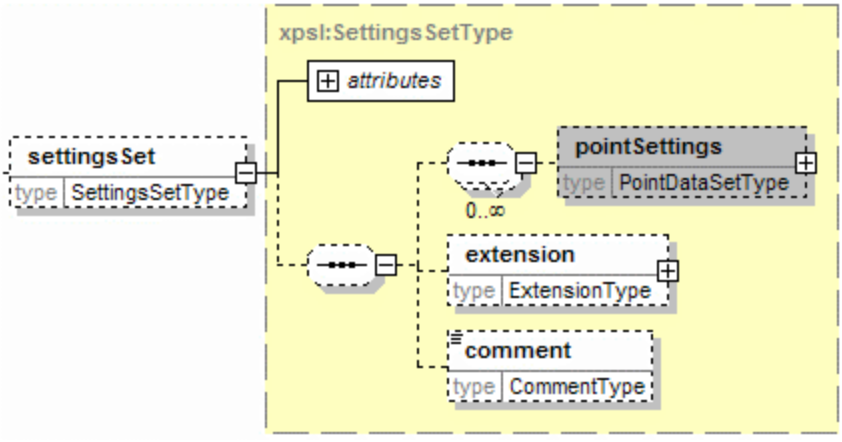
The relativeTime allows one to specify the time relative to the start of a simulation.



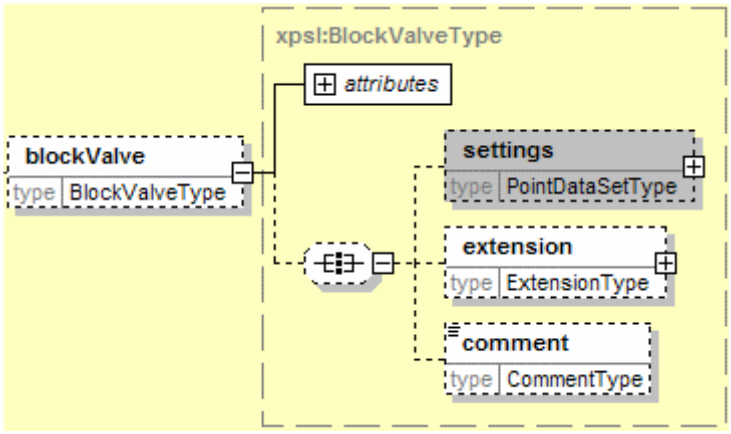
4.8.2 The settingsSet element

A Settings Set is basically a collection of one or more pointSettings elements. The pointSettings element is a set of data for a specific pipeline location, device, or device end. A settingsSet may be used to represent the complete state of a pipeline at some point in time. It may also be used to represent any subset of the pipeline's state.

Each pointSettings element represents the state of a single pipeline device or device end.



The pointSettings element is of type PointDataSetType. This is the same type that was used for the device settings, upSettings, and downSettings elements as shown in the following figure:



The PointDataSetType is described in the following section.

4.9 The PointDataSetType and the Settings Elements

The PointDataSetType is used as an all-encompassing representation of the data at a specific point in a pipeline configuration.

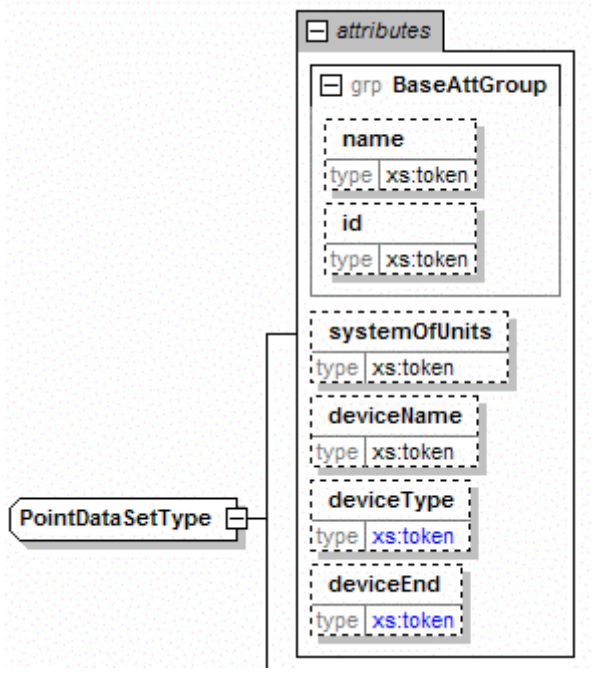
In a device configuration, it is the type of the following elements

- settings element
- upSettings element
- downSettings element

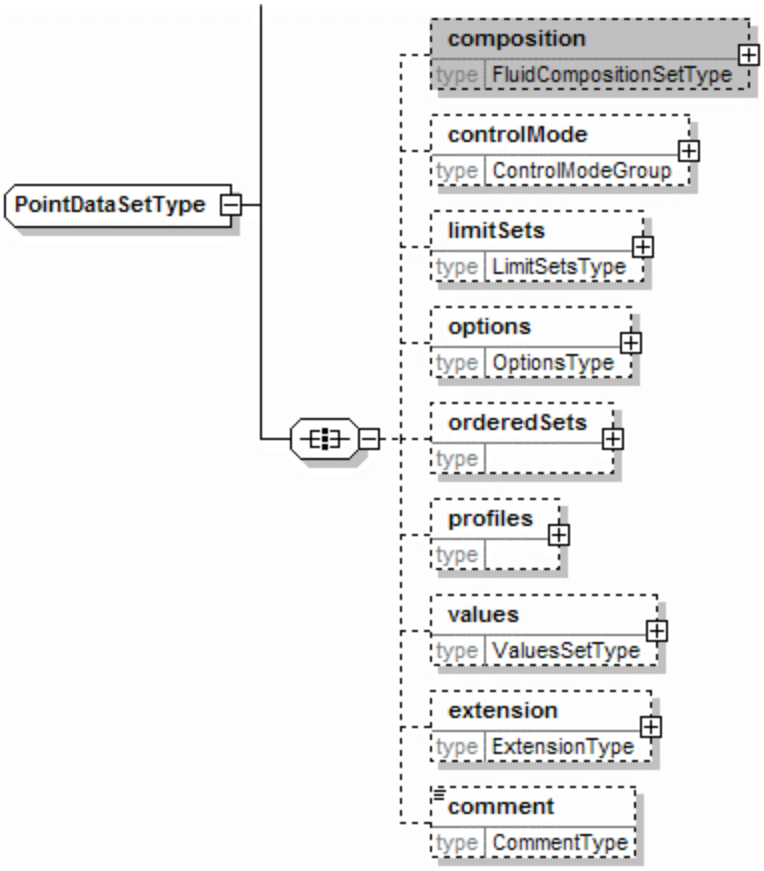
In settingsSets, it is the type of the following element

- pointSettings element

Elements of the PointDataSetType have attributes as shown in the following figure. Of most interest are the deviceName, deviceType, and deviceEnd attributes. These are used to specify which device (and device end where applicable) the data is associated with.



Elements of the PointDataSetType have child elements as shown in the following figure:



4.9.1 The composition Element

The composition Element is used to specify a fluid composition. It has the following child elements:

a. methaneMF
b. ethaneMF
c. propaneMF
d. iButaneMF
e. nButaneMF
f. iPentaneMF
g. nPentaneMF
h. nHexaneMF
i. nHeptaneMF
j. nOctaneMF
k. nNonaneMF
l. nDecaneMF
m. argonMF
n. carbonDioxideMF
o. carbonMonoxideMF
p. heliumMF

q. hydrogenMF
r. hydrogenSulfideMF
s. nitrogenMF
t. oxygenMF
u. waterMF

An example composition element is:

```
<composition>
  <methaneMF>95.0</methaneMF>
  <ethaneMF>5.0</ethaneMF>
</composition>
```

4.9.2 The controlMode Element

The controlMode element is used to specify the mode being used to control a device. Its content is one or more of its child elements with no further detail. The presence of a child element specifies that that mode is controlling the device:

The controlMode element may have any of the following child elements defined with empty content:

a. compressionRatio
b. downPressure
c. downTemperature
d. energyFlow
e. exhaustGasTemperature
f. flow
g. heatRate
h. massFlow
i. openFraction
j. power
k. pressure
l. speed
m. temperature
n. thermalFlow
o. upPressure
p. upTemperature

The following example would specify flow control:

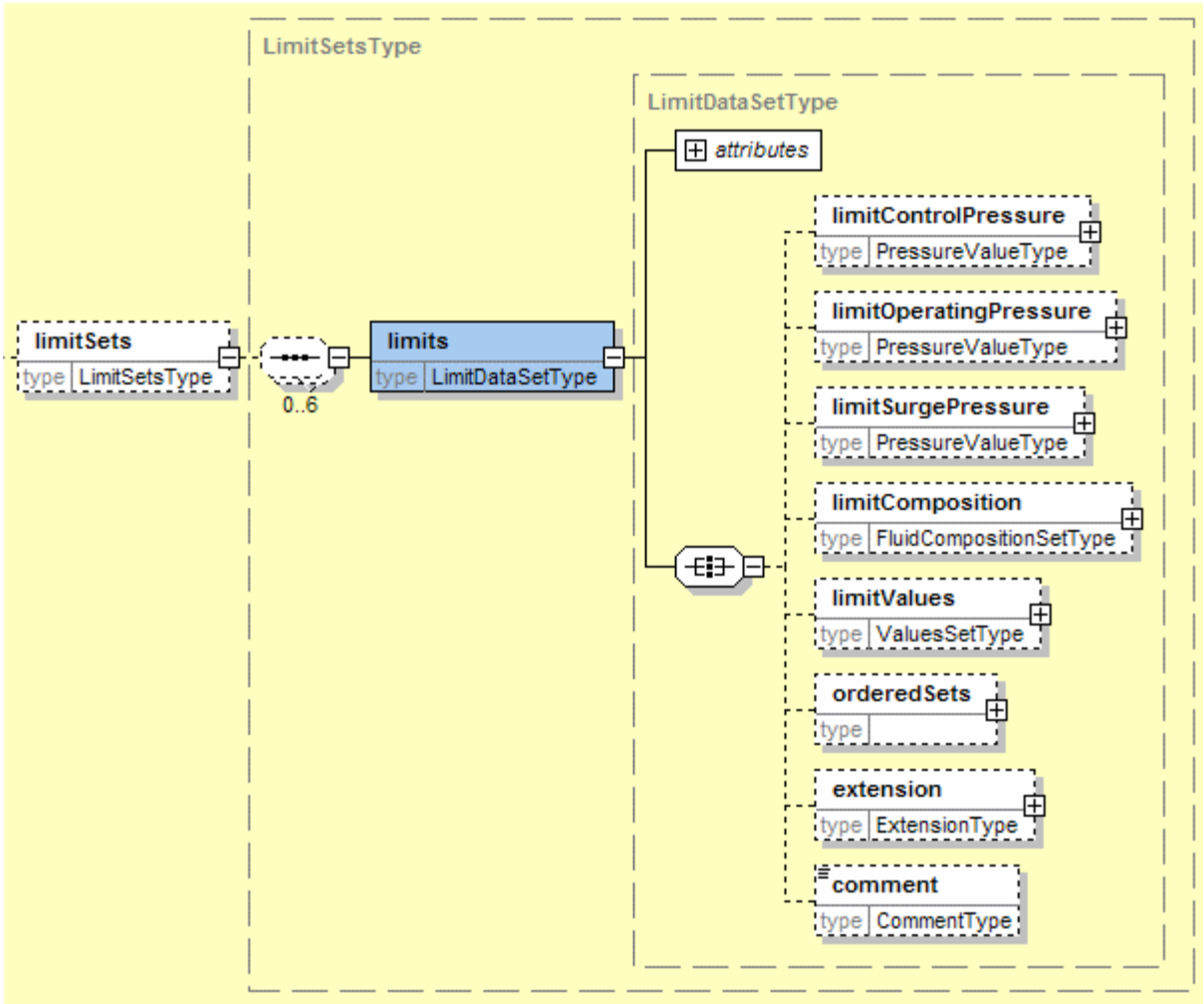
```
<controlMode><flow/></controlMode>
```

4.9.3 The limitSets Element

The limitSets element is a collection of up to six limits elements. Each limits element defines one type of limit. The type of limit being defined is specified using the limitType attribute of the limits element. The limitType attribute is an enumeration with the following possible values:

- Maximum
- Minimum
- HIHI
- HI
- LO
- LOLO

Each limits element consists of a set of data associate with that limitSet type. At most, only one limits element should be defined for each limitsType value. The limits element has the child elements shown in the following figure:



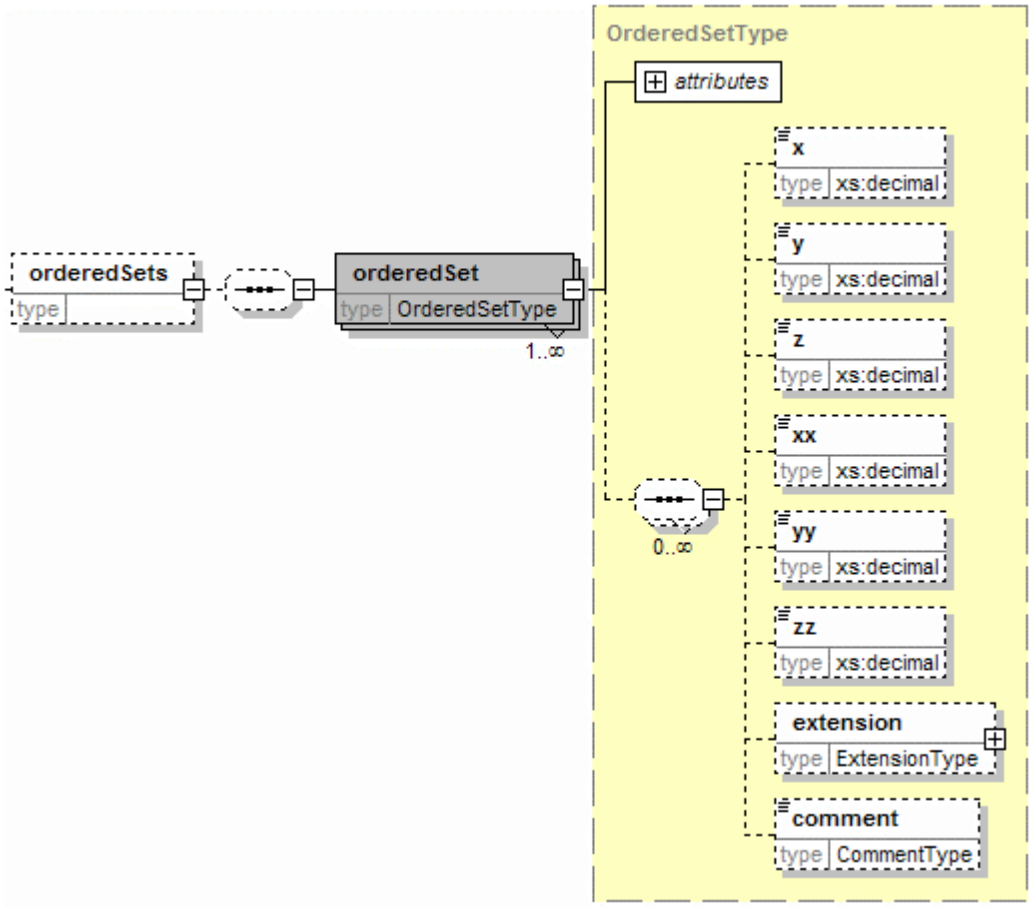
4.9.4 The options Element

The options element may be specified for a device. This element overrides the value of the options element specified for the XPSL instance as a whole. The options element is identical in content to the options element previously described in section 4.5.

4.9.5 The orderedSets Element

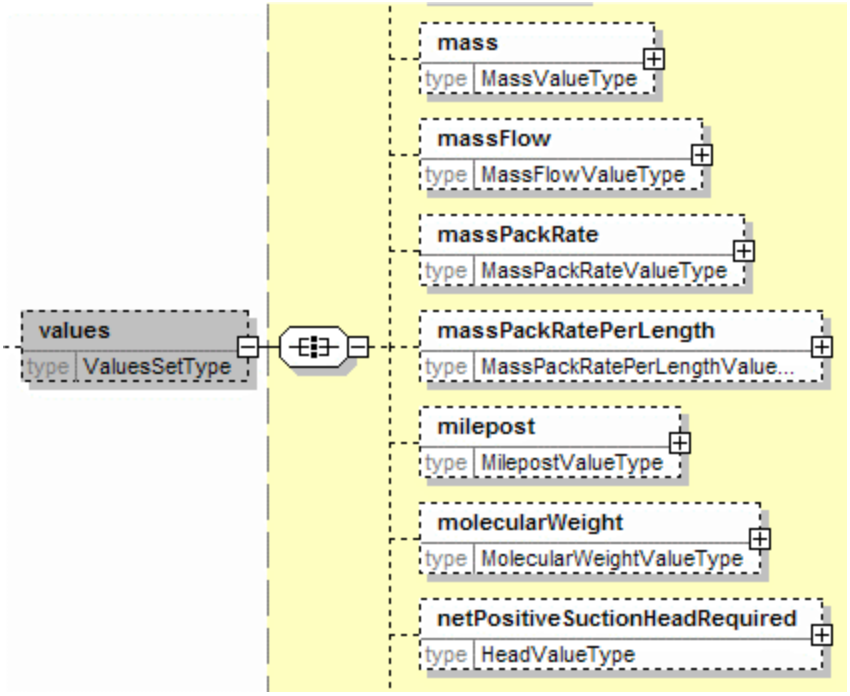
The orderedSets Element provides for an unbounded number of child orderedSet elements.

Each orderedSet element allows an unbounded number of child n-tuple elements where n is less than or equal to six. The orderedSet is may be used to define compressor and pump maps. These elements are illustrated in the following figure:



4.9.6 The values Element

The values Element is an all-encompassing scalar numeric data type element. The goal was to define all possible pipeline numeric data values as specific child elements of this values Element. A small portion of this element's content is shown in the following figure:

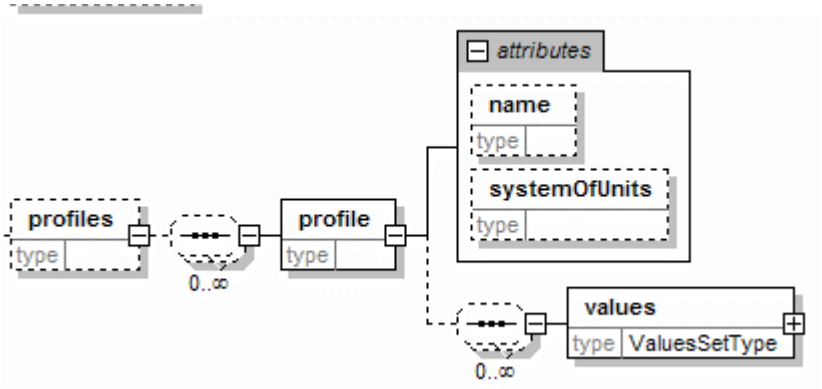


The complete list of child elements follows:

actualDensity actualFlow actualSpecificGravity actualVolume ambientTemperature angularVelocity apiGravity barometricPressure clearanceVolume closedValveCg closedValveCv closingTransitTime cost compressionRatio costPerTime costPerVolume crossectionalArea density dewPointTemperature differentialPressure differentialTemperature displacementVolume distanceStep draConcentration dynamicViscosity elevation efficiency exhaustTemperature flow fluidBulkModulus fluidCoefficientOfThermalExpansion fluidLevel frictionFactor head headGradient heatingValue heatingValuePerMass heatTransferCoefficient heatRate internalDiameter isentropicEfficiency kinematicViscosity length linePack loadStep mass massFlow massPackRate massPackRatePerLength milepost molecularWeight netPositiveSuctionHeadRequired openingTransitTime openValveCg openValveCv outsideDiameter packRate packRatePerLength pipeRoughness power pressure pressurePack relativeHumidity relativeRoughness reynoldsNumber sedimentAndWaterContent shaftPower specificEnthalpy specificGravity specificHeatCapacity specificHeatCapacityPerMole slackVolume standardPressure standardTemperature temperature temperaturePack tensileStrength thermalConductivity thermalEfficiency thermalExpansionCoefficient thermalFlow thermalFlowPerLength thermalFlowPerVolume timeStep torque valveCg valveCv valveOpenFraction vaporPressure velocity volume volumeCorrectionFactor volumetricEfficiency wallThickness waterContent

4.9.7 The profiles Element

The profiles element is used to specify an input or output profile. A profile is defined as a milePost array and a set of values associated with each milePost. The profile element content is shown in the following figure:



Since the values element has a child element milePost as well as every other XPSL numeric data value, the profile may include a wide range of data values at each milepost.