AIMMS Language Reference - Reading and Writing XML Data

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

Email: info@aimms.com
WWW: www.aimms.com

Chapter 30

Reading and Writing XML Data

This chapter

The Extensible Markup Language (XML) is a universal format for exchanging structured documents and data on the web. For those unfamiliar with XML, Section 30.1 provides a short introduction. It is taken literally from http://www.w3.org/XML/1999/XML-in-10-points, and is copyrighted © 1999–2000 by the W3C organization. Sections 30.2 onwards explain in detail how AIMMS lets you employ the XML data format from within your AIMMS applications.

Further information about XML

If you are unfamiliar with XML, the explanation given here will probably not be sufficient. The best starting point to obtain further information about XML, as well as references to specific XML specifications, formats and tools is the W3C XML site http://www.w3.org/XML/.

30.1 XML in 10 points

XML is for structuring data

Structured data includes things like spreadsheets, address books, configuration parameters, financial transactions, and technical drawings. XML is a set of rules (you may also think of them as guidelines or conventions) for designing text formats that let you structure your data. XML is not a programming language, and you don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML avoids common pitfalls in language design: it is extensible, platform-independent, and it supports internationalization and localization. XML is fully Unicode-compliant.

XML looks a bit like HTML

Like HTML, XML makes use of tags (words bracketed by '<' and '>') and attributes (of the form `name="value"`). While HTML specifies what each tag and attribute means, and often how the text between them will look in a browser, XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. In other words, if you see "<p>" in an XML file, do not assume it is a paragraph. Depending on the context, it may be a price, a parameter, a person, a p... (and who says it has to be a word with a "p"?).

XML is text, but isn't meant to be read

Programs that produce spreadsheets, address books, and other structured data often store that data on disk, using either a binary or text format. One advantage of a text format is that it allows people, if necessary, to look at the data without the program that produced it; in a pinch, you can read a text format with your favorite text editor. Text formats also allow developers to more easily debug applications. Like HTML, XML files are text files that people shouldn't have to read, but may when the need arises. Less like HTML, the rules for XML files are strict. A forgotten tag, or an attribute without quotes makes an XML file unusable, while in HTML such practice is tolerated and is often explicitly allowed. The official XML specification forbids applications from trying to second-guess the creator of a broken XML file; if the file is broken, an application has to stop right there and report an error.

XML is verbose by design

Since XML is a text format and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the designers of XML. The advantages of a text format are evident (see point 3), and the disadvantages can usually be compensated at a different level. Disk space is less expensive than it used to be, and compression programs like zip and gzip can compress files very well and very fast. In addition, communication protocols such as modem protocols and HTTP/1.1, the core protocol of the Web, can compress data on the fly, saving bandwidth as effectively as a binary format.

XML is a family of technologies

XML 1.0 is the specification that defines what "tags" and "attributes" are. Beyond XML 1.0, "the XML family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks. Xlink describes a standard way to add hyperlinks to an XML file. XPointer and XFragments are syntaxes in development for pointing to parts of an XML document. An XPointer is a bit like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file. CSS, the style sheet language, is applicable to XML as it is to HTML. XSL is the advanced language for expressing style sheets. It is based on XSLT, a transformation language used for rearranging, adding and deleting tags and attributes. The DOM is a standard set of function calls for manipulating XML (and HTML) files from a programming language. XML Schemas 1 and 2 help developers to precisely define the structures of their own XML-based formats. There are several more modules and tools available or under development. Keep an eye on W3C's technical reports page.

XML is new, but not that new

Development of XML started in 1996 and has been a W3C Recommendation since February 1998, which may make you suspect that this is rather immature technology. In fact, the technology isn't very new. Before XML there was SGML, developed in the early '80s, an ISO standard since 1986, and widely used for large documentation projects. The development of HTML started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience

with HTML, and produced something that is no less powerful than SGML, and vastly more regular and simple to use. Some evolutions, however, are hard to distinguish from revolutions... And it must be said that while SGML is mostly used for technical documentation and much less for other kinds of data, with XML it is exactly the opposite.

XML leads HTML to XHTML

There is an important XML application that is a document format: W3C's XHTML, the successor to HTML. XHTML has many of the same elements as HTML. The syntax has been changed slightly to conform to the rules of XML. A document that is "XML-based" inherits the syntax from XML and restricts it in certain ways (e.g, XHTML allows "<p>", but not "<r>"); it also adds meaning to that syntax (XHTML says that "<p>" stands for "paragraph", and not for "price", "person", or anything else).

XML is modular

XML allows you to define a new document format by combining and reusing other formats. Since two formats developed independently may have elements or attributes with the same name, care must be taken when combining those formats (does "<p>" mean "paragraph" from this format or "person" from that one?). To eliminate name confusion when combining formats, XML provides a namespace mechanism. XSL and RDF are good examples of XML-based formats that use namespaces. XML Schema is designed to mirror this support for modularity at the level of defining XML document structures, by making it easy to combine two schemas to produce a third which covers a merged document structure.

XML is the basis for RDF and the Semantic Web

W3C's Resource Description Framework (RDF) is an XML text format that supports resource description and metadata applications, such as music playlists, photo collections, and bibliographies. For example, RDF might let you identify people in a Web photo album using information from a personal contact list; then your mail client could automatically start a message to those people stating that their photos are on the Web. Just as HTML integrated documents, menu systems, and forms applications to launch the original Web, RDF integrates applications and agents into one Semantic Web. Just like people need to have agreement on the meanings of the words they employ in their communication, computers need mechanisms for agreeing on the meanings of terms in order to communicate effectively. Formal descriptions of terms in a certain area (shopping or manufacturing, for example) are called ontologies and are a necessary part of the Semantic Web. RDF, ontologies, and the representation of meaning so that computers can help people do work are all topics of the Semantic Web Activity.

XML is license-free, platform-independent and well-supported

By choosing XML as the basis for a project, you gain access to a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs and procedures that manipulate it, and there are many tools available and many people who can help you. And since XML is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor. XML isn't always the best solution, but it is always worth considering.

30.2 Introduction to XML support in AIMMS

XML support in AIMMS

In order to help you understand the XML support available within AIMMS to its full extent, this section provides an explanation of the basic concepts used in XML. If you are already familiar with XML and XML schemas, the material in this section may help you to understand how the XML concepts you are already familiar with are used by the XML facilities in AIMMS.

The XML data format

The XML data format is a text format built around just two syntactical components, *elements* and *attributes*. Because the semantics of these components are not fixed and can be user-defined, the XML data format can be used to represent virtually any meaningful concept.

XML elements

XML elements are denoted by a start tag (a word identifying the type of element enclosed by the "<" and ">" characters) and an end tag (the same element type enclosed by the "</" and ">" characters). Elements delimit the piece of XML data between its start and end tag. Elements may hold only text or numeric data, or they can provide depth to XML data, since the enclosed XML data may contain other XML elements. An element in a stream of XML data is, in fact, a node in the tree associated with the entire stream of XML data. The root node of this tree corresponds to the *obligatory and unique* root element of the XML data stream.

XML elements without content

If an element does not contain any enclosed XML content, it is possible to omit the end tag alltogether, and enclose the start tag between "<" and "/>" characters. This format is commonly used if the element contains only attributes.

XML attributes

XML attributes provide additional information about a particular element in an XML data stream. Attributes are specified by the form `name="value"` between the "<" and ">" (or "/>") characters of the start tag of the element in question.

The Data Reconciliation example

All examples in this chapter make use of a single AIMMS project that comes with the AIMMS system, the *Data Reconciliation* project. In this example, flows between units in a chemical plant, as well the chemical composition of these flows, are measured. Unfortunately, such measurements might not be internally consistent despite, for instance, a physical requirement that the sum of all flows into any unit be equal to the sum of all flows out of that unit (i.e. no material is created or lost within a unit). Due to the inaccuracy of the measurement devices (or, even worse, broken measurement devices), the measured values do not necessarily satisfy such balances. The objective of the model is to find a set of flow values and compositions which *are* internally consistent, and lie as close as possible to the corresponding measured values. Such consistent values are called *reconciled* values. Within the model the measured and reconciled values are stored in the identifiers

- `MeasuredFlow(f)`,
- `MeasuredComposition(f,c)`,
- `Flow(f)`, and
- `Composition(f,c)`,

where `f` is an index into a set of `Flows`, and `c` is an index into a set of `Components`. Table 30.1 contains the measured and reconciled flow values and compositions used throughout this chapter.

Flow name	Measured values				
	Flow value [ton/h]	Flow composition [%]			
		N_2	H_2	NH_3	Ar
Inflow	111.98	26.96	72.71		0.33
Mix		24.56			4.91
NH3-Mix		19.99			
NH3-Flow	105.59				
Residue			69.68		
Ar-Flow					
Feedback	358.00				

Flow name	Reconciled values				
	Flow value [ton/h]	Flow composition [%]			
		N_2	H_2	NH_3	Ar
Inflow	117.03	26.96	72.71	0.00	0.33
Mix	475.03	23.95	71.08	0.05	4.91
NH3-Mix	475.03	19.99	59.08	15.27	5.66
NH3-Flow	105.59	0.00	0.00	100.00	0.00
Residue	369.44	23.57	69.68	0.07	6.67
Ar-Flow	11.44	89.19	0.00	0.00	10.81
Feedback	358.00	22.82	70.48	0.07	6.62

Table 30.1: Measured and reconciled values

XML data example

The XML fragment below illustrates a possible XML format to store the measured and reconciled flows and compositions. The root element `FlowMeasurementData` has a `date` attribute to indicate the date of the measurements. The root element contains one or more `Flow` elements which contain the measured (if any) and reconciled flow values for all the flows in the network. Each `Flow` element contains a single `Composition` element, which, in turn, contains one or more `Component` elements with the measured (if any) and reconciled composition values for each component of the flow in question.

```
<FlowMeasurementData xmlns="http://www.aimms.com/Reconciliation" date="2001-10-01">
    <Flow name="Inflow" measured="111.98" reconciled="117.03">
        <Composition>
            <Component name="N2" measured="26.96" reconciled="26.96"/>
            <Component name="H2" measured="72.71" reconciled="72.71"/>
            <Component name="Ar" measured="0.33" reconciled="0.33"/>
        </Composition>
    </Flow>
    <Flow name="Mix" reconciled="475.03">
        <Composition>
            <Component name="N2" measured="24.56" reconciled="23.95"/>
            <Component name="H2" reconciled="71.08"/>
            <Component name="NH3" reconciled="0.05"/>
            <Component name="Ar" measured="4.91" reconciled="4.91"/>
        </Composition>
    </Flow>
    <Flow name="NH3-Mix" reconciled="475.03">
        <Composition>
            <Component name="N2" measured="19.99" reconciled="19.99"/>
            <Component name="H2" reconciled="59.08"/>
            <Component name="NH3" reconciled="15.27"/>
            <Component name="Ar" reconciled="5.66"/>
        </Composition>
    </Flow>
    <Flow name="NH3-Flow" measured="105.59" reconciled="105.59">
        <Composition>
            <Component name="NH3" reconciled="100.00"/>
        </Composition>
    </Flow>
    <Flow name="Residue" reconciled="369.44">
        <Composition>
            <Component name="N2" reconciled="23.57"/>
            <Component name="H2" measured="69.68" reconciled="69.68"/>
            <Component name="NH3" reconciled="0.07"/>
            <Component name="Ar" reconciled="6.67"/>
        </Composition>
    </Flow>
    <Flow name="Ar-Flow" reconciled="11.44">
        <Composition>
            <Component name="N2" reconciled="89.19"/>
            <Component name="Ar" reconciled="10.81"/>
        </Composition>
    </Flow>
    <Flow name="Feedback" measured="358.00" reconciled="358.00">
        <Composition>
            <Component name="N2" reconciled="22.82"/>
            <Component name="H2" reconciled="70.48"/>
            <Component name="NH3" reconciled="0.07"/>
            <Component name="Ar" reconciled="6.62"/>
```

```
        </Composition>
    </Flow>
</FlowMeasurementData>
```

Not unique

The XML data format illustrated above is not unique. For instance, the measured and reconciled values could have been represented by child elements of the Flow and Component elements instead of by attributes. Thus, a different, but equally valid, XML representation of the same data is illustrated in the XML data snippet below.

```
<Flow name="Inflow">
    <MeasuredValue>111.984</MeasuredValue>
    <ReconciledValue>117.034</ReconciledValue>
    <Composition>
        <Component name="N2">
            <MeasuredValue>26.960</MeasuredValue>
            <ReconciledValue>26.960</ReconciledValue>
        </Component>
        ...
    </Composition>
</Flow>
```

The particular XML data format chosen may be a matter of taste, or the result of a formal agreement between several parties who wish to use the corresponding XML data.

XML schema

To support you in defining a particular XML data format in a formal manner, XML provides an XML-based standard to specify such definitions. This standard is called *XML Schema*. It allows you, among other things, to specify

- the allowed (tree) structure of a particular XML data format in terms of all possible elements and their child elements,
- the minimum and maximum number of times a particular element can occur,
- which attributes are supported by particular elements,
- whether attributes are optional or required, and
- the intended data types of elements and attributes in your XML data format.

To create an XML schema file that matches an intended XML data format, it is best to use one of the tools available for this purpose. For more detailed information about XML schema, as well as the tools available for creating an XML schema file, refer to http://www.w3.org/XML/Schema

XML schema example

The following XML schema definition, formally defines the XML data format as used in the XML data example above.

```
<xs:schema targetNamespace="http://www.aimms.com/Reconciliation"
           xmlns="http://www.aimms.com/Reconciliation"
```

```
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="FlowMeasurementData">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Flow" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Composition" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Component" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="measured" type="xs:double" use="optional"/>
                        <xs:attribute name="reconciled" type="xs:double" use="optional"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="measured" type="xs:double" use="optional"/>
            <xs:attribute name="reconciled" type="xs:double" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="date" type="xs:date" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Schema namespaces

An XML schema definition can specify a namespace by which the schema is to be known. In the example above, the `targetNamespace` attribute of the `xs:schema` element specifies that the schema that follows defines the namespace `http://www.aimms.com/XMLSchema/ReconciliationExample`. In the XML data example listed earlier in this section, the `xmlns` attribute of the root element specifies that all element and attribute names underneath the root element are to be interpreted in the context of that namespace.

Two modes of XML support

AIMMS allows you to read and write XML data from within your model in two modes:

- it lets AIMMS generate and read XML data based on identifier definitions in your model, or
- it lets AIMMS generate and read XML data according to a given XML schema specification.

AIMMS-generated XML

In the first mode, AIMMS will generate and read XML for the subset of identifiers that you specify. The format of the generated XML closely resembles the declaration of the identifiers in your model, generates XML data for one identifier at a time, and adds a tree level for each dimension. Letting AIMMS generate XML data for your model is the fastest way of getting XML data that corresponds to your model, but

- gives you little control over the final result, and
- programs that use the generated XML data have to adhere to the generated format.

User-defined XML

In the second mode, AIMMS assumes that you already have a XML schema file that specifies the precise XML data format that you want to generate from within AIMMS, or want to read from an external XML data file. AIMMS provides a tool to let you map the elements and attributes in the XML schema onto sets and multidimensional identifiers in your model. Based on this mapping, and the data in your model, you can let AIMMS generate XML data according to the specified schema, or let AIMMS fill the corresponding identifiers according to an XML data file in the specified format.

30.3 Reading and writing AIMMS-generated XML data

Obtaining generated XML output

Through the functions

- `GenerateXML`(*XMLFile*,*IdentifierSet*[,*merge*][,*SchemaFile*])
- `ReadGeneratedXML`(*XMLFile*[,*merge*])

AIMMS will generate XML output associated with one or more identifiers in your model, or read AIMMS-generated XML from a file.

The function `GenerateXML`

Using the function `GenerateXML`, you can let AIMMS write XML data to the file *XMLFile*. AIMMS will generate XML data for all the identifiers in the *IdentifierSet* argument, which must be a subset of the predefined set `AllIdentifiers`. With the optional *merge* argument (default 0) you can indicate whether you want to merge the generated XML data in another XML document, in which case AIMMS will omit the XML header from the generated XML file. This allows you to merge the contents of the generated file into another XML file. Note that setting the *merge* argument to 1 does not mean that the generated XML data will be appended to the specified file, its contents are always completely overwritten. If you specify a *SchemaFile* name, AIMMS will also generate an XML schema file with the specified file name, matching the generated XML data. All data in the XML file is represented in terms of the currently active unit convention (see also Section 32.8). The function will return 1 if successful, or 0 if not.

The function ReadGeneratedXML

With the function ReadGeneratedXML you can read back AIMMS-generated XML data from the specified *XMLFile*. With the optional *merge* argument (default 0), you can choose whether you want to merge the data included in the XML file with the existing data, or overwrite any existing data (default). All data in the XML file will be interpreted in accordance with the currently active unit convention (see also Section 32.8). The function will return 1 if successful, or 0 if not.

Fixed format

The XML data format generated by the function GenerateXML solely depends on the declaration of the identifiers for which you want the XML data to be generated. Thus, you can use the generated XML data simply to store some model data in an XML file, ready to be read back into AIMMS through the function ReadGeneratedXML, or by any other program that adheres to the XML data format as generated by AIMMS.

Example

A call to the function GenerateXML, for the identifiers listed in Section 30.2, will result in the following XML data being generated.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AimmsData>
    <Flows>
        <Flows elem="Inflow"/>
        <Flows elem="Mix"/>
        <Flows elem="NH3-Mix"/>
        <Flows elem="NH3-Flow"/>
        <Flows elem="Residue"/>
        <Flows elem="Ar-Flow"/>
        <Flows elem="Feedback"/>
    </Flows>
    <MeasuredFlow>
        <f elem="Inflow" value="111.98"/>
        <f elem="NH3-Flow" value="105.59"/>
        <f elem="Feedback" value="358.00"/>
    </MeasuredFlow>
    <Flow>
        <f elem="Inflow" value="117.03"/>
        <f elem="Mix" value="475.03"/>
        <f elem="NH3-Mix" value="475.03"/>
        <f elem="NH3-Flow" value="105.59"/>
        <f elem="Residue" value="369.44"/>
        <f elem="Ar-Flow" value="11.44"/>
        <f elem="Feedback" value="358.00"/>
    </Flow>
    <MeasuredComposition>
        <nmf elem="Inflow">
            <c elem="N2" value="26.96"/>
            <c elem="H2" value="72.71"/>
            <c elem="Ar" value="0.33"/>
        </nmf>
        <nmf elem="Mix">
            <c elem="N2" value="24.56"/>
            <c elem="Ar" value="4.91"/>
        </nmf>
        <nmf elem="NH3-Mix">
```

```
            <c elem="N2" value="19.99"/>
        </nmf>
        <nmf elem="Residue">
            <c elem="H2" value="69.68"/>
        </nmf>
    </MeasuredComposition>
    <Composition>
        <nmf elem="Inflow">
            <c elem="N2" value="26.96"/>
            <c elem="H2" value="72.71"/>
            <c elem="Ar" value="0.33"/>
        </nmf>
        <nmf elem="Mix">
            <c elem="N2" value="23.95"/>
            <c elem="H2" value="71.08"/>
            <c elem="NH3" value="0.05"/>
            <c elem="Ar" value="4.91"/>
        </nmf>
        <nmf elem="NH3-Mix">
            <c elem="N2" value="19.99"/>
            <c elem="H2" value="59.08"/>
            <c elem="NH3" value="15.27"/>
            <c elem="Ar" value="5.66"/>
        </nmf>
        <nmf elem="NH3-Flow">
            <c elem="NH3" value="100.00"/>
        </nmf>
        <nmf elem="Residue">
            <c elem="N2" value="23.57"/>
            <c elem="H2" value="69.68"/>
            <c elem="NH3" value="0.07"/>
            <c elem="Ar" value="6.67"/>
        </nmf>
        <nmf elem="Ar-Flow">
            <c elem="N2" value="89.19"/>
            <c elem="Ar" value="10.81"/>
        </nmf>
        <nmf elem="Feedback">
            <c elem="N2" value="22.82"/>
            <c elem="H2" value="70.48"/>
            <c elem="NH3" value="0.07"/>
            <c elem="Ar" value="6.62"/>
        </nmf>
    </Composition>
</AimmsData>
```

Numeric width and precision

Using the AIMMS options XML_number_width and XML_number_precision you can specify the print width and precision of any numerical data generated through the function GenerateXML. The rules are as follows.

- If the option XML_number_width is set to -1, AIMMS will always use scientific format with precision XML_number_precision.
- If the option XML_number_width is 0, AIMMS will print a fixed point floating point number with precision XML_number_precision, provided the number can be represented exactly with the given precision, otherwise scientific format will be used.

- If the option XML_number_width is greater than 0, AIMMS will print a fixed point floating point number with precision XML_number_precision, provided the number to be printed fits within the specified width, otherwise scientific format will be used.

30.4 Reading and writing user-defined XML data

Writing user-defined XML

If you already have a given XML data format to which you want your AIMMS application to adhere, the simple XML generation functions discussed in the previous section will not work. This section discusses the tools and functions provided by AIMMS to help you read and write XML data in a given format, on the explicit assumption that your XML data format is formally described through an XML schema file. If you do not have a schema file for your XML data format, you are advised to use one of the XML schema editors available on the market to construct an XML schema file corresponding to your XML data format.

Mapping XML schema to AIMMS identifiers

Once you do have an XML schema file corresponding to your XML data format, you must create a mapping between the tree structure described by the XML schema, and the identifiers in your AIMMS model that will hold the corresponding data. This mapping is described using another XML format (as illustrated later in this section), which, in principle, can be edited manually. However, to create such a mapping, you can also use the **Tools-XML Schema Mapping** menu. This will ask you to select an XML schema file (with an .xsd extension), and will open the **XML Schema Mapping** dialog box for the corresponding schema, as illustrated in Figure 30.1. If there is already an AIMMS

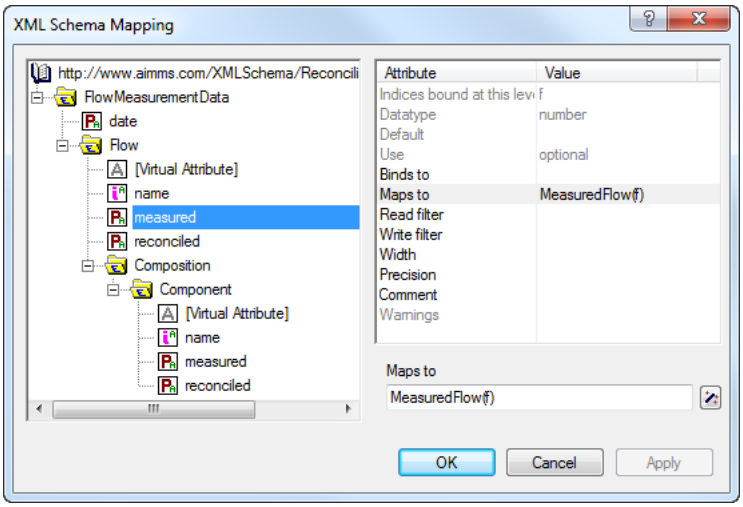

Figure 30.1: The **XML Schema Mapping** dialog box

XML Mapping file (with an .axm extension) corresponding to the XML schema file, AIMMS will also read the information in that file, and adapt the attributes of the nodes in the XML mapping tree accordingly.

Binding indices

The mapping between an XML schema and AIMMS is based on the principle that any element that occurs multiple times in an XML data stream can be associated with an index in your AIMMS model. If an index is bound for a particular element, it is also considered to be bound for all attributes and child elements of the element at hand. These can then be mapped onto multidimensional AIMMS identifiers defined over all indices bound at a particular level in the XML tree.

Index value

The element values of such an index associated with an element occurring multiple times in the XML tree can come from several sources:

- a *required* (i.e. non-optional) attribute of the element,
- the data of a direct data-only child element (i.e. without any child elements of its own) which occurs *exactly* once, or
- if there is no such attribute or child element, an element name generated by AIMMS as if there were an attribute containing the generated name.

Examples

In the example of Section 30.2, the index f is bound to the element Flow through the value of its attribute name.

```
<Flow name="Inflow" measured="111.98" reconciled="117.03">
    ...
</Flow>
```

Equally, the index f, associated with the Flow element, could have obtained its value from the data-only child element FlowName, as illustrated below.

```
<Flow>
   <FlowName>Inflow</FlowName>
   <MeasuredValue>111.98</MeasuredValue>
   <ReconciledValue>117.03</ReconciledValue>
    ...
</Flow>
```

Consider the following XML logging format.

```
<LogEntries>
   <Log date="2008-03-31 12:07:31" severity="error">No value for 'Inflow'</Log>
   <Log date="2008-03-31 12:07:35" severity="warning">Error for 'Inflow'</Log>
</LogEntries>
```

None of the attributes nor the element value can uniquely identify a Log element. Rather the LogEntries element contains a sequence of Log entries. The log date, severity and message can perfectly be stored in parameters LogDate(l), LogSeverity(l) and LogMessage(l), where the values of index l into

the set LogEntries are numbered elements generated by AIMMS when reading the XML file, and ignored when writing.

Mapped data

In addition to binding indices, the values of attributes and data-only elements can also be mapped to multidimensional identifiers in your model. Such multidimensional identifiers can be defined over a subset of, or all, indices bound at the level of the attribute or element to be mapped. Attributes mapped to multidimensional identifiers may be optional, corresponding to the mapped identifier holding a default value in the AIMMS model.

Example

In the example above, the attributes measured and reconciled are mapped to the multidimensional identifiers MeasuredFlow(f) and Flow(f), respectively. This is a valid mapping since the index f is bound at the level of the element Flow and hence also to all its child attributes and elements. Similarly, the identifiers MeasuredFlow(f) and Flow(f) could have been mapped to the data-only elements MeasuredValue and ReconciledValue in the second part of the example above.

Mapping tree nodes

The **XML Schema Mapping** dialog box displays an XML mapping tree based on the information available in the schema file. The XML mapping tree consists of the following components.

- A single root node AimmsXMLSchemaMapping, which contains the single ElementMapping node for the root element defined in the XML schema to be mapped. In the **XML Schema Mapping** tree, the AimmsXMLSchemaMapping node is displayed by the icon.
- ElementMapping nodes, each of which will have zero or more AttributeMapping, VirtualAttributeMapping and ElementMapping child nodes. In the **XML Schema Mapping** tree, a data-only ElementMapping node is displayed by either the icon, or the icon when a data-only element is bound to an index, or the icon when a data-only element is mapped to multidimensional data. ElementMapping nodes with children are displayed by the icon.
- AttributeMapping nodes, which do not have child nodes. In the tree, an AttributeMapping node is displayed by either the icon, or the icon when the attribute is bound to an index, or the icon when the attribute is mapped to multidimensional data.
- VirtualAttributeMapping nodes, which basically behave like AttributeMapping nodes, but have no counterpart in the XML schema. In the tree, a VirtualAttributeMapping node is displayed by either the icon, or the icon when the virtual attribute is bound to an index. VirtualAttributeMapping nodes are automatically inserted by AIMMS underneath elements that can occur multiple times, and can only be bound to an index. They can be used to let AIMMS generate element names for an index that cannot be bound to a real attribute or child element.

Mapping attributes

To each node type in the mapping tree, a number of possibly node-specific attributes are associated. Some of these attributes are based on the information in the schema file and cannot be edited, while others define the actual mapping between the XML schema and identifiers in your model, and can naturally be edited. When creating a mapping tree, AIMMS will look for an .axm file corresponding to the schema file you selected, and read the actual mapping attributes from the mapping file.

AimmsXMLSchemaMapping attributes

The AimmsXMLSchemaMapping node supports the attributes listed in Table 30.2.

Attribute	Use	Editable	Stored	Value-type
MappedNameSpace	info	no	yes	*namespace-URI*
AimmsModel	optional	yes	yes	*string*
default-width	optional	yes	yes	*integer*
default-precision	optional	yes	yes	*integer*
comment	optional	yes	yes	*string*

Table 30.2: AimmsXMLSchemaMapping attributes

The MappedNameSpace attribute

The MappedNameSpace attribute contains the namespace URI (Universal Resource Identifier) by which the XML schema is identified. AIMMS will retrieve it from the XML schema, whenever the schema contains a namespace URI, or will generate an artificial namespace URI "http://tempuri.org/AIMMS/auto-generated-namespace" if the XML schema does not contain this information.

The AimmsModel attribute

Using the AimmsModel attribute you can indicate the AIMMS model for which the mapping is intended. The information you enter here is solely for your own use, and is ignored by AIMMS when reading or writing XML data according to this mapping.

The default-width and -precision attributes

Using the default-width and default-precision attributes you can specify the width and precision with which you want AIMMS to write numerical data, when writing an XML file subject to this mapping. These attributes override the AIMMS options XML_number_width and XML_number_precision discussed in Section 30.3, and use the same semantics.

ElementMapping attributes

An ElementMapping node supports the attributes listed in Table 30.3. Some attributes used in an element mapping do not apply to all ElementMapping nodes. Binding the contents of an element to an index, or mapping the contents to a multidimensional identifier in your model, is only useful if the element is a data-only element, and not when the element contains child elements. When reading an XML schema file, AIMMS distinguishes between these two types of elements, and omits the attributes for mapping data-only elements whenever

Attribute	Use	Data-only	Editable	Stored	Value-type
name	required	no	no	yes	*string*
occurrence	info	yes	no	no	*string*
datatype	info	yes	no	no	*string*
default	info	yes	no	yes	*string*
binds-to	optional	yes	yes	yes	*index-reference*
maps-to	optional	yes	yes	yes	*reference*
width	optional	yes	yes	yes	*integer*
precision	optional	yes	yes	yes	*integer*
read-filter	optional	no	yes	yes	*expression*
write-filter	optional	no	yes	yes	*expression*
comment	optional	no	yes	yes	*string*

Table 30.3: ElementMapping attributes

appropriate. If the schema file indicates that an element can have a mixed content (i.e. both character data and child elements), AIMMS will ignore the character data.

XML schema-based attributes

The occurrence, datatype and default attributes of an ElementMapping node contain information about the element that is obtained from the XML schema. The values of these attributes cannot be edited, and are displayed in the **XML Schema Mapping** dialog box solely for your information.

The occurrence attribute

The occurrence attribute of an element can hold the values optional/once, optional/many, never, once, or many. If you try to bind an index to an optional data-only element, AIMMS will issue a warning, since this can potentially cause problems when reading an XML file.

The datatype attribute

In the datatype attribute, AIMMS displays the datatypes as either unspecified, number, integer, string, or any, whichever is nearest to the datatype of the element specified in the XML schema. You can use this information to determine to which AIMMS identifiers a particular element can be mapped.

The default attribute

In the default attribute, AIMMS displays the default value of a data-only element as specified in the XML schema file (if any). If there is a default value, this information is also stored in the mapping file, as this information is used by AIMMS to interpret the value of non-existent elements when reading an XML file.

The binds-to attribute

With the binds-to attribute you can indicate that AIMMS must bind the contents of a data-only element to a particular index in your model. The value of the binds-to attribute must be a reference to an index in your AIMMS model.

As explained at the start of this section, the binding propagates to the direct parent of the element, and recursively to any of the child attributes and elements of the parent. Those indices that are bound at a particular level of the tree, are displayed in the **XML Schema Mapping** dialog box in the attribute `Indices bound at this level`, which is automatically updated by AIMMS if you change the value of a `binds-to` attribute.

The `maps-to` attribute

With the `maps-to` attribute you can indicate that the contents of a data-only element must be mapped to a multidimensional identifier in your model (including subsets). The value of this attribute must be a reference to an AIMMS identifier in your model, and can refer to the indices that are bound at the level of the `ElementMapping` in question (or a subset thereof). Note that you might obtain unexpected results when reading XML data if the `maps-to` attribute does not refer to all indices bound at this level. If there are multiple instances of the element (corresponding to indices not used in the identifier), only the value of the most recent instance will be registered. The expression that you specify for this attribute can be a slice of a higher-dimensional identifier, and the indices may also be permuted.

`maps-to` in the presence of `binds-to`

Even if you have specified a `binds-to` attribute for a node in the tree, you are also allowed to specify the `maps-to` attribute as well, which will then be used when reading and writing an XML file in the given XML data format. If, in that situation, the `maps-to` attribute contains a reference to a multidimensional identifier, AIMMS will assign a value of 1.0 to that identifier, or if the `maps-to` attribute contains a reference to a, possibly multidimensional, subset, AIMMS will add the corresponding tuple to the subset. When writing an XML file, AIMMS will always write out the element if the identifier contained in the `maps-to` attribute contains non-default data, even if there is no other data to be written that is defined over the index associated with the `binds-to` attribute.

The `read-filter` attribute

Using the `read-filter` attribute you can specify an AIMMS expression to use as a filter when reading an XML data file. The value of the `read-filter` attribute must be a reference to a multidimensional identifier in your model, similar to the `maps-to` attribute, or can be 0 or 1 (the default). If the value is 0, the element and all its child attributes and elements are ignored when reading an XML file. If the value is a reference to an AIMMS identifier, the element, along with its child attributes and elements, is skipped if the identifier at hand does not contain a nonzero value for the index tuple bound at that particular position in the XML file. If the `read-filter` attribute refers to an identifier that is also read from the XML file, AIMMS will use the value for that identifier as contained in the XML file, provided that this value is read before the corresponding reference to the `read-filter` is evaluated.

The write-filter attribute

With the write-filter attribute you can specify an AIMMS expression to use as a filter when writing an XML data file. The value of the write-filter attribute must be a reference to a multidimensional identifier in your model, similar to the maps-to attribute, or can be 0 or 1. If the value is 0, the element and all its child attributes and elements are ignored when writing an XML file. If the value is 1, the element is always written, regardless of whether there are any nondefault data within your model for that particular element. If there is no nondefault data, AIMMS will write the corresponding default value. If the value is a reference to an AIMMS identifier, the element, along with its child attributes and elements, is skipped if the identifier at hand does not contain a nonzero value for the index tuple bound at that particular position in the XML file.

The width and precision attributes

Using the width and precision attributes of a data-only element you can override the values of the default-width and default-precision attributes of the AimmsXMLSchemaMapping node (or, eventually, of the AIMMS options XML_number_width and XML_number_precision) for the element in question. The attributes will only be used if a maps-to attribute has also been specified. With these options you can determine, for each individual element type, how numerical data will be formatted when writing an XML file.

AttributeMapping attributes

AttributeMapping nodes support the attributes listed in Table 30.4.

Attribute	Use	Editable	Stored	Value-type
name	required	no	yes	*string*
datatype	info	no	no	*string*
default	info	no	yes	*string*
use	info	no	no	*namespace-URI*
binds-to	optional	yes	yes	*index-reference*
maps-to	optional	yes	yes	*reference*
width	optional	yes	yes	*integer*
precision	optional	yes	yes	*integer*
read-filter	optional	yes	yes	*expression*
write-filter	optional	yes	yes	*expression*
comment	optional	yes	yes	*string*

Table 30.4: AttributeMapping attributes

The use attribute

The use attribute contains the value of the attribute of the same name obtained from the XML schema, and indicates whether an XML attribute is optional, required or prohibited. If you try to bind an optional attribute to a index in your AIMMS model, AIMMS will issue a warning, since such bindings may

cause problems when reading an XML file in which the optional attribute is not present.

Other attributes similar to element attributes

The remaining attributes of an AttributeMapping node have identical interpretations to those of an ElementMapping node. For information about these attributes refer to the documentation for the corresponding attributes of ElementMapping nodes above.

Example

The following XML data fragment shows the mapping between the XML data file, illustrated in Section 30.2, and the identifiers

- MeasuredFlow(f),
- Flow(f),
- MappedMeasuredComposition(f,c), and
- MappedComposition(f,c)

which contain the corresponding data in the *Data Reconciliation* project.

```
<AimmsXMLSchemaMapping xmlns="http://www.aimms.com/XMLSchema/AimmsXMLMappingSchema"
            MappedNameSpace="http://www.aimms.com/Reconciliation"
            default-width=16 default-precision=2>
  <ElementMapping name="FlowMeasurementData">
    <AttributeMapping name="date" maps-to="ReconciliationDate"/>
    <ElementMapping name="Flow">
      <AttributeMapping name="measured" maps-to="MeasuredFlow(f)"/>
      <AttributeMapping name="name" binds-to="f"/>
      <AttributeMapping name="reconciled" maps-to="Flow(f)"/>
      <ElementMapping name="Composition">
        <ElementMapping name="Component">
          <AttributeMapping name="measured" maps-to="MappedMeasuredComposition(f,c)"/>
          <AttributeMapping name="name" binds-to="c"/>
          <AttributeMapping name="reconciled" maps-to="MappedComposition(f,c)"/>
        </ElementMapping>
      </ElementMapping>
    </ElementMapping>
  </ElementMapping>
</AimmsXMLSchemaMapping>
```

VirtualAttributeMapping attributes

VirtualAttributeMapping nodes support the attributes listed in Table 30.5. The VirtualAttributeMapping allows you to associate an index with an element that occurs multiple times in your XML file, but which has no unique attribute or child element in the XML schema to which you can bind this index. A VirtualAttributeMapping allows you to still associate such elements with an index, as if there were a virtual, hidden attribute to which you bind. When reading an XML file, the element names associated with that index are then generated by AIMMS either numbered on the basis of a given prefix, or by retrieving the names from the element contents itself. When writing an XML file, the element names associated with an index bound to a VirtualAttributeMapping attribute are ignored.

Attribute	Use	Editable	Stored	Value-type
binds-to	required	yes	yes	*index-reference*
maps-to	optional	yes	yes	*reference*
read-filter	optional	yes	yes	*expression*
write-filter	optional	yes	yes	*expression*
assume-element-value	required	yes	yes	Yes / No
element-prefix	optional	yes	yes	*string*
comment	optional	yes	yes	*string*

Table 30.5: VirtualAttributeMapping attributes

VirtualAttributeMapping attributes

The binds-to, maps-to, read-filter and write-filter have the exact same interpretation as for a normal AttributeMapping. Through the assume-element-value attribute you can indicate whether AIMMS should generate element values when reading, or, when the parent element is a data-only element, whether the element content should be taken as the element value for the index. The default value of the assume-element-value attribute is No. Element names generated by AIMMS are numbered starting from 1, with the prefix specified in the element-prefix attribute.

binds-to is mandatory

Note that the binds-to attribute is required for a VirtualAttributeMapping attribute. The VirtualAttributeMapping node and all changes you made to any of its other attributes in the **XML Schema Mapping** dialog box will be ignored when saving the mapping, unless the binds-to attribute has a value.

Example

Consider the XML logging format discussed above

```
<LogEntries>
   <Log date="2008-03-31 12:07:31" severity="error">No value for 'Inflow'</Log>
   <Log date="2008-03-31 12:07:35" severity="warning">Error for 'Inflow'</Log>
</LogEntries>
```

The following schema mapping maps the contents of this XML file to identifiers LogDate(l), LogSeverity(l) and LogMessage(l), where l is an index into a set LogEntries.

```
<AimmsXMLSchemaMapping xmlns="http://www.aimms.com/XMLSchema/AimmsXMLMappingSchema"
            MappedNameSpace="http://www.aimms.com/LoggingData"
            default-width=16 default-precision=2>
  <ElementMapping name="LogEntries">
    <ElementMapping name="Log" maps-to="LogMessage(l)">
      <VirtualAttributeMapping binds-to="l" assume-element-value="No"
                               element-prefix="logentry-"/>
      <AttributeMapping name="date" maps-to="LogDate(l)"/>
      <AttributeMapping name="severity" maps-to="LogSeverity(l)"/>
    </ElementMapping>
  </ElementMapping>
</AimmsXMLSchemaMapping>
```

When reading the XML file, AIMMS will create two elements 'logentry-1' and 'logentry-2' into the set LogEntries. When writing the XML file, AIMMS will write Log elements whenever there is non-default data for LogDate(l), LogSeverity(l) or LogMessage(l), regardless of the specific format of the elements in the set LogEntries.

A second example

Consider the following XML file

```
<Flows>
   <Flow>Inflow</Flow>
   <Flow>Mix</Flow>
   <Flow>NH3-Mix</Flow>
   <Flow>NH3-Flow</Flow>
   <Flow>Residue</Flow>
   <Flow>Ar-Flow</Flow>
   <Flow>Feedback</Flow>
</Flows>
```

This XML format can be used to represent an AIMMS set Flows with an index f. The following schema mapping accomplishes this.

```
<AimmsXMLSchemaMapping xmlns="http://www.aimms.com/XMLSchema/AimmsXMLMappingSchema"
            MappedNameSpace="http://www.aimms.com/FlowsExample"
            default-width=16 default-precision=2>
  <ElementMapping name="Flows">
    <ElementMapping name="Flow">
      <VirtualAttributeMapping binds-to="f" maps-to="Flows"
                               assume-element-value="Yes"/>
    </ElementMapping>
  </ElementMapping>
</AimmsXMLSchemaMapping>
```

In this mapping, the element values of the Flow elements are taken as the value of a virtual attribute bound to the index f. The maps-to attribute is added to ensure that on reading the set Flows is filled with the encountered flow names, and on writing a Flow element is written out for every element in the set Flows.

Checking and saving the mapping file

On pressing the **OK** button in the **XML Schema Mapping** dialog box, AIMMS checks the validity of your mapping, and reports any errors it encounters. If there are no errors, AIMMS will save (or update) the mapping file associated with the XML schema file (.xsd extension) that you selected when opening the dialog box. The mapping file will be saved as an .axm file, with the same base name as the .xsd file.

Obtaining user-defined XML

Once you have created a mapping file between a given XML schema and the appropriate identifiers in your model, you can use the functions

- WriteXML(*XMLFile*,*MappingFile*[,*merge*])
- ReadXML(*XMLFile*,*MappingFile*[,*merge*][,*SchemaFile*])

to read data from, and write data to, an XML data file in the specified format.

The function WriteXML

The function WriteXML lets AIMMS generate XML data and write it into the file *XMLFile* based on the mapping file *MappingFile*. The optional *merge* argument (default 0) indicates whether you want to merge the generated XML data into another XML document, in which case AIMMS will omit the XML header from the generated XML file. This allows you to merge the contents of the generated file into another XML file. Note that setting the *merge* argument to 1 does not result in the generated XML data being appended to the specified file, its contents are completely overwritten. All data in the XML file are represented with respect to the currently active unit convention (see also Section 32.8). The function will return 1 if successful, or 0 if not.

Adding a namespace

If your XML schema file defines a namespace, reflected in the MappedNameSpace attribute of the root node in the corresponding .axm file, AIMMS will add this namespace to the XML file written by WriteXML through the xmlns attribute the root node of that file. If your XML schema file does not define a namespace, the MappedNameSpace attribute in the .axm file contains an artificial namespace URI "http://tempuri.org/AIMMS/auto-generated-namespace", which will not be added as the xmlns attribute to the root node of the file being written

The function ReadXML

Using the function ReadXML you can let AIMMS read the XML data contained in the file *XMLFile* into the AIMMS identifiers specified in the mapping file *MappingFile*. If the mapping file contains a valid (i.e. not generated by AIMMS) namespace URI of the corresponding XML schema, AIMMS requires the root element of the XML data file to be also associated with the namespace through the xmlns attribute. With the optional *merge* argument (default 0), you may indicate whether you want to merge the data included in the XML file with the existing data, or overwrite any existing data (default). All data in the XML file will be interpreted in accordance with the currently active unit convention (see also Section 32.8). The function will return 1 if successful, or 0 if not.

Schema validation

If you specify an optional *SchemaFile*, the XML parser used by AIMMS will validate the contents of the XML data contained in your XML file against this schema. This will only work, however, if the specified schema file defines a namespace matching the xmlns attribute of the root node of your XML file.