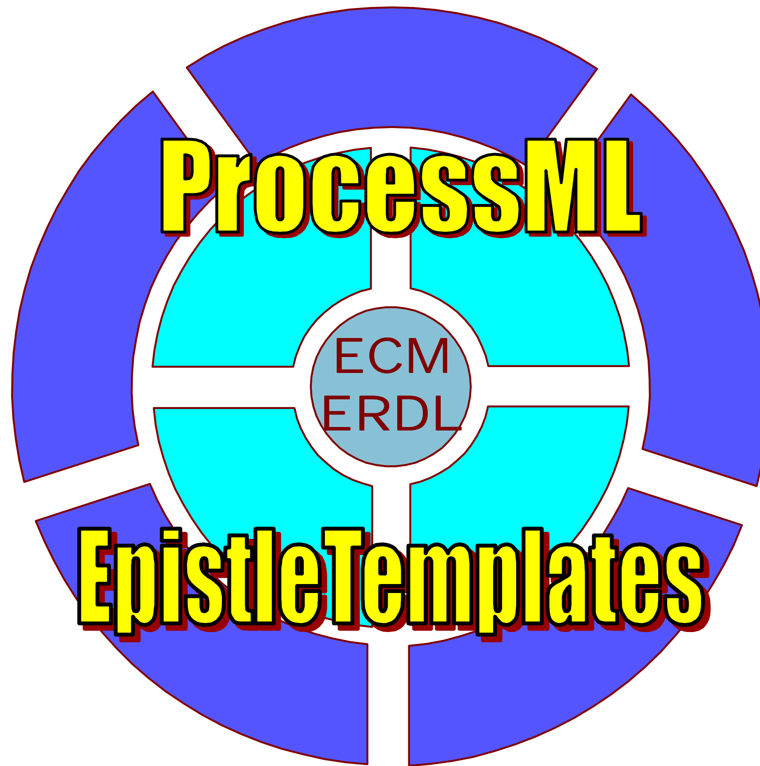


ProcessML^tm Architecture

rev. 1.0
29 May 2001
by Hans Teijgeler – Fluor Corp.

Table of Contents

Introduction ____ *3*

ProcessML^{tm} ____ *4*

Using XML for ECM-compliant data exchange ____ *5*

ProcessML^{tm} Architecture ____ *6*

Annex A – Examples ____ *8*

ECM-21 Template ____ **8**

ECM-X Template ____ **9**

EpistleTemplate ____ **10**

e-DataViewer – XML file ____ **11**

e-DataViewer – browser screen paint ____ **12**

Annex B – Description of used W3C Recommendations ____ *13*

XML 1.0 2nd Edition ____ **14**

XML Information Set ____ **15**

XML Inclusions (XInclude) 1.0 ____ **17**

XML Schema Part 1: Structures ____ **18**

XML Schema Part 2: Datatypes ____ **20**

XML Namespaces ____ **22**

URI ____ **23**

XPath 1.0 ____ **24**

XLink 1.0 ____ **26**

XSL 1.0 ____ **27**

XSLT 1.0 ____ **29**

XML Fragment Interchange ____ **32**

ProcessML[tm] Architecture rev 1.0

Introduction

This document is intended to support the decision making process in EPISTLE[1] regarding the application of XML[2] for the implementation of EPISTLE Templates[tm] in the realm of the ProcessML[tm 3] initiative in EPISTLE.

The positions taken in this document are for the author's own account only, and do not represent a formal position taken by his employer, Fluor Corporation, nor by EPISTLE or its member consortia or the member companies of those consortia.

The reader is invited to challenge such positions taken, because in the end we need consensus about how EPISTLE-endorsed templates will be implemented.

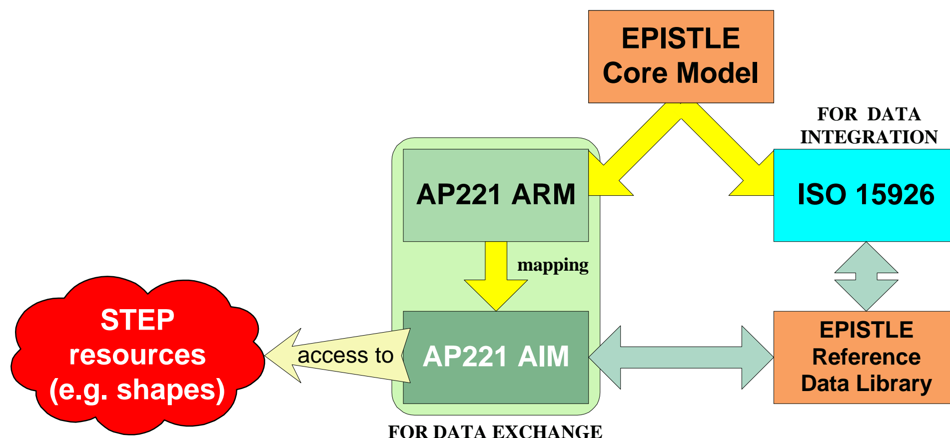
This document has a controlled distribution for W3C **copyright** reasons.
The recipient is urgently requested **NOT** to make paper or electronic copies or to forward this document in electronic format.

[1] EPISTLE = European Process Industries STEP Technical Liaison Executive
[2] XML = eXtensible markup Language
[3] ProcessML = XML as applied for the Process Industries, based on ISO 15926-2 and the ERDL (EPISTLE Reference Data Library).

ProcessML^tm

In the, mainly European, world of the process industries (refineries, chemical plants, pharma, food, power, etc) a data model for the description of process facilities and their lifetime data has emerged in the form of the EPISTLE Core Model version 4 ('ECMv4').

That model is being used in two ISO standards:
- ISO 15926-2
- ISO 10303-221 ('AP221' in STEP)



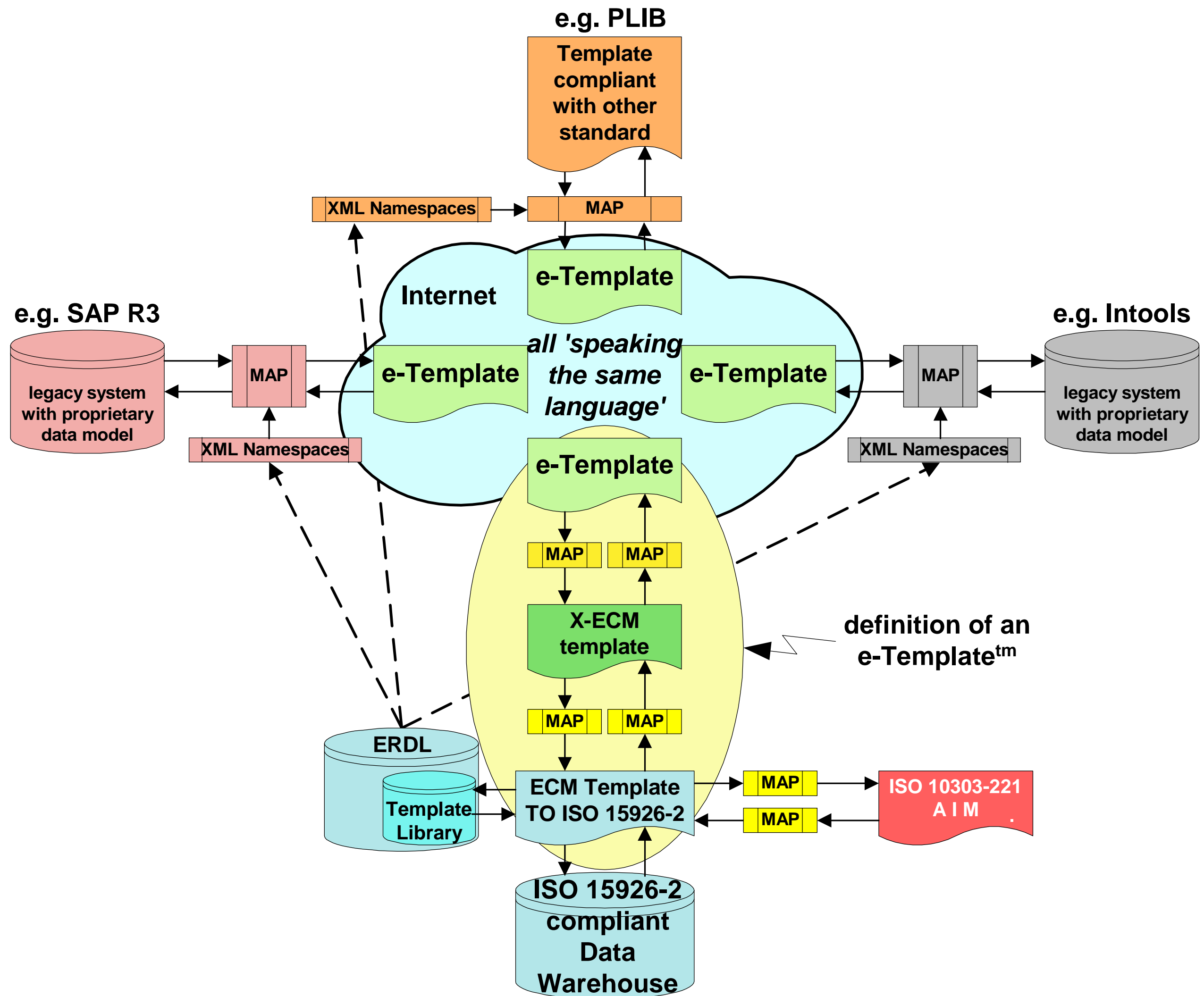
ECMv4 is a highly generic data model with 155 entity types which are considered enough to describe the technical universe. This is possible by using a library of concepts, the EPISTLE Reference Data Library ('ERDL'). The ERDL is also expressed in ECMv4 terms.

For the exchange of information about technical installations and the activities (processes) that are performed by such installations, the ECMv4 can be used only by means of what is dubbed the EPISTLE Templates^tm, or **e-Templates^tm** , a.k.a. business objects.

Using XML for ECM-compliant data exchange

These e-Templates™ are represented in XML, the new e**X**tensible **M**ark-up **L**anguage, that will bring the Internet to new levels of usefulness.

An overview of the proposed architecture is given below:



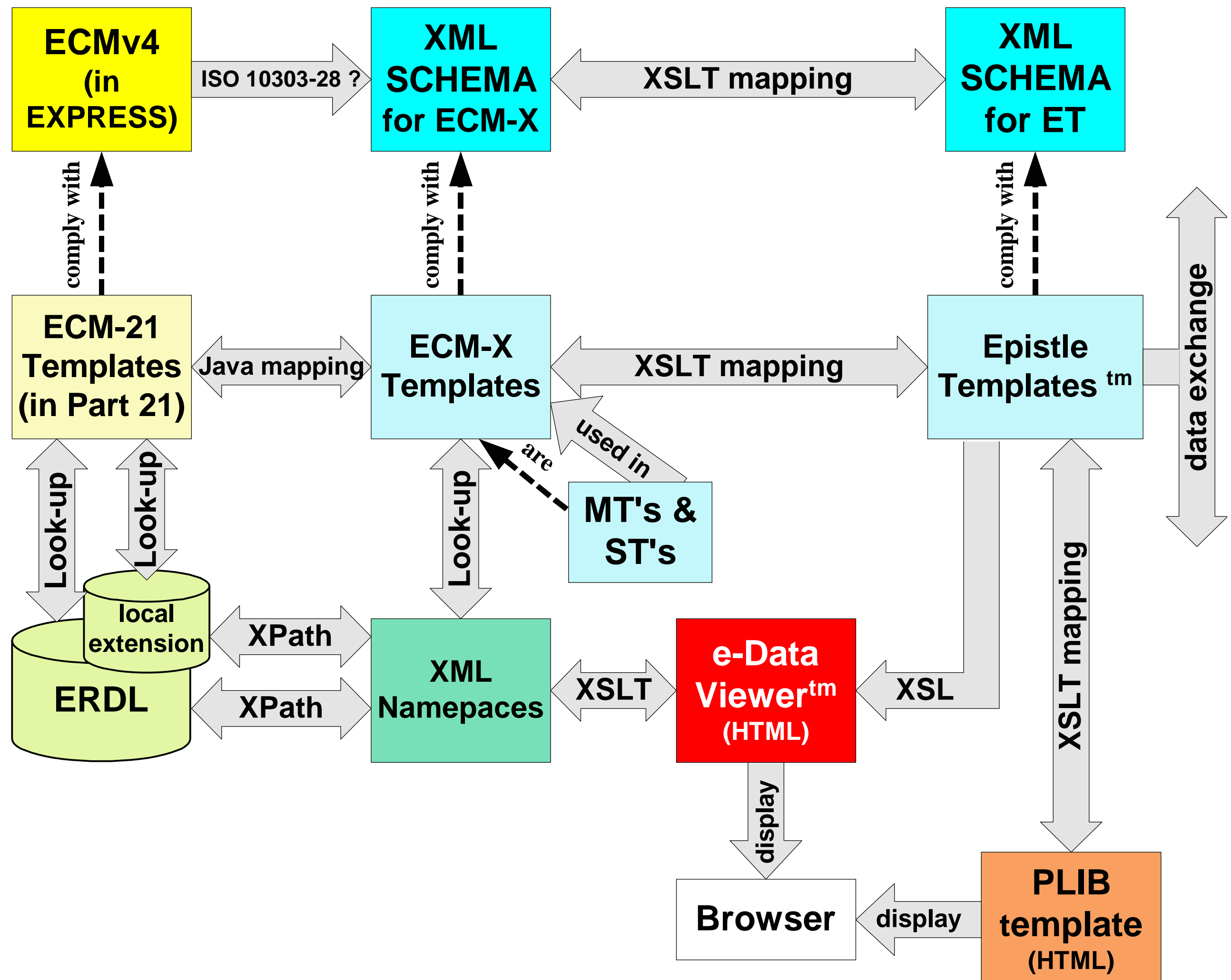
Some notes:

- It is assumed that legacy systems will have an XML import/export facility. Microsoft is taking a leading position in this.
- Mapping to and from templates complying with other standards will be on the basis of e-Templates, so without the full set of relations that (X-)ECM Templates have.
  This includes PLIB templates
- The full set of:
  - ECM Template
  - X-ECM Template
  - e-Template™
  - all mapping programs

  must be tested and approved by an EPISTLE-assigned body, and after approval be made available from the EPISTLE website.

ProcessML^tm Architecture

A more detailed architecture for the mapping between ECM Templates and e-Templates^tm is shown in the following diagram, that gives an impression of the ProcessML^tm architecture and its modules:



ECM-21 Template
ECM-21 Templates are EPISTLE ECM4-compliant, expressed in Part21 (ISO 10303-21) format using the ERDL and its local extension(s) for the full definition of its terms and values.

ECM-X Template
These templates are mapped to and from their XML-based replica ECM-X Templates.
This mapping is done in Java (or another procedural language, to the choice of the software developer). These templates must comply with the related XML Schema.

> Note – Whether or not a mapping between the ECM4 EXPRESS model and its XML Schema counterpart by means of ISO 10303-28 is possible is still uncertain. This will be investigated.

The ECM-X templates are composed from Molecular and Structure Templates that will be stored in the ERDL and its local extension(s).

Using the W3C Recommendation ‘XML Namespaces’ and ‘XPath’ these templates are linked to the ERDL and its local extension(s) as well.

EpistleTemplate[tm]
From the ECM-X Templates we generate, by means of mapping, thereby using the W3C Recommendation ‘XSLT’, the so-called EpistleTemplates[tm]. These contain only the ‘sensible data’, be it in terms of ERDL numbers.
These EpistleTemplates[tm] are used for the actual data exchange between systems.

e-DataViewer[tm]
These EpistleTemplates[tm] can be mapped to their human-readable format, such as in the form of data sheets. The use of XSL-FO, XSLT, and HTML allows us to present these with a standard browser, in a layout of choice, fully coloured, if so required.. These can be read by Microsoft’s Internet Explorer 5.5 or higher (Netscape 6 still shows problems). The name for this is e-DataViewer[tm].

Other viewers
Other ways of providing a human-readable format, such as PDXML, where data sheets are made in Excel, and mapped to XML, are possible. In order to become EPISTLE-certified the mapping to an EpistleTemplate[tm], and then to ECM-X Template and ECM-21 Template must be made and approved.

PLIB Templates
The EpistleTemplates[tm] can be mapped to/from PLIB Templates for display with a browser as well.

Legacy System Interfaces
Computer systems can get an import/export facility for EpistleTemplates[tm]. This will require extra, system-specific, templates, in particular in cases of “implicit information”. Such templates can be composed of EPISTLE-certified Molecular Templates and, sometimes, Structure Templates.

Annex A – Examples

In this annex examples are given of prototype-quality templates:
- ECM-21 Template
- ECM-X Template
- EpistleTemplate
- e-DataViewer – XML file
- e-DataViewer – browser screen paint

In due time a complete set of final template files and the transformations in between will be published.

Please bear with us that none of the templates shown in this document has been formally approved by EPISTLE. The ones shown are representative examples required to demonstrate the concept.

ECM-21 Template

This is a straight-forward Part 21 file with reference to ISO 15926-2

This is what comes after the Part 21 header when expressing an instance of MT_identification:

```
#1 THING(erdl:79834511:327634992094); -- substitute with an instance of a subtype
#2 CLASS_OF_IDENTIFICATION(erdl:93876302, 1,1,1,1,#3,#1);
#3 EXPRESS_STRING(erdl:79834511:902398238953,content);
#4 CLASSIFICATION(erdl:798345:,#2,#5);
#5 CLASS_OF_CLASS_OF_IDENTIFICATION(erdl:79834511,$,$);
#6 CLASS_OF_RESPONSIBILITY_FOR_REPRESENTATION(erdl:83757392,1,1,1,1,#2,#7);
#7 STATE(erdl:79870203);
```

Note 1: The numbering scheme used here for the ‘id’ still has to be agreed upon in EPISTLE.
Explanation:
- “erdl:79834511:” stands for the organization that is a “reference individual” that is registered in the ERDL under this id. It provides the context for the id that follows.
- “erdl:93876302” is the (made-up) id of a class registered in the ERDL
- “erdl:79870203” is the (made-up) id for the reference individual ‘Shell Nederland Raffinaderijen N.V.’ as registered in the ERDL

Such a numbering scheme fits perfectly with XML Namespaces shown below.

ECM-X Template

```
  <?xml version="1.0" ?>
- <ISO-10303-data>
  - <header_section>
      <produced-by />
      <time-stamp />
    </header_section>
  - <data_section>
    - <schema_instance express_schema_name="process_plant_lifecycle">
      - <entity_instance instance_type="physical_object" instanceID="i1">
        - <attribute_instance attribute_name="id">
            <string_literal>loc:100001</string_literal>
          </attribute_instance>
        </entity_instance>
      - <entity_instance instance_type="temporal_whole_part" instanceID="i2">
        - <attribute_instance attribute_name="id">
            <string_literal>loc:100002</string_literal>
          </attribute_instance>
        - <attribute_instance attribute_name="whole">
            <entity_instance_ref instanceID="i1" />
          </attribute_instance>
        - <attribute_instance attribute_name="part">
            <entity_instance_ref instanceID="i3" />
          </attribute_instance>
        </entity_instance>
      - <entity_instance instance_type="physical_object" instanceID="i3">
        - <attribute_instance attribute_name="id">
            <string_literal>loc:100003</string_literal>
          </attribute_instance>
        </entity_instance>
      + <entity_instance instance_type="start_temporal_bounding_of_state" instanceID=
      - <entity_instance instance_type="point_in_time" instanceID="i5">
        - <attribute_instance attribute_name="id">
            <string_literal>loc:100005</string_literal>
          </attribute_instance>
        </entity_instance>
```

etc

The two role attributes of relations and classes of relation are represented in XML like in the following snippet:

<attribute_instance attribute_name=”whole”>
<entity_instance_ref instanceID=”i1”>
</attribute_instance>

where the “i1” is used for the Part 21 #number.

EpistleTemplate

```
  <?xml version="1.0" ?>
- <wrapper xmlns:erdl="file:erdl.xml" xmlns:loc="file:loc.xml">
    <!--       -->
  - <globals erdl:name_id="?">
    - <file_id erdl:name_id="-51">
        <loc:value />
      </file_id>
    - <plant erdl:name_id="-52">
        <erdl:value>-21</erdl:value>
      </plant>
    - <plant_owner erdl:name_id="-50">
        <loc:value>10097661</loc:value>
      </plant_owner>
    - <plant_operator erdl:name_id="-53">
        <loc:value />
      </plant_operator>
    </globals>
  - <e-templates>
    - <e-template>
        <!--  flow transmitter   -->
      - <mt_physical_object_instantiation_with_identification>
        - <whole_physical_object>
            <loc:lifetime_id>100007</loc:lifetime_id>
          </whole_physical_object>
        - <class>
            <erdl:lifetime_id>70096</erdl:lifetime_id>
          </class>
        - <whole_physical_object_start_date_time>
            <loc:start_date_time>100002</loc:start_date_time>
          </whole_physical_object_start_date_time>
        - <identified_substate_start_date_time>
            <loc:start_date_time>100002</loc:start_date_time>
          </identified_substate_start_date_time>
        - <type_of_identifier>
            <erdl:lifetime_id>910300</erdl:lifetime_id>
          </type_of_identifier>
        - <identifier>
            <loc:lifetime_id>100001</loc:lifetime_id>
          </identifier>
        - <controller_of_identifier>
            <erdl:lifetime_id>-21</erdl:lifetime_id>
          </controller_of_identifier>
```

etc.

Here only the “sensible data” (the yellow blocks in the template instance diagrams) are being exchanged.

Above example, starting with <e-template>, shows the sensible data required for the Molecular Template called MT_physical_object_instantiation_with_identification (shown in the document above). We define:

- the id of the instantiated whole physical object
- the id of its “take-off” class (e.g. ‘flow transmitter’ rather than already ‘electronic dp type flow transmitter’ (that comes later)
- the start datetime for the substate that we created for the identification
- the type of identification (e.g. ‘with tag number’)
- the actual identifier (EXPRESS_)string
- the ‘controller’ state of that identification.

Note: id’s with a negative value are still to be defined, this is work in progress.

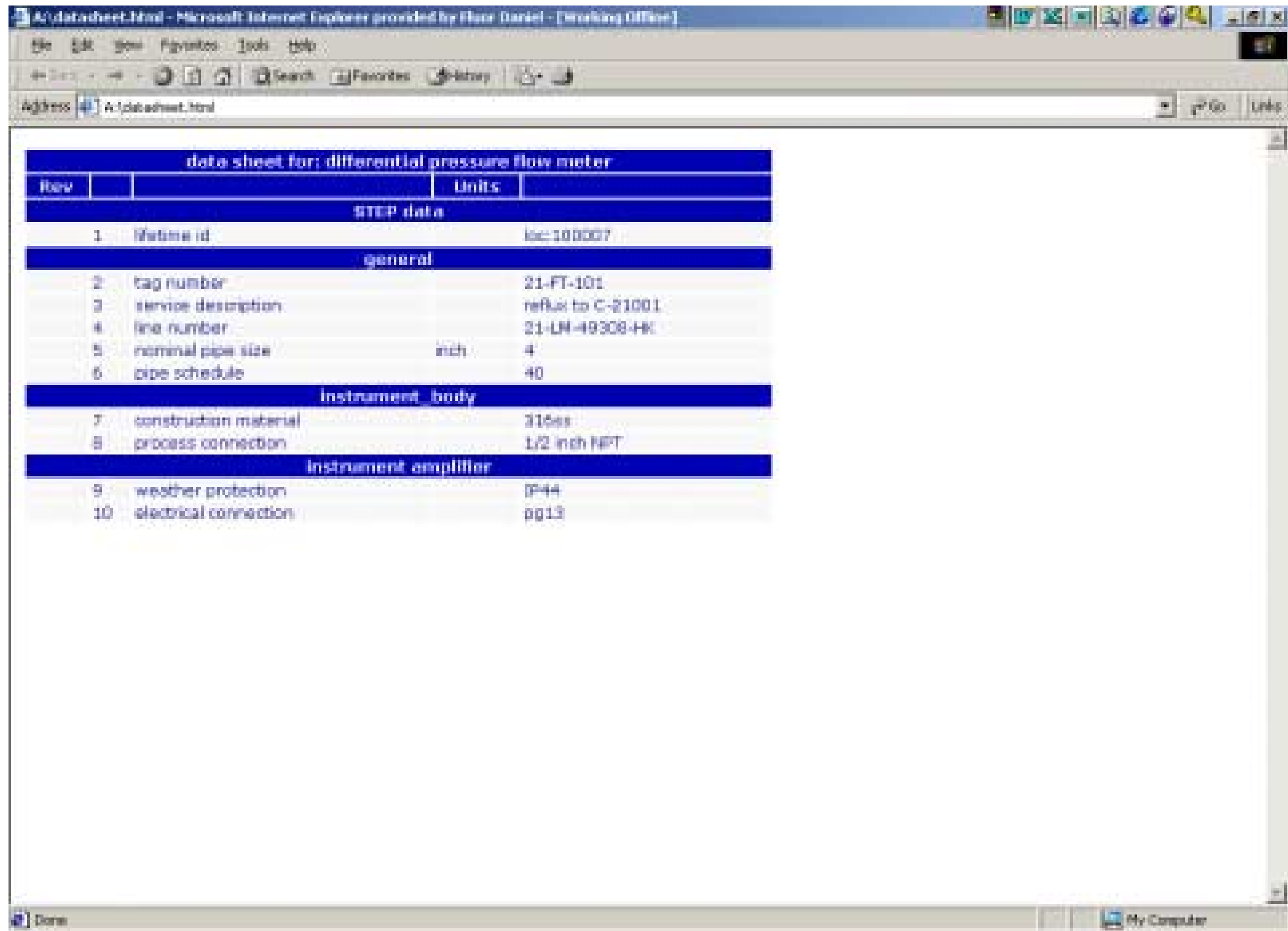
e-DataViewer – XML file


```
  <?xml version="1.0" encoding="utf-8" ?>
- <wrapper xmlns:erdl="file:erdl.xml" xmlns:loc="file:loc.xml">
  - <e-dataviewer>
    - <item>
        <lifetime_id>loc:100007</lifetime_id>
        <lifetime_class>differential pressure flow meter</lifetime_class>
        <unique_id prompt="tag number" controller="my process plant (mpp)">21-FT-101</unique_id>
      - <instrument_body prompt="instrument body">
          <construction_material prompt="construction material">316ss</construction_material>
          <process_connection prompt="process connection">1/2 inch NPT</process_connection>
        </instrument_body>
      - <instrument_amplifier prompt="instrument amplifier">
          <weather_protection prompt="weather protection">IP44</weather_protection>
          <electrical_connection prompt="electrical connection">pg13</electrical_connection>
        </instrument_amplifier>
        <service_description prompt="service description">reflux to C-21001</service_description>
        <pid_number />
        <connected_line_segment>loc:100023</connected_line_segment>
        <document_title_block />
      </item>
    - <item>
        <lifetime_id>loc:100024</lifetime_id>
        <lifetime_class>pipe line</lifetime_class>
        <unique_id prompt="line number" controller="my process plant (mpp)">21-LM-49308-HK</unique_id>
      - <line_segments>
        - <line_segment>
            <lifetime_id>loc:100023</lifetime_id>
            <lifetime_class>line segment</lifetime_class>
            <unique_id prompt="line segment number" controller="my process plant (mpp)">90031200_1</unique_id>
            <construction_material prompt="construction material">carbon steel</construction_material>
            <pipe_size prompt="nominal pipe size" units="inch">4</pipe_size>
            <pipe_schedule prompt="pipe schedule">40</pipe_schedule>
          </line_segment>
        </line_segments>
        <service_description prompt="service description">reflux to C-21001</service_description>
        <pid_number />
      </item>
    </e-dataviewer>
  </wrapper>
```


Here the user can define the prompts as they must appear on the data sheet on the browser. Furthermore the class and individual id's as defined in the ERDL and mentioned local extensions have been sutomatically translated. Which one applies is defined with the prefix 'erdl' or 'loc' (for local extension). These prefixes are defined in the second line, after 'wrapper'. The acronym 'xmlns' stands for XML Namespace. These namespaces are defined as a 'resource', such as the URL for ERDL on the EPISTLE website. Here we used some local files with an excerpt of the ERDL as namespace.

This file can be used to create the screen shown on the next page.

e-DataViewer – browser screen paint



data sheet for: differential pressure flow meter				
Rev			Units	
STEP data				
	1	lifetime id		loc:100007
general				
	2	tag number		21-FT-101
	3	service description		reflux to C-21001
	4	line number		21-LM-49308-HK
	5	nominal pipe size	inch	4
	6	pipe schedule		40
instrument_body				
	7	construction material		316ss
	8	process connection		1/2 inch NPT
instrument amplifier				
	9	weather protection		IP44
	10	electrical connection		pg13

This is just a quick-and-dirty, but demonstrable, example. There are tools on the market that allow you to make the layout you want, and then map the data in it.

Annex B – Description of used W3C Recommendations

On the following pages an overview of the W3C Recommendations will be given, with copyrighted text selected from their site www.w3.org/TR/ and some other sources.

The reader is urged to respect this, and not to make further copies of this document.

These excerpts are meant to give a first insight in the scope of the given Recommendation (the W3C word for 'standard', for anti-trust reasons). As can be observed, much of that material is brandnew, and many are still under development.

NOTE – The W3C classification for Technical Reports is:

- **Note**: A Note is a dated, public record of an idea, comment, or document. A Note does not represent commitment by W3C to pursue work related to the Note.
- **Working Draft**: A Working Draft represents work in progress and a commitment by W3C to pursue work in this area. A Working Draft does not imply consensus by a group or W3C.
- **Candidate Recommendation**: A Candidate Recommendation is work that has received significant review from its immediate technical community. It is an explicit call to those outside of the related Working Groups or the W3C itself for implementation and technical feedback.
- **Proposed Recommendation**: A Proposed Recommendation is work that (1) represents consensus within the group that produced it and (2) has been proposed by the Director to the Advisory Committee for review.
- **Recommendation**: A Recommendation is work that represents consensus within W3C and has the Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission.

XML 1.0 2nd Edition

Status: W3C Recommendation
Date: 6 October 2000

http://www.w3.org/TR/REC-xml-names/

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

XML describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them.

XML is an application profile or restricted form of SGML *(ISO 8879:1986(E). Information processing -- Text and Office Systems – Standard Generalized Markup Language (SGML). First edition -- 1986-10-15)*. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application.

This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

A data object is an XML document if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly

> NOTE – The XML 'entity' has a meaning that is different from what is common practice in IT.

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named *xml:lang* may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by http://www.w3.org/TR/REC-xml-names/

XML Information Set

Status: Working Draft
Date: 16 March 2001

http://www.w3.org/TR/xml-infoset/

XML Information Set (also referred to as 'Infoset') is an abstract data set. Its purpose is to provide a consistent set of definitions for use in other specifications that need to refer to the information in a well-formed XML document.

It does not attempt to be exhaustive; the primary criterion for inclusion of an information item or property has been that of expected usefulness in future specifications. Nor does it constitute a minimum set of information that must be returned by an XML processor.

An XML document has an information set if it is well-formed and satisfies the namespace constraints described below. There is no requirement for an XML document to be valid in order to have an information set.

An XML document's information set consists of a number of information items (the information set for any well-formed XML document will contain at least a document information item and several others).

An information item is an abstract representation of some part of an XML document: each information item has a set of associated named properties.

The XML Information Set does not require or favor a specific interface or class of interfaces. This specification presents the information set as a modified tree for the sake of clarity and simplicity, but there is no requirement that the XML Information Set be made available through a tree structure; other types of interfaces, including (but not limited to) event-based and query-based interfaces are also capable of providing information conforming to the XML Information Set.

The terms "information set" and "information item" are similar in meaning to the generic terms "tree" and "node", as they are used in computing. However, the latter terms were avoided in this specification to reduce possible confusion with other specific data models. Information items do not map one-to-one with the nodes of the DOM or the "tree" and "nodes" of the XPath data model.

Namespaces
XML 1.0 documents that do not conform to XML Namespaces, though technically well-formed, are not considered to have meaningful information sets. That is, this specification does not define an information set for documents that have element or attribute names containing colons that are used in other ways than as prescribed by [Namespaces].

Furthermore, this specification does not define an information set for documents which use relative URI references in namespace declarations. This is in accordance with the decision of the W3C XML Plenary Interest Group described in Relative Namespace URI References. Thus the value of a [namespace name] property is always an absolute URI with an optional fragment identifier.

An information set describes its XML document with entity references already expanded, that is, represented by the information items corresponding to their replacement text. However, there are

various circumstances in which a processor may not perform this expansion. An entity may not be declared, or may not be retrievable. A non-validating processor may choose not to read all declarations, and even if it does may not expand all external entities. In these cases an unexpanded entity reference information item is used to represent the entity reference.

Base URIs

Several information items have a ‘base URI’ property. This is computed according to XML Base. Note that retrieval of a resource may involve redirection at the parser level (for example, in an entity resolver) or below; in this case the base URI is the final URI used to retrieve the resource after all redirection.

``Unknown'' and ``No Value''

Some properties may sometimes have the value unknown or no value, and it is said that a property value is unknown or that a property has no value respectively. These values are distinct from each other and from all other values. In particular they are distinct from the empty string, the empty set, and the empty list, each of which simply has no members. This specification does not use the term null since in some communities it has particular connotations which may not match those intended here.

Synthetic Infosets

This specification describes the information set resulting from parsing an XML document. Information sets may be constructed by other means, for example by use of an API such as the DOM or by transforming an existing information set.

An information set corresponding to a real document will necessarily be consistent in various ways; for example the in-scope namespaces property of an element will be consistent with the namespace attributes properties of the element and its ancestors. This may not be true of an information set constructed by other means; in such a case there will be no XML document corresponding to the information set, and to serialize it will require resolution of the inconsistencies (for example, by outputting namespace declarations that correspond to the namespaces in scope).

XML Inclusions (XInclude) 1.0

Status: Working Draft
Date: 26 October 2000

XInclude specifies a processing model and syntax for general purpose inclusion. Inclusion is accomplished by merging a number of XML Infosets into a single composite Infoset. Specification of the XML documents (infosets) to be merged and control over the merging process is expressed in XML-friendly syntax (elements, attributes, URI References).

XML Schema Part 1: Structures

Status: W3C Recommendation
Date: 2 May 2001

http:/www.w3.org/xmlschema-1/

An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element information items (as defined in XML Information Set), and furthermore may specify augmentations to those items and their descendants.

This augmentation makes explicit information which may have been implicit in the original document, such as normalized and/or default values for attributes and elements and the types of element and attribute information items.

Schema-validity assessment has two aspects:

1. determining local schema-validity, that is whether an element or attribute information item satisfies the constraints embodied in the relevant components of an XML Schema;
2. synthesising an overall validation outcome for the item, combining local schema-validity with the results of schema-validity assessments of its descendants, if any, and adding appropriate augmentations to the infoset to record this outcome.

The definition of XML Schema: Structures depends on the following specifications:

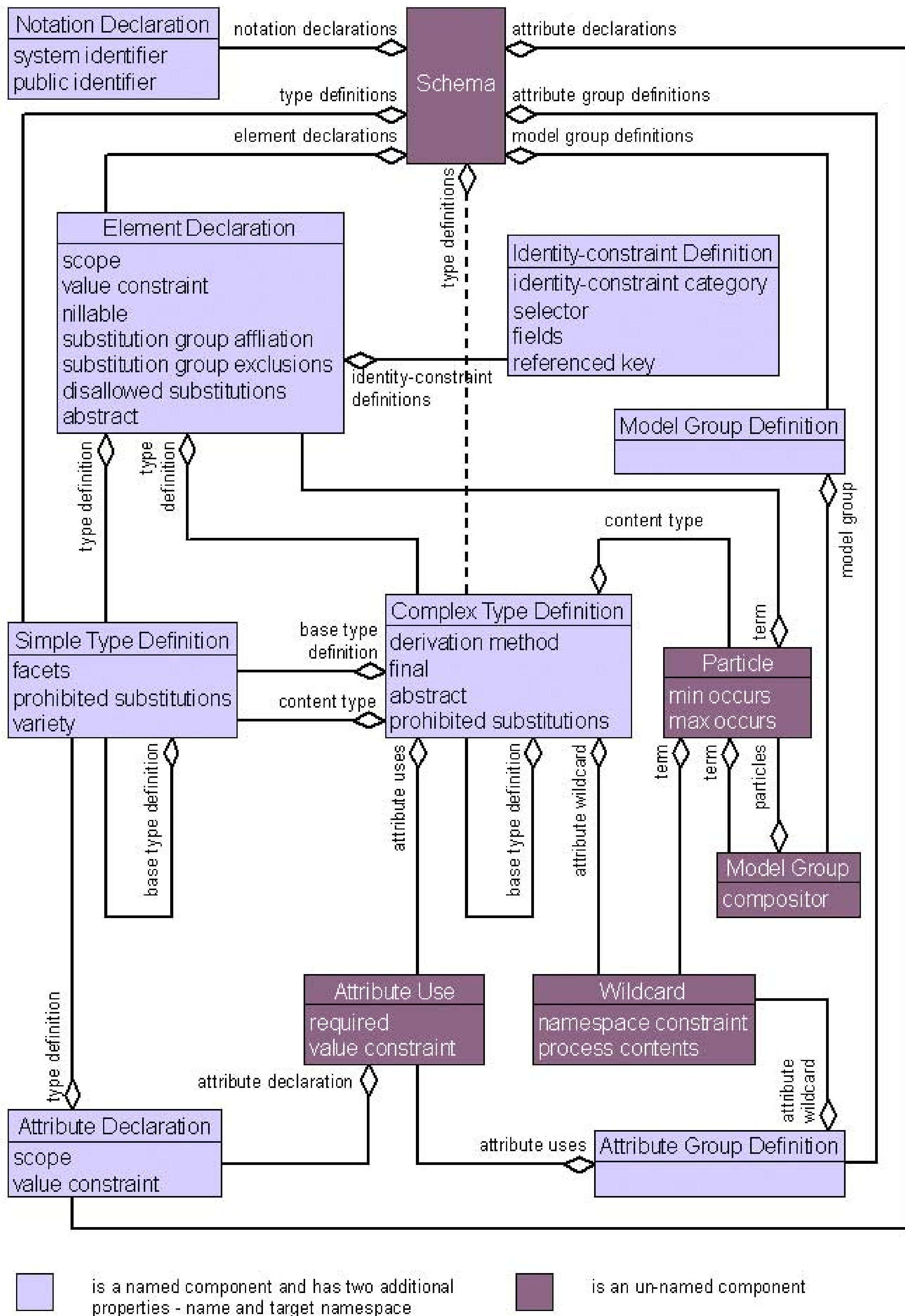
- XML Schemas: Datatypes
- XML-Information Set
- XML-Namespaces
- XPath

Before this specification is finally completed, we will need to account for any changes XML Base makes to XML Information Set in the areas of QName interpretation and value space and the interpretation of all aspects of schemas involving values identified as being of type anyURI, including in particular *xsi:schemaLocation*, *xsi:noNamespaceSchemaLocation* and *targetNamespace*.

> NOTE - A QName is a name with an optional namespace qualification, as defined in XML-Namespaces. When used in connection with the XML representation of schema components or references to them, this refers to the simple type QName as defined in XML Schemas: Datatypes.

See XML Schemas: Datatypes for the details of the anyURI type and all uses of URI references in this specification.

See Required Information Set Items and Properties (normative) for a tabulation of the information items and properties specified in XML-Information Set which this specification requires as a precondition to schema-aware processing. (see schema overleaf).



XML Schema Component Data Model

XML Schema Part 2: Datatypes

Status: W3C Recommendation
Date: 2 May 2001

http://www.w3.org/TR/xmlschema-2/

The XML 1.0 (Second Edition) specification defines limited facilities for applying datatypes to document content in that documents may contain or refer to DTDs that assign types to elements and attributes. However, document authors, including authors of traditional documents and those transporting data in XML, often require a higher degree of type checking to ensure robustness in document understanding and data interchange.

The XML Schema Requirements document spells out concrete requirements to be fulfilled by this specification, which state that the XML Schema Language must:
1. provide for primitive data typing, including byte, date, integer, sequence, SQL and Java primitive datatypes, etc.;
2. define a type system that is adequate for import/export from database systems (e.g., relational, object, OLAP);
3. distinguish requirements relating to lexical data representation vs. those governing an underlying information set;
4. allow creation of user-defined datatypes, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (e.g., range, precision, length, format).

This portion of the XML Schema Language discusses datatypes that can be used in an XML Schema. These datatypes can be specified for element content that would be specified as #PCDATA and attribute values of various types in a DTD.

It describes the conceptual framework behind the type system defined in this specification. The framework has been influenced by the [ISO 11404] standard on language-independent datatypes as well as the datatypes for SQL and for programming languages such as Java.

It is the intention of this specification that it be usable outside of the context of XML Schemas for a wide range of other XML-related activities such as XSL and RDF Schema.

Namespace considerations
The built-in datatypes defined by this specification are designed to be used with the XML Schema definition language as well as other XML specifications. To facilitate usage within the XML Schema definition language, the built-in datatypes in this specification have the namespace name:

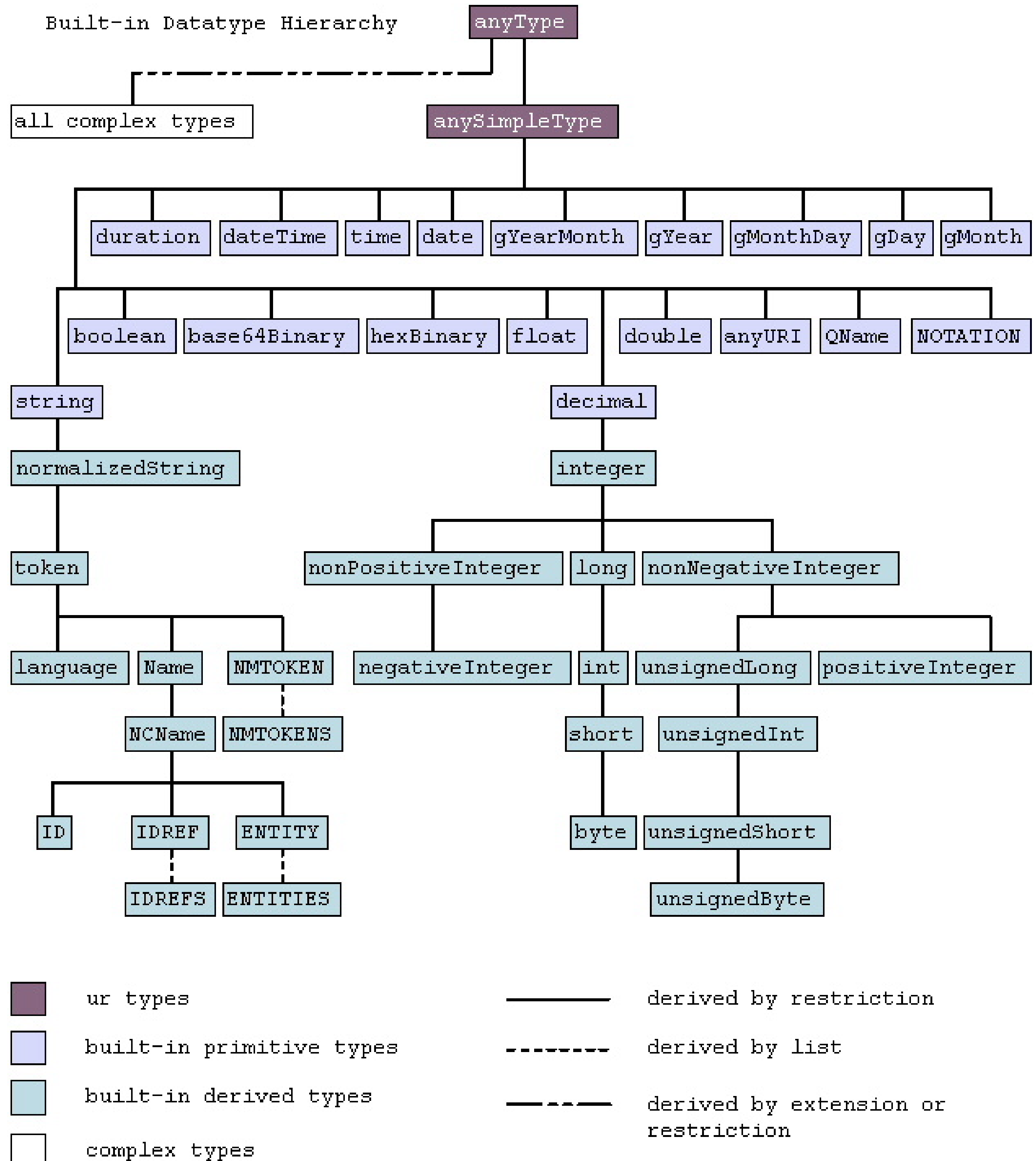
http://www.w3.org/2001/XMLSchema

To facilitate usage in specifications other than the XML Schema definition language, such as those that do not want to know anything about aspects of the XML Schema definition language other than the datatypes, each built-in datatype is also defined in the namespace whose URI is:

http://www.w3.org/2001/XMLSchema-datatypes

This applies to both built-in primitive and built-in derived datatypes.

Each user-derived datatype is also associated with a unique namespace. However, user-derived datatypes do not come from the namespace defined by this specification; rather, they come from the namespace of the schema in which they are defined (see XML Representation of Schemas in XML Schema Part 1: Structures).



XML Namespaces

Status: W3C Recommendation
Date:

http://www.w3.org/TR/REC-xml-names/

XML Namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references.

An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

> NOTE - XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set.

URI references which identify namespaces are considered identical when they are exactly the same character-for-character. Note that URI references which are not identical in this sense may in fact be functionally equivalent. Examples include URI references which differ only in case, or which are in external entities which have different effective base URIs.

Names from XML namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace. The combination of the universally managed URI namespace and the document's own namespace produces identifiers that are universally unique. Mechanisms are provided for prefix scoping and defaulting.

URI references can contain characters not allowed in names, so cannot be used directly as namespace prefixes. Therefore, the namespace prefix serves as a proxy for a URI reference. An attribute-based syntax is used to declare the association of the namespace prefix with a URI reference; software which supports this namespace proposal must recognize and act on these declarations and prefixes.

In XML documents conforming to this specification, no tag may contain two attributes which:
1. have identical names, or
2. have qualified names with the same local part and with prefixes which have been bound to namespace names that are identical.

URI

Status:
Date:

http://www.ietf.org/rfc/rfc2396.txt

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource.

URI are characterized by the following definitions:

1. Uniform
   Uniformity provides several benefits: it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ; it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers; it allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers.

2. Resource
   A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources.
   The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process.

An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax.

Having identified a resource, a system may perform a variety of operations on the resource, as might be characterized by such words as `access', `update', `replace', or `find attributes'.

A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource.

The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable

A URN differs from a URL in that it's primary purpose is persistent labeling of a resource with an identifier. That identifier is drawn from one of a set of defined namespaces, each of which has its own set name structure and assignment procedures. The "urn" scheme has been reserved to establish the requirements for a standardized URN namespace, as defined in "URN Syntax" and its related specifications.

XPath 1.0

Status: W3C Recommendation
Date: 16 Nov. 1999

http://www.w3.org/TR/xpath

XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer.

XPath gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document.

XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer.

The primary purpose of XPath is to address parts of an XML document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans.

XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax.

In addition to its use for addressing, XPath is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern); this use of XPath is described in XSLT.

XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes.

XPath defines a way to compute a string-value for each type of node. Some types of nodes also have names.

XPath fully supports XML Namespaces. Thus, the name of a node is modeled as a pair consisting of a local part and a possibly null namespace URI; this is called an expanded-name.

The primary syntactic construct in XPath is the expression. An expression is evaluated to yield an object, which has one of the following four basic types:
- node-set (an unordered collection of nodes without duplicates)
- boolean (true or false)
- number (a floating-point number)
- string (a sequence of UCS characters)

Expression evaluation occurs with respect to a context. XSLT and XPointer specify how the context is determined for XPath expressions used in XSLT and XPointer respectively. The context consists of:
- a node (the context node)
- a pair of non-zero positive integers (the context position and the context size)
- a set of variable bindings
- a function library
- the set of namespace declarations in scope for the expression

The context position is always less than or equal to the context size.

The variable bindings consist of a mapping from variable names to variable values. The value of a variable is an object, which can be of any of the types that are possible for the value of an expression, and may also be of additional types not specified here.

The function library consists of a mapping from function names to functions. Each function takes zero or more arguments and returns a single result. This document defines a core function library that all XPath implementations must support For a function in the core function library, arguments and result are of the four basic types. Both XSLT and XPointer extend XPath by defining additional functions; some of these functions operate on the four basic types; others operate on additional data types defined by XSLT and XPointer.

The namespace declarations consist of a mapping from prefixes to namespace URIs.

The variable bindings, function library and namespace declarations used to evaluate a subexpression are always the same as those used to evaluate the containing expression. The context node, context position, and context size used to evaluate a subexpression are sometimes different from those used to evaluate the containing expression. Several kinds of expressions change the context node; only predicates change the context position and context size.

When the evaluation of a kind of expression is described, it will always be explicitly stated if the context node, context position, and context size change for the evaluation of subexpressions; if nothing is said about the context node, context position, and context size, they remain unchanged for the evaluation of subexpressions of that kind of expression.

XPath expressions often occur in XML attributes. The grammar specified in this section applies to the attribute value after XML 1.0 normalization.

So, for example, if the grammar uses the character <, this must not appear in the XML source as < but must be quoted according to XML 1.0 rules by, for example, entering it as <. Within expressions, literal strings are delimited by single or double quotation marks, which are also used to delimit XML attributes. To avoid a quotation mark in an expression being interpreted by the XML processor as terminating the attribute value the quotation mark can be entered as a character reference (" or '). Alternatively, the expression can use single quotation marks if the XML attribute is delimited with double quotation marks or vice-versa.

One important kind of expression is a location path. A location path selects a set of nodes relative to the context node. The result of evaluating an expression that is a location path is the node-set containing the nodes selected by the location path. Location paths can recursively contain expressions that are used to filter sets of nodes. A location path matches the production LocationPath.

XLink 1.0

Status: Proposed Recommendation
Date: 20 Dec. 2000

http://www.w3.org/TR/xlink/

This specification defines the XML Linking Language (XLink), which allows elements to be inserted into XML documents in order to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated links.

It allows XML documents to:
- Assert linking relationships among more than two resources
- Associate metadata with a link
- Express links that reside in a location separate from the linked resources

An important application of XLink is in hypermedia systems that have hyperlinks. A simple case of a hyperlink is an HTML A element, which has these characteristics:
- The hyperlink uses URIs as its locator technology.
- The hyperlink is expressed at one of its two ends.
- The hyperlink identifies the other end (although a server may have great freedom in finding or dynamically creating that destination).
- Users can initiate traversal only from the end where the hyperlink is expressed to the other end.
- The hyperlink's effect on windows, frames, go-back lists, stylesheets in use, and so on is determined by user agents, not by the hyperlink itself. For example, traversal of A links normally replaces the current view, perhaps with a user option to open a new window.

This set of characteristics is powerful, but the model that underlies them limits the range of possible hyperlink functionality. The model defined in this specification shares with HTML the use of URI technology, but goes beyond HTML in offering features, previously available only in dedicated hypermedia systems, that make hyperlinking more scalable and flexible.

Along with providing linking data structures, XLink provides a minimal link behavior model; higher-level applications layered on XLink will often specify alternate or more sophisticated rendering and processing treatments.

Integrated treatment of specialized links used in other technical domains, such as foreign keys in relational databases and reference values in programming languages, is outside the scope of this specification.

XSL 1.0

Status: Candidate Recommendation
Date: 21 Nov. 2000

http://www.w3.org/TR/xsl/
(unfortunately one cannot browse through that document, and has to download it)

XSL is a language for expressing stylesheets. It consists of two parts:
1. a language for transforming XML documents, and
2. an XML vocabulary for specifying formatting semantics.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

XSL is a language for expressing stylesheets. Given a class of arbitrarily structured XML documents or data files, designers use an XSL stylesheet to express their intentions about how that structured content should be presented; that is, how the source content should be styled, laid out, and paginated onto some presentation medium, such as a window in a Web browser or a hand-held device, or a set of physical pages in a catalog, report, pamphlet, or book.

An XSL stylesheet processor accepts a document or data in XML and an XSL stylesheet and produces the presentation of that XML source content that was intended by the designer of that stylesheet. There are two aspects of this presentation process: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, in speech, or onto other media. The first aspect is called tree transformation and the second is called formatting.

The process of formatting is performed by the formatter. This formatter may simply be a rendering engine inside a browser.

Tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

Formatting is enabled by including formatting semantics in the result tree. Formatting semantics are expressed in terms of a catalog of classes of formatting objects. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of formatting objects and formatting properties provide the vocabulary for expressing presentation intent.

The XSL processing model is intended to be conceptual only. An implementation is not mandated to provide these as separate processes. Furthermore, implementations are free to process the source document in any way that produces the same result as if it were processed using the conceptual XSL processing model. A diagram depicting the detailed conceptual model is shown below.

XSL is Three Languages

XSL actually consists of three languages:
- XSLT is a language to **transform XML** into other types of documents, or into other XML documents.
- XPath is a language to **define XML parts or patterns** by addressing parts of an XML document. XPath was designed to be used by XSLT.
- XSL Formatting Objects is a language to **define XML display** Formatting is the process of turning the result of an XSL transformation into a suitable output form for a reader or listener.

XSLT and XPath were released as two separate W3C Recommendations 16. November 1999. No separate W3C document exists for XSL Formatting Objects, but a description can be found inside the XSL 1.0 Recommendation

XSL - More than a Style Sheet

XSL consists of three parts:
- a method for **transforming XML documents**
- a method for **defining XML parts and patterns**
- a method for **formatting XML documents**

If you don't understand the meaning of this, think of XSL as a language that can **transform** XML into HTML, a language that can **filter and sort** XML data, a language that can **address parts** of an XML document, a language that can **format** XML data based on the data value, like displaying negative numbers in red, and a language that can **output** XML data to different devices, like screen, paper or voice.

Because XML **does not use predefined tags** (we can use any tags we want), the meanings of these tags are **not understood**: <table> could mean an HTML table or maybe a piece of furniture. Because of the nature of XML, the browser **does not know how to display** an XML document.

In order to display XML documents, it is necessary to have a mechanism to describe how the document should be displayed. One of these mechanisms is CSS, but XSL (the e**X**tensible **S**tylesheet **L**anguage) is the preferred style sheet language of XML, and XSL is far more sophisticated than the CSS used by HTML.

Microsoft's Internet Explorer 5.0 is, at present, the only available browser for XSL.

XSLT 1.0

Status: W3C Recommendation
Date: 16 Nov. 1999

http://www.w3.org/TR/xslt

XSLT is a language for transforming XML documents into other XML documents.

XSLT is designed for use as part of XSL.

XSLT is the most important part of the XSL Standard. It is the part of XSL that is used to transform an XML document into another XML document, or another type of document.

XSLT can be used to transform an XML document into a format that is recognizable to a browser. One such format is HTML. Normally XSLT does this by transforming each XML element into an HTML element.

XSLT can also add completely new elements into the output file, or remove elements. It can rearrange and sort the elements, and test and make decisions about which elements to display, and a lot more.

A common way to describe the transformation process is to say that XSL uses XSLT to transform an XML **source tree** into an XML **result tree** (or an XML **source document** into an XML **result document**)

In addition to XSLT, XSL includes an XML vocabulary for specifying formatting.

XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

How does it work?
In the transformation process, XSLT uses XPath to define parts of the source document that **match** one or more predefined **templates**. When a match is found, XSLT will **transform** the matching part of the **source** document into the **result** document. The parts of the source document that do not match a template will (as a general rule) end up unmodified in the result.

A transformation in the XSLT language is expressed as a well-formed XML document conforming to the XML Namespaces, which may include both elements that are defined by XSLT and elements that are not defined by XSLT.

XSLT-defined elements are distinguished by belonging to a specific XML namespace, which is referred to in this specification as the XSLT namespace.

A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely

different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.

A transformation expressed in XSLT is called a stylesheet. This is because, in the case when XSLT is transforming into the XSL formatting vocabulary, the transformation functions as a stylesheet.

The XSLT recommendation does not specify how an XSLT stylesheet is associated with an XML document. It is recommended that XSL processors support the mechanism described in http://www.w3.org/TR/xml-stylesheet/ When this or any other mechanism yields a sequence of more than one XSLT stylesheet to be applied simultaneously to a XML document, then the effect should be the same as applying a single stylesheet that imports each member of the sequence in order

A stylesheet contains a set of template rules. A template rule has two parts: a pattern which is matched against nodes in the source tree and a template which can be instantiated to form part of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

A template is instantiated for a particular source element to create part of the result tree. A template can contain elements that specify literal result element structure. A template can also contain elements from the XSLT namespace that are instructions for creating result tree fragments. When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates. Instructions can select and process descendant source elements. Processing a descendant element creates a result tree fragment by finding the applicable template rule and instantiating its template. Note that elements are only processed when they have been selected by the execution of an instruction. The result tree is constructed by finding the template rule for the root node and instantiating its template.

A single template by itself has considerable power: it can create structures of arbitrary complexity; it can pull string values out of arbitrary locations in the source tree; it can generate structures that are repeated according to the occurrence of elements in the source tree. For simple transformations where the structure of the result tree is independent of the structure of the source tree, a stylesheet can often consist of only a single template, which functions as a template for the complete result tree. Transformations on XML documents that represent data are often of this kind

XSLT makes use of the expression language defined by XPath for selecting elements for processing, for conditional processing and for generating text.

XSLT provides two "hooks" for extending the language, one hook for extending the set of instruction elements used in templates and one hook for extending the set of functions used in XPath expressions. These hooks are both based on XML namespaces. This version of XSLT does not define a mechanism for implementing the hooks.

> NOTE: The XSL WG intends to define such a mechanism in a future version of this specification or in a separate specification

The XSLT namespace has the URI http://www.w3.org/1999/XSL/Transform

Normally an XSLT stylesheet is a complete XML document with the xsl:stylesheet element as the document element. However, an XSLT stylesheet may also be embedded in another resource. Two forms of embedding are possible:

1. the XSLT stylesheet may be textually embedded in a non-XML resource, or
2. the *xsl:stylesheet* element may occur in an XML document other than as the document element.

To facilitate the second form of embedding, the *xsl:stylesheet* element is allowed to have an ID attribute that specifies a unique identifier.

The data model used by XSLT is the same as that used by XPath with the additions described in section 3 of the XSLT Recommendation. XSLT operates on source, result and stylesheet documents using the same data model. Any two XML documents that have the same tree will be treated the same by XSLT.

XSLT uses the expression language defined by XPath. Expressions are used in XSLT for a variety of purposes including:
- selecting nodes for processing;
- specifying conditions for different ways of processing a node;
- generating text to be inserted in the result tree.

An expression must match the XPath production Expr.

Missing features which may come with XSLT 2.0

The following features are under consideration for versions of XSLT after XSLT 1.0:

- a conditional expression;
- support for XML Schema datatypes and archetypes;
- support for something like style rules in the original XSL submission;
- an attribute to control the default namespace for names occurring in XSLT attributes;
- support for entity references;
- support for DTDs in the data model;
- support for notations in the data model;
- a way to get back from an element to the elements that reference it (e.g. by IDREF attributes);
- an easier way to get an ID or key in another document;
- support for regular expressions for matching against any or all of text nodes, attribute values, attribute names, element type
- names;
- case-insensitive comparisons;
- normalization of strings before comparison, for example for compatibility characters;
- a function string resolve(node-set) function that treats the value of the argument as a relative URI and turns it into
- an absolute URI using the base URI of the node;
- multiple result documents;
- defaulting the select attribute on xsl:value-of to the current node;
- an attribute on xsl:attribute to control how the attribute value is normalized;
- additional attributes on xsl:sort to provide further control over sorting, such as relative order of scripts;
- a way to put the text of a resource identified by a URI into the result tree;
- allow unions in steps (e.g. foo/(bar|baz));
- allow for result tree fragments all operations that are allowed for node-sets;
- a way to group together consecutive nodes having duplicate subelements or attributes;
- features to make handling of the HTML style attribute more convenient.

XML Fragment Interchange

Status: Candidate Recommendation
Date: 12 February 2001

http://www.w3.org/TR/xml-fragment

The XML standard supports logical documents composed of possibly several entities. It may be desirable to view or edit one or more of the entities or parts of entities while having no interest, need, or ability to view or edit the entire document. The problem, then, is how to provide to a recipient of such a fragment the appropriate information about the context that fragment had in the larger document that is not available to the recipient.

In the case of many XML documents, it is suboptimal to have to receive and parse the entire document when only a fragment of it is desired. If the user asked to look at chapter 20, one shouldn't need to parse 19 whole chapters before getting to the part of interest. The goal of this activity is to define a way to enable processing of small parts of an XML document without having to process everything up to the part in question. This can be done regardless of whether the parts are entities or not, and the parts can either be viewed immediately or accumulated for later use, assembly, or other processing.

Conceptually, the holder of the complete source document considers a fragment of that document and, using the notation to be defined by this activity, constructs a fragment context specification. The object representing the fragment removed from its source document is called the fragment body. The fragment context specification and the fragment body are transmitted to the recipient.

The storage object in which the fragment body is transmitted is call the fragment entity. (In some packaging schemes, the fragment context specification may also be embedded in the fragment entity.)

The recipient processes the fragment context specification to determine the proper parser state for the context at the beginning of the fragment and uses that information to enable the XML parser to parse the fragment body.

> NOTE - The terms “sender,” “recipient,” “transmit,” are used throughout this document to describe the process of fragment interchange. It should be noted, however, that there are many feasible and useful scenarios for fragment interchange, and in some cases, the “sender” and “recipient” may be on the same machine, node, system, or network, and may even be the same tool in different guises.

The challenge is that an isolated element from an XML document may not contain quite enough information to be parsed correctly.

The goal of this activity is to enable senders to provide the remaining information required so that systems can interchange any XML elements they choose, from books or chapters all the way down to paragraphs, tables, footnotes, book titles, and so on, without having to manage each as a separate entity or having to risk incorrect parsing due to loss of context.

The goal of this activity is to define a way to send fragments of an XML document—regardless of whether the fragments are predetermined entities or not—without having to send all of the containing document up to the part in question. The delivered parts can either be viewed or edited immediately or accumulated for later use, assembly, or other processing; what the receiving

application does with the information, and issues involved with the possible “return” of such a fragment to the original sender, is beyond the scope of this activity.

While implementations of this Recommendation may serve as part of a larger system that allows for “fragment reuse,” the many important issues about reuse of XML text and “concurrent multiple author environments” are beyond the scope of this Recommendation.

The point of the fragment context information is to provide information that is not available in the fragment body itself but that would be available from the complete XML document. Specifically, any information not available from the XML document (which may include an external subset) as a whole (plus knowledge of the location of the fragment body within the document) is out of scope for inclusion in the fragment context information. Such information may well be useful and important metadata in a variety of applications, but there are (or need to be) other mechanisms for handling this information.

To accomplish these ends, this Recommendation defines:
- exact constraints on what portions of an XML document may constitute fragments to be supported by this Recommendation;
- the set of information (fragment context information) that allows for successful parsing as well as for viewing or editing of a fragment in a useful and important set of cases;
- the notation (i.e., language) in which this information will be described (the fragment context specification);
- some mechanisms for associating this information with a fragment.

Design Principles (Non-Normative)

In the design of any language, trade-offs in the solution space are necessary. To aid in making these trade-offs the follow design principles were used (the order of these principles is not necessarily significant):
- XML fragment specifications should be usable over the internet.
- XML fragment specifications should support the specification of context for any well-formed chunk of XML; the definition of a fragment may be broadened to allow any chunk of XML that matches XML's “content” production (production [43]). Chunks of XML that do not match XML's “content” production (i.e., that are not well-formed entities) are specifically out of scope.
- XML fragment specifications should be optimized to work with simpler XML fragments (such as those conforming to the simpler XML profile being developed by the XML Syntax WG), though the language should also work with any XML (“the easy stuff should be easy, and the harder stuff should be possible”); working with SGML features not included in XML (including those, such as tag omission, allowed in HTML) is not a goal.
- XML fragment specifications should be capable of being specified both in the same storage object as the fragment body itself as well as in a separate object linked in some fashion to the fragment body.
- XML fragment specifications should support interaction with XML browsers, editors, repositories, and other XML applications.
- SGML features and characteristics not included in XML shall not be taken into consideration in the design of our fragment context specification solution.

- It is specifically not a goal that XML fragment specifications be designed in consideration of non-XML HTML browsers, parsers, or other non-XML applications.

- Since interoperability is a primary goal, there should be only one language for the fragment context specification rather than multiple "features." However, since the goal is to provide enough information to parse the fragment, and well-formed XML may not require any extra information to allow it to be parsed, no specific set of context information should be required in all context specifications. (No implementation should choke on any valid piece of context information, but no implementation should be considered non-compliant for choosing to ignore [on the receiving end]—or not include [on the sending end]—a specific piece of context information if doing so makes sense in the particular environment.)
- XML fragment specifications should leverage other recommendations and standards, including XML 1.0, XML Namespace, XPointer, XML Information Set, the SGML Open TR9601:1996 on Fragment Interchange, and relevant IETF work.
- XML fragment specifications should be human-readable and reasonably clear.
- Terseness in XML fragment specification syntax is of minimal importance.
- Issues involved with the possible "return" of any fragment to its original context and the determination of the possible validity of the "returned" fragment in its original context are beyond the scope of this activity.