FpML.org
Financial products Markup Language

FpML Architecture Version 1.0
Recommendation 16 March 2001

This version:

   http://www.fpml.org/spec/2001/rec-fpml-arch-1-0-2001-03-16

Latest version:

   http://www.fpml.org/spec/fpml-arch-1-0

Previous version:

   http://www.fpml.org/spec/2000/tr-fpml-arch-1-0-2000-12-21

Copyright © 1999, 2000, 2001. All rights reserved.

Financial Products Markup Language is subject to the Mozilla Public License Version 1.0.

A copy of this license is available at http://www.fpml.org/documents/license.

Status of this Document:

This is the FpML Architecture Version 1.0 Recommendation, and has been reviewed by FpML.org members and other interested parties and has been endorsed by the FpML Standards Committee as an FpML Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. FpML.org's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment.

Comments on this specification may be sent to fpml-issues@yahoogroups.com. An archive of the comments is available at http://groups.yahoo.com/group/fpml-issues/messages.

Public discussion of FpML takes place on the FpML Discussion List at http://groups.yahoo.com/group/fpml-discuss.

The errata list for this specification is available at http://groups.yahoo.com/group/fpml-issues/files/rec-fpml-arch-1-0-2001-03-16-errata.html.

A list of current FpML Recommendations and other technical documents can be found at http://www.fpml.org/spec.

This document was produced by the Architecture Working Group, with feedback from the Products Working Group, as part of the FpML Version 1.0 Activity. The FpML Board of Directors initiated this Activity in November 1999 to produce a limited scope standard covering trade content definitions for interest rate derivatives, and a more general architecture for FpML.

Authors

Nic Fulton	– Reuters
Andrew Jacobs	– IBM
Aidan Killian	– Concordia Net
John O'Sullivan	– Chase Manhattan Bank
Robert Silver	– Deutsche Bank
Ashutosh Tripathi	– J.P. Morgan

Acknowledgements

1   Summary

This is the FpML Architecture Version 1.0 Recommendation.

This document defines the standard and provides some additional information about the reasoning behind the standard. In addition, this document highlights important changes since FpML v1.0 b2, and provides information on anticipated additions and changes for the future.

The FpML 1.0 Products Working Group is the first of many working groups that will use this Architectural Standard.

2 Table of Contents

3 Introduction

This is the FpML Architecture Version 1.0 Recommendation.   In addition to defining the standard, this document outlines the principles responsible for the architecture.

The FpML 1.0 Architecture builds upon the earlier FpML v1.0b2 standard. Similarly, the FpML 1.0 Architecture will evolve in response to availability of new standards, the evolution of XML, and new requirements as the scope of FpML expands.

FpML v1.0b2 had a distinctive look and feel. However, the corresponding architecture documentation did not explicitly document the architecture that results in the look and feel. In contrast, this document explicitly defines the FpML 1.0 architecture.

A clear goal was to build upon the FpML v1.0b2 architecture. Despite this goal, there are several significant changes from the FpML v1.0b2 to the FpML 1.0 Architecture. In each case there were specific reasons for the changes. For example, removal of namespaces due to a contradiction between namespace aware and namespace unaware DTD based parsers.

Finally, the current Architecture answers several important questions posed in section 3.3.1 of the v1.0b2 documentation, that is:

> *What is missing in v1.0b2?*

> *4) References within the FpML*

> *Exactly how references will be represented and used needs to be decided. Questions such as, should xptr be used? Should the use of forward references be restricted? Currently we represent references by name, but a more formalized approach needs to be defined.*

> *5) Reference Data*

> *Exactly how reference data will be utilized needs to be made explicit. For instance, referring to ISO codes that are accessible to everyone by name should be acceptable. However, other cases, for instance counterparty names, are not so straightforward.*

> *9) Versioning of DTDs*

> *As FpML is extended, and later versions are produced, some mechanism for doing source control for the DTDs and a process whereby various parties can ensure that they are using compatible versions needs to be established.*

3.1 Significant Changes

This section highlights the significant changes between the current Architecture Standard and the FpML v1.0b2.

- Introduction of an explicit Content Model. The model:

  - moves the representation of the FpML object model from the XML to the DTD (or Schema in the future). The instantiation of the model remains in the XML.

  - improves re-use through Entities within DTD. Again, equivalent functionality will be provided in the future for Schemas.

  - defines the representation of inheritance, containers, arrays, etc.

  - improves the *signal to noise ratio*. That is, the same information content generally requires fewer XML elements.

- Introduction of version identification, with a specific mechanism to identify the version. Software can inspect FpML 1.0, or any future versions, and unambiguously determine the version.

- Elimination of Namespaces. Here, specific technical problems required the removal of namespaces.   Future versions of FpML may re-introduce namespaces.

- Introduction of Schemes as a general mechanism for identification of external standards.

- Changes to both internal and external references. The FpML v1.0b2 mechanism for internal referencing was replaced with a XLink/XPointer compatible mechanism.

- Restructuring of the FpML DTD into significantly fewer separate files.

3.2   Architectural Principles

The Overview to the FpML Working Draft v1.0b2 documentation has two sections devoted to architecture:

- *Section 2.2 Architectural goals*

- *Section 4 Architecture*

The documents are available at:

> http://www.fpml.org/documents/standard/FpML-10b2/FpML-10b2.html

and remain highly applicable today.

In addition, the following principles helped shape FpML 1.0's Architecture.

- It must be feasible to implement FpML 1.0 with current technology and standards. As an example, FpML 1.0 is a DTD based and not Schema based standard.

- Industry standards should prevail over FpML specific solutions. Thus, the internal referencing mechanism of FpML v1.0b2 was retired and replaced with a mechanism that is more closely aligned with XLink/XPointer.

  In contrast, the FpML versioning standard is an FpML specific solution, because no existing external standard could be identified.

- FpML 1.0 represents a unique opportunity to make substantial changes to FpML should they be necessary.

  The Content Model results in changes to the Look and Feel of FpML. Changes of similar magnitude will be much more difficult in the future.

- Going forwards, the goal should be to preserve FpML's XML representation. In contrast, we expect to make significant changes to DTDs and ultimately drop them, while minimizing the impact on the XML.

There is a general reduction in the verbosity of FpML (i.e., compared to FpML v1.0b2, the same information content generally requires fewer XML elements). It is significant to note that this reduction is not due to a specific goal of reducing the verbosity of FpML. Instead, it is due to the introduction of the Content Model, as detailed in Section 4.

The following specifically address comments in the FpML v1.0b2 documentation:

- The FpML v1.0b2 description of design tradeoffs regarding validation, remain applicable for FpML 1.0. Specifically, FpML is DTD based and cannot use the more sophisticated features provided by Schemas.

- The FpML v1.0b2 documentation describes the advantages of re-use. The FpML architecture introduces a DTD based mechanism for such code re-use.

- No work has been done to build on FpML v1.0b2's concept of views, because there was no identified need for views for the trade content focus of FpML 1.0. The current Architecture document does not preclude the introduction of views in the future. However, views should only be introduced if there is a specific need for the functionality.

3.3   Future

In the future, we expect to see additions and changes to the FpML Architecture, including:

- expansion of Versioning beyond its current role of identifying the FpML version. This includes standards for what changes are permissible between versions, and standards for interoperability between versions. It is important to allow FpML to evolve over time, and these standards need to be in place prior to the introduction of the FpML version that supersedes FpML 1.0.

- architectural support for equivalence.

- architectural support for extensions.

- changes to keep up with XML developments, in particular introduction of Schemas.

- changes to keep up with external standards.   In turn, potentially retiring FpML specific features in favor of standards (as was done with references).

- additional features needed to support specific Working Groups that are tasked with expanding the product scope, or overall functionality of FpML.

4   Content Model

This section defines the FpML 1.0 style for structuring DTDs. The approach defines how the basic elements of object oriented design would be mapped to XML DTD entities and elements. A simple example is given for each transformation.

In addition, the mappings result in XML documents that are consistent with the future use of XML Schemas. Examples of the equivalent Schema definition for each mapping have been provided.

4.1   Introduction

Broadly speaking, FpML v1.0b2 used an object-oriented approach for data modeling. The resulting model was subsequently converted to an XML DTD definition by a relatively simple and direct content mapping.

This section defines a new model for this transformation which results in less verbose XML messages at the cost of a slightly more complex set of DTD definitions. It must be stressed that the content model described in this section can not be fully utilized without the use of a validating parser that provides access to element attributes and their default values.

The rules in this section have been based on the following XML Schema Candidate Recommendations:

- XML Schema Part 0: Primer

  http://www.w3.org/TR/2000/CR-xmlschema-0-20001024/

- XML Schema Part 1: Structures

  http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/

- XML Schema Part 2: Datatypes

  http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/

4.2   Naming Convention

With the exception of Entities, names must not use any name scoping mechanism that is inconsistent with the XML Namespaces Recommendation. See Section 4.13 for additional details. For technical reasons, XML Namespaces cannot be used in FpML 1.0. Overall, scoping mechanisms cannot be used for elements and similar names. Section 7, Namespaces, provides additional details.

The XML Namespaces Recommendation can be located at:

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

All established or well-known acronyms will be named as defined e.g., FpML. All other acronyms will be named according to the rules below.

All element and attribute names in FpML Schemata must follow the convention of:

- First word or acronym in lower case.

- All following words or acronyms with upper case first letter.

- All other characters are lower case, including the remaining characters or acronyms.

For example:

- currency

- tradeId

- currencyScheme

In contrast, all entities in FpML will use all upper CamelCase. For example:

```
<!ENTITY % Shape "x, y">
```

As noted above, entities can use a name scoping mechanism. The reason for this are explained in section 4.13, Entities and Namespaces, below.

All type or class names should follow the convention of:

- All primitive data types as specified by the XML Schema Part 2: Datatypes W3C Candidate Recommendation 24 October 2000

- For all other types:

  - First letter of each word or acronym in upper case.

  - All other characters in lower case, including the remaining characters or acronyms.

4.3   Primitive Datatypes
FpML uses the built-in data types as defined in XML Schema Part 2: Datatypes W3C Candidate Recommendation 24 October 2000.

The built-in data types are described at:

http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/#built-in-datatypes

The size and other properties of these data types are specified using the constraining facets for them described in the Candidate Recommendation above.

Examples are:

http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/#dt-length
http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/#dt-precision
http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/#dt-scale

4.4 Classes

In object-oriented design, a class defines a common set of data attributes and functionality that will be present in any instance of the class. Because XML is not procedural, only the data attributes of a class are mapped to XML.

Money
currency:string amount:decimal

Note that the original FpML content model would have mapped this class to three XML elements, one for the class as a whole and two for the member attributes. The new content model represents this somewhat differently, namely:

- An XML entity is created to hold this list of members of the class. The entity name is derived from the class name rendered in CamelCase (e.g., Money, BusinessCenters, etc.).

- Elements are defined for each member attribute with appropriate meta-data attributes (e.g., type).

- An element is also created to expand the member list entity for the class.

The complete XML definition for this simple class would therefore be as follows:

```
<!-- Member list entity (used for inheritance and instantiation) -->
<!ENTITY % Money "currency, amount">

<!-- Member declarations -->
<!ELEMENT currency (#PCDATA)>
<!ATTLIST currency type NMTOKEN #FIXED "string">
<!ELEMENT amount (#PCDATA)>
<!ATTLIST amount type NMTOKEN #FIXED "decimal">
```

The following XML is used to create an appropriate element and define its type, where the 'Money' class needs to be instantiated:

```
<!-- Class type element -->
<!ELEMENT premium (%Money;)>
<!ATTLIST premium type NMTOKEN #FIXED "Money">
```

These rules avoid the use of class name tags.

XML Schemas support the notion of type directly and will allow for a far simpler mapping in the future. For example, the equivalent XML Schema for the DTD above is:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="currency" type="xsd:string"/>
<xsd:element name="amount" type="xsd:decimal"/>
<xsd:complexType name="Money">
    <xsd:sequence>
        <xsd:element ref="currency"/>
        <xsd:element ref="amount"/>
    </xsd:sequence>
</xsd:complexType>

<!-- Matching example element -->
<xsd:element name="premium" type="Money"/>

</xsd:schema>
```

Both the DTD and Schema definitions allow the following XML instance to be valid.

```
<premium>
    <currency>GBP</currency>
    <amount>1000000.00</amount>
</premium>
```

The semantics of the type attribute are those of the attribute 'type' as defined in:

http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/

The namespace is defined in:

xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"

as seen in the example above. When FpML moves to XML Schema, this attribute is expected to move into this namespace.

4.5   Inheritance

Inheritance allows for extending the definition of a base class by providing it with additional members. For future compatibility with Schemas, only single inheritance is permitted.
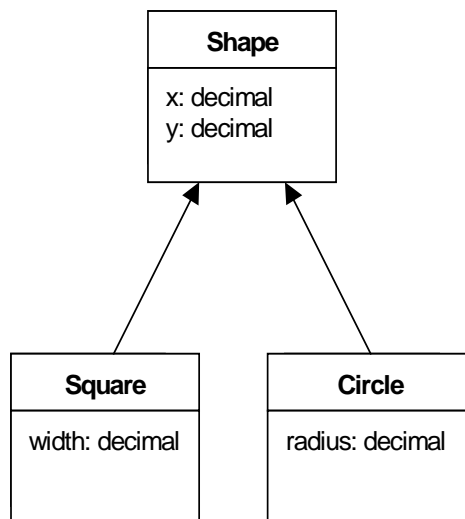
In order to represent the sub-types, the XML definitions should create new member list entities (derived from the parent classes member lists) for each new class.

```
<!-- Member list entities -->
<!ENTITY % Shape "x, y">
<!ENTITY % Circle "(%Shape;), radius">
<!ENTITY % Square "(%Shape;), width">

<!-- Member declarations -->
<!ELEMENT x (#PCDATA)>
<!ATTLIST x type NMTOKEN #FIXED "decimal">
<!ELEMENT y (#PCDATA)>
<!ATTLIST y type NMTOKEN #FIXED "decimal">
<!ELEMENT radius (#PCDATA)>
<!ATTLIST radius type NMTOKEN #FIXED "decimal">
<!ELEMENT width (#PCDATA)>
<!ATTLIST width type NMTOKEN #FIXED "decimal">
```

4.6   Abstract Types

An abstract type is a type that can not be instantiated. There is no mechanism for representing
an abstract type in an XML DTD.   Therefore, the only indication of an abstract type is an entity
containing a member list that is not referenced by any element.

A subtype of an abstract type must also be declared as an element with one attribute to declare
its type or class name, and another attribute to declare its base type. Note that XML Schemas
have a mechanism to derive a base type by extension or restriction. This model only allows
derivation by extension of the base type.

```
<!ELEMENT circle  (%Circle;)>
<!ATTLIST circle type NMTOKEN #FIXED "Circle"
                 base NMTOKEN #FIXED "Shape">
<!ELEMENT square  (%Square;)>
<!ATTLIST square  type NMTOKEN #FIXED "Square"
                  base NMTOKEN #FIXED "Shape">

<!ENTITY % Rhombus "(%Square;), skew">
<!ELEMENT rhombus  (%Rhombus;)>
<!ATTLIST rhombus  type NMTOKEN #FIXED "Rhombus"
                   base NMTOKEN #FIXED "Square">
```

4.7 Aggregation

Like association, aggregation is a relationship between two objects. However, unlike
association, the aggregated object does not have an independent existence. In the following
example the squareFrame is represented as an aggregation based on the Square and Shape
classes.
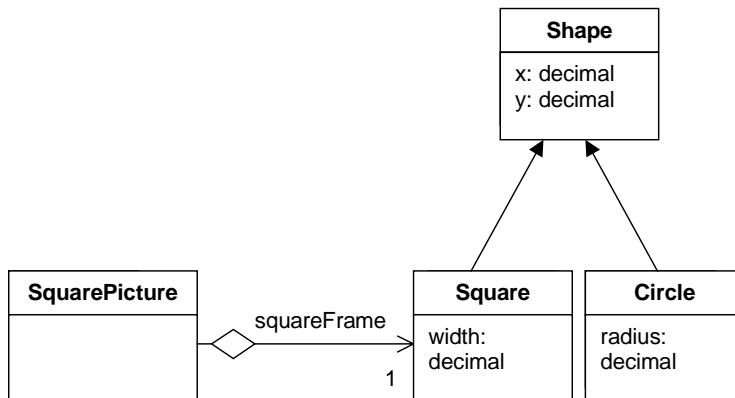
Within the DTD aggregation is modeled by expanding the member list of the aggregated class
within an element representing its role in the containing object.

Applying this transformation to the example diagram shown produces the following XML DTD
fragment:

```
<!-- Example square instantiation -->
<!ELEMENT squareFrame (%Square;)>
<!ATTLIST squareFrame type NMTOKEN #FIXED "Square">
```

If multiple instances can be aggregated, then the tags can be repeated. The role can be used
to differentiate aggregations of the same underlying class into the container.

4.8   Aggregation of Sub-Types

Combining the definitions seen so far allows us to manage the case when the aggregated instances are subtypes of a common base type.
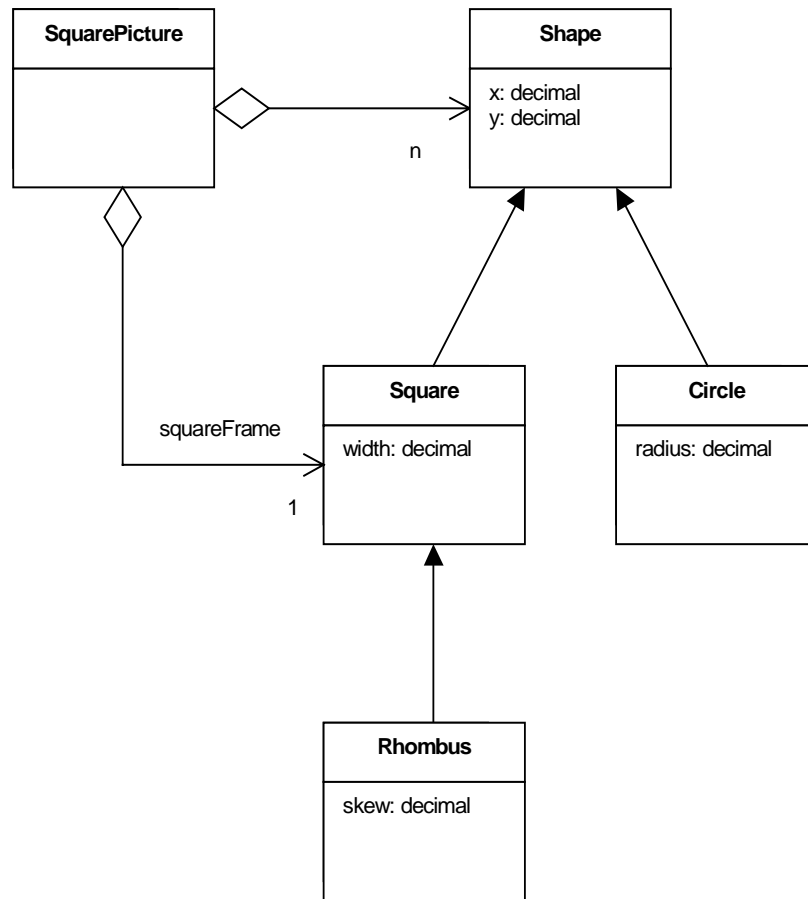
A further entity is provided to list all valid instantiable subtypes in the form of a list of elements describing those subtypes. Any element that accepts any subtype of a base type must be declared using this entity.

This entity can only include instantiable types in the type hierarchy and must not include any abstract base type.

```
<!-- Instantiable types -->
<!ENTITY % ShapeType "circle | square | rhombus">

<!-- Example shape instantiation -->
<!ELEMENT squarePicture (squareFrame, (%ShapeType;)+)>
```

The following XML fragment illustrates how this would appear when, apart from the square frame, there are a number of shapes in this square picture.

```
<squarePicture>
    <squareFrame>
        <x>1.0</x>
        <y>2.0</y>
        <width>3.0</width>
    </squareFrame>
    <circle>
        <x>1.0</x>
        <y>2.0</y>
        <radius>3.0</radius>
    </circle>
    <square>
        <x>1.0</x>
        <y>2.0</y>
        <width>3.0</width>
    </square>
    <rhombus>
        <x>1.0</x>
        <y>2.0</y>
        <width>3.0</width>
        <skew>4.0</skew>
    </rhombus>
</squarePicture>
```

The following is the equivalent Schema example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="x" type="xsd:decimal"/>
<xsd:element name="y" type="xsd:decimal"/>
<xsd:complexType name="Shape">
    <xsd:sequence>
        <xsd:element ref="x"/>
        <xsd:element ref="y"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="shape" type="Shape" abstract="true"/>
<xsd:element name="radius" type="xsd:decimal"/>
<xsd:complexType name="Circle">
    <xsd:complexContent>
        <xsd:extension base="Shape">
            <xsd:element ref="radius"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="circle" type="Circle" substitutionGroup="shape"/>

<xsd:element name="width" type="xsd:decimal"/>
<xsd:complexType name="Square">
    <xsd:complexContent>
        <xsd:extension base="Shape">
            <xsd:element ref="width"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="square" type="Square" substitutionGroup="shape"/>

<xsd:element name="skew" type="xsd:decimal"/>
<xsd:complexType name="Rhombus">
    <xsd:complexContent>
        <xsd:extension base="Square">
            <xsd:element ref="skew"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="rhombus" type="Rhombus" substitutionGroup="shape"/>

<xsd:complexType name="SquarePicture">
    <xsd:sequence>
        <xsd:element name="squareFrame" type="Square"/>
        <xsd:element ref="shape" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="squarePicture" type="SquarePicture"/>

</xsd:schema>
```

In this example we have deliberately chosen to define the shape element to be abstract instead of setting it as a characteristic of the underlying type. This makes the mapping to XML Schemas simpler and avoids the need to use a choice group that would otherwise be required.

4.9   Association

An association represents a relationship between two independent objects. The relationship normally has a 'role' that identifies the purpose to which the associated object is put.
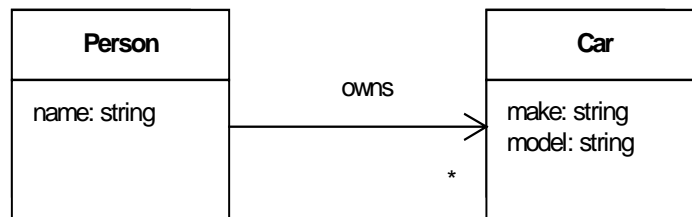
Within the DTD, the 'owns' association is mapped to an element which references the required Car object. In this example, a simple string URI holds the reference. However, in the FpML Schema an XLink should be used.

Both Person and Car in the above model are 'top level' classes. They have been mapped to elements in the following DTD fragment:

```
<!-- Member list entities -->
<!ENTITY % Person "name, owns*">
<!ENTITY % Car "make, model">

<!-- Class elements (as person and car are top level classes) -->
<!ELEMENT person (%Person;)>
<!ATTLIST person type NMTOKEN #FIXED "Person">
<!ELEMENT car (%Car;)>
<!ATTLIST car type NMTOKEN #FIXED "Car">

<!-- Member declarations -->
<!ELEMENT name (#PCDATA)>
<!ATTLIST name type NMTOKEN #FIXED "string">
<!ELEMENT owns EMPTY>
<!ATTLIST owns href CDATA #REQUIRED>
<!ELEMENT make (#PCDATA)>
<!ATTLIST make type NMTOKEN #FIXED "string">
<!ELEMENT model (#PCDATA)>
<!ATTLIST model type NMTOKEN #FIXED "string">
```

The XML will contain repeating tags when an association allows multiple instances to be associated. Note that the following XML fragment assumes that the referenced cars are

described in the document "cars.xml".

```
<?xml version="1.0"?>
<!DOCTYPE Person SYSTEM "…">

<person>
    <name>John Smith</name>
    <owns href="cars.xml#XYZ1234"/>
    <owns href="cars.xml#ABC5678"/>
</person>
```

The following is the equivalent Schema example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:complexType name="Person">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="owns" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:attribute name="href" use="required"
                                type="xsd:uriReference"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Car">
    <xsd:sequence>
        <xsd:element name="make" type="xsd:string"/>
        <xsd:element name="model" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="person" type="Person"/>
<xsd:element name="car" type="Car"/>

</xsd:schema>
```

4.10 Arrays (Sequences)

There are two basic approaches for the handling of arrays or sequences of repeating elements in XML.   Outside of FpML, the two basic approaches are subject to an ongoing debate. The approaches are:

- Simply repeat the required element as many times as necessary between its sibling elements. This leads to the simplest XML definitions.

- Define a container to hold the repeating element and use the container in the parent element. This assists in message building and parsing.

Unfortunately most XML book and paper authors use both styles interchangeably in their texts and hence are unreliable references.

The examples in this section so far have treated arrays as a simple repeating element within the containing master element. This is the simplest and most direct manner in which to represent association or aggregation within XML and accurately represents the following model.
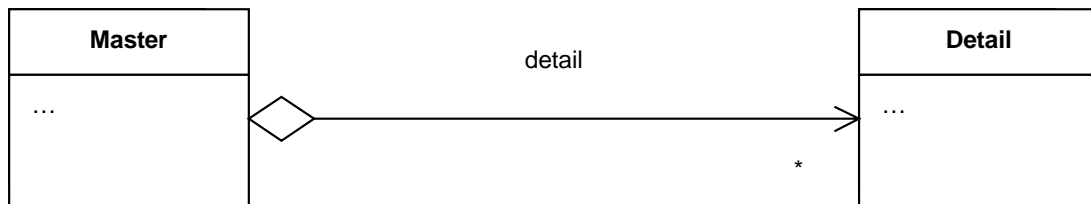
Prior to FpML 1.0, the FpML DTD files defined a number of container elements that were used to group the detail elements together. A similar approach is used by the published XML DTDs (e.g., SOAP, FixML, and BizTalk.). However, in all these cases the container not only provides grouping, but also carries attributes or elements of its own describing the collection as a whole (e.g., in SOAP arrays must have a xsd:type attribute; in FixML an OrderList has a ListNoOrders attribute). Currently the FpML containers do not have additional attributes for the array as a whole.

The introduction of collection related information means that we should model the container explicitly within the object model. For example, if the container explicitly holds a count of the number of detail items present then the model becomes as follows:
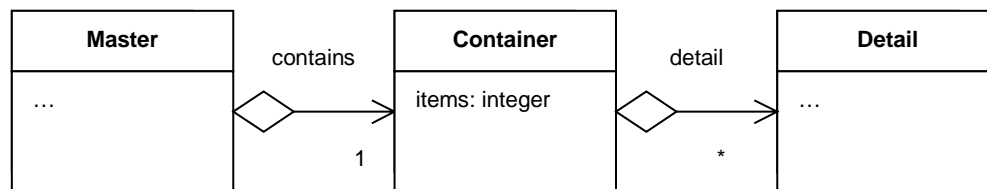
Using the rules established so far, this would yield a model that accepts the following XML:

```
<master>
    …
    <contains>
        <items>2</items>
        <detail>…</detail>
        <detail>…</detail>
    </contains>
    …
</master>
```

We expect the simpler mapping to be appropriate in most cases. FpML XML designers should start with this hypothesis, and should use a container only if the container must be explicitly modeled because either:

●   data analysis shows that it has attributes of its own, independent of the business object that owns it and the objects it contains, or

●   the collection as a whole can be referenced from elsewhere and must be given an identification attribute, independent of the actual data model, to facilitate the reference.

4.11 Compatibility with XML Schemas

The mappings described in this FpML Standard produce XML documents that are consistent with those describable by XML Schemas.

Parts of the Content Model are DTD specific (e.g., the use of entities). XML Schemas directly support the features required to produce a content mapping (e.g., types and inheritance) without resorting to entities.

Any modeling issues which have not been addressed explicitly in this section, should be resolved by referring to the XML Schema concepts. If issues need to be resolved for DTDs, then the resolution should be done in a manner consistent with XML Schema.

4.12 Use of Attributes

This specification follows the FpML v1.0b2 guideline of using elements to contain message data. Attributes are only used to define meta-data for elements.

Elements provide powerful grammatical construction capabilities. These features outweigh the advantages of attributes. That is, defaults and validation of enumerated values and defaults.

4.13 Entities and Namespaces

FpML v1.0b2 used namespaces to identify related definitions. In XML, entity definitions are not scoped in the same way as elements in the presence of namespaces. This effectively means that entity names must be unique within the whole Schema, rather than just being unique within

a namespace. One simple workaround would be to include the namespace as a prefix to the entity name.

```
<!ENTITY % FpML_Money "currency, amount">
```

This mechanism should be used in naming entities defined by FpML, in order to limit the potential for name clashes.

5   References

This section defines two mechanisms for referencing data and domain values within FpML compliant documents.

- XLink/XPointer is the standard technique for creating references between objects residing either in the same document or different documents.

- A standardized Scheme/vocabulary approach is proposed for simple domain values including day count conventions, and currency codes.

5.1   Introduction

An FpML message, such as a swap trade confirmation, comprises product and reference data in a single message. At various points within the product definition for such a message, non-product data objects (e.g., counter-parties, contacts, etc.) are needed to place the product data in the correct context.

Often the same non-product data object is required several times within the message. Two basic strategies can be applied to address this repetition, namely:

- Repeat the whole of the data object every time it is required.

- Include the data object definition once and use references subsequently.

Simple repetition can lead to large XML documents, if the data object is either frequently repeated or if it requires a large number of tags to define. Additionally, the application processing the XML must compare each repeated instance to ensure that they are identical.

A reference requires less XML code than object repetition and ensures consistency between references because there is only one instance definition.

XML 1.0 provides a basic reference mechanism based on the use of ID and IDREF attributes. Elements that may be referenced must be defined with an optional ID attribute, while a referencing element uses an IDREF attribute.

The W3C XLink and XPointer standards define an abstract language for defining the location of XML fragments within a document than can be used wherever a URI reference is required. The XLink standard supports the use of an href attribute to implement HTML style pointers into a document. Its adoption provides compatibility with existing HTML linking practice, and with the XLink standard going forward.

Within an XML DTD the option to select between an object instance or reference can be defined by an appropriate grammatical construction. The following DTD fragment for example shows how this could be modeled for business day adjustments (using the proposed content model mapping style). Note that the enumerated possible values for businessCenter and

businessDayConvention have been removed in favor of the "Schemes" solution to the Domain Values issue detailed below.

Note that the href attribute should have a type of CDATA, to parallel W3C's Xlink, and allow a '#' in the href.

```
<!ELEMENT businessCenter (#PCDATA)>
<!ATTLIST businessCenter type NMTOKEN #FIXED "string"
          businessCenterScheme CDATA #IMPLIED
"http://www.fpml.org/spec/2000/business-center-1-0">

<!ELEMENT businessCenters (businessCenter+)>
<!ATTLIST businessCenters id ID #IMPLIED>

<!ELEMENT businessDayConvention (#PCDATA)>
<!ATTLIST businessDayConvention type NMTOKEN #FIXED "string"
          businessDayConventionScheme CDATA #IMPLIED
"http://www.fpml.org/spec/2000/business-day-convention-1-0">

<!ELEMENT businessCentersReference EMPTY>
<!ATTLIST businessCentersReference href CDATA #REQUIRED>

<!ENTITY % BusinessDayAdjustments "businessDayConvention,
businessCentersReference | businessCenters">

<!ELEMENT businessDayAdjustments (%BusinessDayAdjustments;)>
<!ATTLIST businessDayAdjustments type NMTOKEN #FIXED
"BusinessDayAdjustments">
```

A fragment of an instance document using this DTD follows:

```
<businessCenters id="primaryBusinessCenters">
    <businessCenter>GBLO</businessCenter>
    <businessCenter>USNY</businessCenter>
</businessCenters>

<businessDayAdjustments>
    <businessDayConvention>none</businessDayConvention>
    <businessCentersReference href="#primaryBusinessCenters"/>
</businessDayAdjustments>
```

5.2   Domain Values

Many data elements defined in the FpML DTDs are restricted to holding a limited set of possible values. For example, day count convention, option type, or currency code are all examples of data elements which have a finite set of "legal" values. Such restricted sets of values are frequently referred to as domains, and an individual value in this set is a "domain value".

XML 1.0 has some limited support for the concept of domains through the use of enumerated attributes. For example:

```
<!ELEMENT option (…)>
<!ATTLIST option type (PUT | CALL) #REQUIRED>
```

FpML has adopted the principle of not using attributes to hold business data. As a consequence XML enumerations are not used and an alternative strategy is required.

The alternative strategy is Schemes, that are defined in the section that follows. Schemes allow:

- enumerations to be revised without requiring a re-issue of the FpML DTDs.

- alternate Schemes, without requiring changes to the FpML DTDs.

5.3   Schemes

In the case of an element whose "legal" values are restricted to those of a specific domain, values in that domain should be handled as follows:

- Elements containing domain values should be of string type and will hold a single valid domain value identifier.

- A defaulted attribute, '<schemeName>SchemeDefault', will be provided on the FpML document's root element to define the default Scheme URI. This will be used if no overriding URI is provided.

- An optional attribute, '<schemeName>Scheme', will be provided on each domain valued element to override the URI reference for the domain.

The URI for a Scheme can be categorized as one of the following:

- An external coding Scheme, which has a well-known URI. In this case the URI is assigned by an external body, and may or may not have its own versioning and date syntax and semantics. The external body may be an open standards organization, or it may be a market participant.

- An external coding Scheme, which does not have a well-known URI. In this case FpML assigns a URI as a proxy to refer to the concept of the external Scheme, but this URI will not be versioned or dated.

- An FpML-defined coding Scheme. In this case the Scheme is fully under FpML control and the URI will change reflecting newer versions and revisions as the Scheme evolves and changes.

For the last two categories, the URI is FpML defined. Incorporating a date with the URI name allows for changes in the coding Scheme without having to release a new FpML model. The following XML fragment shows how Schemes will look within a financial message. In this example, the root node defines the Schemes in use by the values given to the attributes currencySchemeDefault and businessCenterSchemeDefault. The value of the currencySchemeDefault attribute is overridden on the currency2 element with a currencyScheme attribute.

```
<FpML currencySchemeDefault="http://www.fpml.org/ext/iso4217"
 businessCenterSchemeDefault="http://www.fpml.org/spec/2000/business-
center-1-0" >
    <trade>
        <fxSwap>
            <nearLeg>
                <currency1>CHF</currency1>
                <currency2
currencyScheme="http://www.chase.com/ext/realmoney"
>UK_Pence</currency2>
                    …
            </nearLeg>
        </fxSwap>
    </trade>
</FpML>
```

The following shows the ATTLIST declarations for the default Scheme and override Scheme attributes. No default values are supplied for either in the DTD. To do so would be to implicitly import a particular version of the Scheme into the DTD – which runs counter to the spirit of the whole Scheme mechanism. The intent of Schemes is to de-couple changes in Schemes from

the versioned changes of the DTD. The default Schemes are #IMPLIED rather than #REQUIRED since not all products require all Schemes. The documentation for each product must specify which default Schemes are compulsory for which products.

```
<!ATTLIST FpML currencySchemeDefault CDATA #IMPLIED
               businessCenterSchemeDefault CDATA #IMPLIED>
<!ATTLIST currency currencyScheme CDATA #IMPLIED>
```

5.4   Specification

XLink/XPointer is the standard for specifying intra and inter document references. For FpML 1.0, XLink/XPointer should be restricted to the simple *bare-name* syntax. Future releases of FpML may lift this restriction in whole, or in part, as the proposal becomes accepted and as XML parsers that implement the advanced features become available.

The XLink mechanism completely replaces the FpML v1.0b2 linking implementation based on matching element values.

Initially FpML will be used to address the exchange of data for confirmation. For this purpose the usage of XPointer must be restricted to being intra-document only to eliminate any ambiguity. That is, references must not be used to reference information not contained in the document itself. For example, a swap trade confirmation cannot reference a counterparty held within the sender's internal trading system.

Rules to prevent the misuse of XPointer in such circumstances cannot be defined within the XML grammar, instead they will have to be present as part of the semantic rules used for message validation and matching.

XPointer may be used in its more general form, if the scope of FpML is broadened to business processes other than confirmation. For example, a trading system might produce trade messages for internal risk management processing that reference counter-parties and securities rather than include them. It would be the responsibility of the receiving system to resolve the references and acquire any information that it needs to complete the trade processing.

The extent to which a reference is resolved depends on the receiver. For example, a pure market risk management system can ignore the counter-party reference entirely. An exposure management system will require some counter-party details and a full credit risk management system will want the entire organization hierarchy.

The current generation of XML parsers does not provide support for XPointer directly. In the short term, some additional coding is required to resolve these references. Some parsers have additional API functions however for using ID values that will greatly simplify the coding. Bare-name XPointer references should be no harder to handle than the FpML v1.0b2 linking Scheme, and in most cases should be far easier to implement.

Schemes should be the standard representation for domain values within FpML documents. Schemes do not provide any parser based validation capability and implementers will need to supply program code to achieve this. Again, this is no more complicated than the convention used in the FpML v1.0b2 version of the standard.

6 Namespaces and URIs

This section specifies the FpML usage of namespaces, and provides design rules for ensuring the version 1.0 FpML specification is compatible with the spirit of namespaces.

6.1 The Spirit of Namespaces

The spirit of namespaces is to avoid name collisions. By avoiding name collisions, elements and attributes from multiple DTDs or Schemata can be mixed in a single XML document instance. The namespace mechanism works by assigning a global identifier, known as the namespace URI, to specify the DTD or Schema, which is then associated with elements as they are used.

This association is done by an arbitrary namespace prefix that is chosen by the author of the XML document. This prefix enables the author to associate the URI with an element or attribute whenever the element or attribute is used. As the URI is in turn associated with a DTD or Schema, the author is, in effect, borrowing the semantics of that original element. By adding a notion of a default namespace, the XML can also remain relatively non-verbose; i.e., some prefixes can be removed.

However, this namespace prefix has no semantic purpose, and is merely a syntactic mechanism to reduce the need to explicitly state the namespace URI on all elements and attributes.

As an example consider the two DTDs below:

```
<!-- ThingA -->
<!-- uri://www.nowhere.org/2000-04-05/DTD/ThingA -->

<!ELEMENT abc (#PCDATA )>
```

And:

```
<!-- ThingZ -->
<!-- uri://www.nowhere.org/2000-04-05/DTD/ThingZ -->

<!ELEMENT xyz (#PCDATA )>
```

Now consider a document, which is not DTD-valid, but is well formed, that uses elements in these two DTDs but mixes the elements up:

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

<i:ijk   xmlns:i="uri:nothing-much"
        xmlns:a="uri://www.nowhere.org/2000-04-05/DTD/ThingA"
        xmlns="uri://www.nowhere.org/2000-04-05/DTD/ThingZ">
    <i:lmn>Content1</i:lmn>
    <a:abc>Content2</a:abc>
    <xyz>Content3</xyz>
</i:ijk>
```

Note that there is a namespace prefix "i" used to associate with "uri:nothing-much" and a second prefix "a" used to associate with "uri://www.nowhere.org/2000-04-05/DTD/ThingA". There is also a default namespace of " uri://www.nowhere.org/2000-04-05/DTD/ThingZ" assigned in the outermost tag, which is applied to any elements that do not have a prefix.

The document could have been rendered differently, but with equivalent structure and content, where only the namespace prefixes and defaulting are different:

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

<ijk   xmlns="uri:nothing-much"
        xmlns:giraffe="uri://www.nowhere.org/2000-04-05/DTD/ThingA"
        xmlns:banana="uri://www.nowhere.org/2000-04-05/DTD/ThingZ">
    <lmn>Content1</lmn>
    <giraffe:abc>Content2</giraffe:abc>
    <banana:xyz>Content3</banana:xyz>
</ijk>
```

These two documents are canonically the same but note that the namespace prefixes are completely different, and appear to be meaningful. However, the prefix is completely arbitrary and could have been "dqkgucnqrstuv" or something in Japanese. It is important to realize that digital signature and content matching systems should sign the canonical version of the XML and not the raw version. [Note: the draft canonicalisation specification has recently undergone a revision and the question of namespace normalisation is still in flux. However, it is advisable to always assume the namespace prefix is arbitrary.]

6.2   Namespaces in FpML

FpML 1.0 does not use namespaces. Additional information on namespaces is provided for future FpML versions. Section 6.4 provides the rationale for not using namespaces for FpML 1.0.

6.3   Design Standards

FpML 1.0 Schemata and content should use the following standards related to the use of Namespaces and URIs:

● FpML elements and attributes shall not use colons.

● A URI shall be specified for groups of related elements and attributes.

● The first FpML version shall have only one specified URI hence all elements and attributes will be grouped together.

● DTDs should not have hard-coded namespace prefixes (as was done in FpML v1.0b2).

● All elements in all FpML 1.0 DTDs shall have unique names in order to avoid collisions.

● Attributes have a scope within elements and therefore do not need to be uniquely named.

● Entities in the DTDs must also have unique names. See section 4.13.

● Element and attribute names must NOT use an alternative namespace mechanism, such as fpml_SwapStream, where *fpml_* represents an alternative namespace mechanism.

● Elements with different meanings should have different names. If elements have the same meaning, then the element should be reused. The same rule applies to entities.

● The attribute xmlns should not be used in any element.

● Schemes, as defined in the previous section, shall be used when the content of an element or attribute contains a value that is based on a non-FpML defined Scheme or vocabulary.

6.4   Removal of Namespaces from FpML 1.0

The FpML v1.0b2 specification made extensive use of XML namespaces. The DTDs worked with many XML parsers. However, when XML 1.0 was released, there was no namespace specification. In situations where the DTDs worked, the parsers treated the namespace prefix and the colon as part of the element name. FpML v1.0b2 had the appearance of being namespace ready. However, the behavior did not fully match the behavior of namespaces. It was not possible to change the namespace prefix, nor was it possible to use the xmlns attribute in a consistent manner.

The introduction of namespace aware parsers caused the contradiction that resulted in the removal of namespaces for FpML 1.0. Consider, for example, <a:thing>. For namespace unaware parsers, <a:thing> corresponds to an element such as
<!ELEMENT a:thing (#PCDATA) >
in the DTD. The element name is "a:thing". However, for namespace aware parsers, <a:thing> corresponds to an element such as
<!ELEMENT thing (#PCDATA)>

where the element name is "thing". The same XML and DTD element definition cannot be used by both namespace aware, and namespace unaware parsers.

Although there are potentially several work-arounds to this contradiction, the cleanest solution was to remove namespaces from FpML 1.0. With the release of the XML Schema specification is if understood that most of these namespace contradictions will be removed.

6.5   Future Compatibility with Namespaces

For the long term, it is important to associate a well-known URI with the DTD elements and attributes. This association is logical and can either be made as a comment in the DTD or in documentation.

6.6   URI recommendations

URIs are used in two situations:

1. For the content model and the sub-parts of the content model.

2. To reference vocabularies and Schemes. See section 5.3 for the three sub-cases of references to vocabularies and Schemes.

FpML 1.0 specifies the structure of the FpML controlled URIs as having six parts:

http://www.fpml.org/AAA/BBBB/CCCC-DDD-EEEE-EE-EE

1. The protocol and server identifier. Following the convention, FpML 1.0 uses "http://www.fpml.org"

    This ensures that the URI will not clash with URIs assigned by other registered entities. For external standards that do not have well known URIs FpML assigns an fpml.org URI.

2. A marker string, "AAA" in this example. This is "spec" for FpML issued specifications and "ext" for external vocabularies and Schemes that are assigned fpml.org URIs because they do not have externally defined URIs. Additional markers may be introduced in the future.

3. On optional year, "BBBB", to help manage documents. This mechanism is used quite widely in the W3C.

4. A string, "CCCC", indicating the name of the particular document. The field consists of lower case letter, and numbers. Minus signs ('-') can also be used, except for the initial or final character.

5. The version, "DDD", indicates the version of the document. The field contains European digits. Minus signs ('-') can also be used, except for the initial or final character, to separate the major and minor version parts. No part of the version number should contain more that 3 digits.

6. An optional date, EEEE-EE-EE. That is year-month number-day number. To be consistent with W3C practices, a date is required for re-issues, and optional for new specifications. Note that re-issues are situations where the version is not changed. A URI without a date, refers to the latest re-issue. In general, you should use the form of the URI without a date when referring to a document. For example, URIs embedded in FpML-compliant documents should not include the re-issue date. The form with a date exists to allow release notes and revision histories to refer to specific re-issues.

Note that URIs do not necessarily resolve to physical resources. However, FpML should provide files that correspond to the www.fpml.org/spec URIs.

6.7   URI Examples

As examples:

- http://www.fpml.org/spec/fpml-arch-1-0
  represents the FpML 1.0 architectural standard.

- http://www.fpml.org/spec/2000/fpml-arch-1-0-2000-04-20
  represents a specific re-issue of the standard.

- http://www.fpml.org/ext/iso4217
  is the FpML assigned URI for the external coding Scheme defined by the ISO 4217 specification.

- http://www.fpml.org/spec/2000/business-day-convention-1-0
  is the URI for an FpML defined coding Scheme.

- http://www.fpml.org/spec/fpml-dtd-1-0
  represents the latest release of the FpML 1.0 DTD.

7   Versioning

FpML is a living standard, that will evolve due to a long list of reasons, including:

- support for financial products.

- new roles for using the FpML standard.

- advances in XML, in particular Schemas.

Version identification is part of the overall strategy for evolving FpML and potentially supporting interoperability between versions.

Future additions to the FpML architecture will address the standards related to evolving FpML and interoperability between versions. This section defines the FpML standard for identification of the version of FpML used by an FpML message.

This versioning recommendation is intended to be applicable to the current DTD based FpML, and for Schema based versions of FpML in the future.

7.1   Version Identification

An FpML document must explicitly indicate which version of the FpML standard it adheres to. The DOCTYPE statement in an FpML document must reference the DTD (external subset) using the following XML Public Identifier.

```
<!DOCTYPE FpML PUBLIC "-//FpML//DTD Financial product Markup Language
1-0//EN" "fpml-1-0.dtd">[1]
```

This public identifier contains the FpML version. The above DOCTYPE includes an optional second literal that identifies a physical file location for the DTD. Normally XML processing tools map Public Identifiers to actual DTDs (files or otherwise) using an Oasis[2] catalogue (http://www.oasis-open.org/html/a401.htm). See Appendix A for a description of the catalog file format.

In addition to the DOCTYPE statement, the FpML element should also have a version attribute of the form:

```
<FpML version="1-0">[3]
```

---

[1]The format of the version string "1-0" was chosen to be suitable for both Public Identifiers and URLs.
[2]Formerly SGML Open
[3]<!ATTLIST FpML version (1-0) #REQUIRED >

7.2   Public Identifier Format

The XML standard only loosely defines the standards for public identifiers. That is:

- a public identifier may consist of the following characters:
  space | carriage_return | newline | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%][4]

- when matching public identifiers, leading and trailing 'white space'[5] is stripped and any remaining sequences of 'white spaces' are collapsed into a single space character.

The format of the public identifier recommended for the FpML DTD is based on the SGML standard (ISO/IEC 8879-1986). The two main parts are the Owner Identifier (in this case '-//FpML//') and the remainder of the Public Identifier ('DTD Financial product Markup Language 1-0//EN').

The idea is that the FpML organization owns all public identifiers that begin with "-//FpML//". This is actually an *unregistered owner identifier*. In contrast, if the FpML organization registered the FpML owner identifier with the registration authority, FpML could use the *registered owner identifier* form ("+//FpML//").

The first part of the remainder of the public identifier ('DTD') is meant to be the type of the external entity. In this case, the DTD external subset. The last part of the public identifier ('//EN') is the *native language* of the entity, expressed as an ISO 2 letter language code. In this case, English.

The portion of the identifier between the 'DTD' and the '//EN' is free form and is specified by FpML 1.0.

7.3   Public Identifiers and Catalogues

Public Identifiers provide an abstract way of referencing external entities including DTDs and DTD components. When using shrink-wrapped software products, such as XML editors or XML transformation engines, public identifiers are mapped to physical file locations using a catalog file. The catalog file can typically be specified via environment variable, configuration file, or command line argument.

This catalog file is a local configuration resource that is updated to include the DTDs that a software system is expected to support. A single catalog file can cater for all components and versions of such DTDs assuming these have distinct Public Identifiers.

7.4   Mapping Public Identifiers in Software

Many popular XML parsers already implement Oasis Catalog functionality. If not, these parsers provide mechanisms for intercepting entity references, so that these references can be mapped to locally cached copies of external entities such as DTDs and DTD components.

---

[4]TABs are not allowed in public identifiers.
[5]'white space' consists of 'space', 'carriage return' and 'newline' in the case of public identifiers.

Different XML parsers provided subtly different mechanisms for doing this. Appendix C provides some sample Java code that works with the Apache Xerces DOM parser.

7.5   Modular DTD Structure

The following standards apply to the FpML 1.0 DTD:

- The FpML DTD should be broken down into a small number of modular component DTDs.

- The FpML DTD should directly include all the other DTDs. Unlike component element sets, the FpML DTD is the only FpML DTD that should be directly referenced by an XML document.

- The division into modules may be influenced by versioning considerations. Each DTD has its own version number. In any one release of FpML, the main DTD must change. Some of the component DTDs may also change, and therefore their version numbers. Where a component DTD does not change, its version number may be left unchanged.

- Separate DTD components should be identified using an XML Public Identifier[6] that contains the version of the component that it identifies. The FpML DTD should reference the DTD components using these Public Identifiers.

- Only the top level FpML element has an associated version number. Sub-components do not have a version attribute.

7.6   DTD Versioning

The overall FpML standard follows a single release cycle specified by a single version string. Individual parts of the FpML DTD cannot change between releases of FpML as a whole.

The granularity of FpML versioning is based on the modularity of the FpML DTD. Every release of the FpML standard will involve assigning a new version to the FpML DTD and any DTD components that have changed.

This requires that FpML documents use a new DOCTYPE Public Identifier to reference the FpML DTD and that the FpML DTD use new DTD component Public Identifiers to reference changed DTD components.[7]

Logically the whole FpML DTD, and thus the whole FpML document that references it, has a single FpML version even if some DTD components may not have changed from an earlier version.

---

[6]These Public Indentifiers should be variants of the one used to identify the FpML DTD itself.
[7]As an example, assume that Forward Rate Agreements are in a separate DTD, that is included into the top most FpML DTD. The Forward Rate Agreement DTD could not move from version 1.3 to 1.4, without also incrementing the version of the top most FpML DTD. In contrast, the top most FpML DTD can move from version 1.3 to 1.4 without incrementing the DTD associated with Forward Rate Agreements.

7.7 Future XML Schema Versioning

In the future, when FpML moves to using XML Schema[8] the modular structuring of the DTD will be reflected in the XML Schema. Even if XML Public Identifiers cannot be used to reference XML Schema, or Schema components, the equivalent versioning Scheme can be applied to their Universal Resource Identifiers.

7.8 Versioning of External Entities

External standards have their own versioning methodology. For example, FpML uses the ISO currency codes. In this standard, currency codes may be added and removed from day to day[9], without any change in version number. FpML does not lock the standard. In turn, new ISO currency codes can be used without requiring a new release of FpML.

The References section of this document addresses external standards in greater detail, and defines the standards for identification of standards through the use of Schemes.

7.9 Upwards Compatibility

FpML 1.0 does not fully specify the standards regarding the evolution of FpML. Such standards will be released in the future. However, the remaining information in this section provides a foundation for such standards.

Where possible, every attempt should be made to keep changes to the FpML DTD, or Schema, as upwardly compatible as possible. In general, the loosening of restriction, or the adding of new optional elements is upwardly compatible.

For example, adding a completely new financial product in the form of a new element that is added to an *exclusive or* choice of elements in its parent's element model is upwardly compatible.

```
<!ENTITY % Product "(swap | fra | newProduct)" >
```

Making an optional element required, or completely changing the nature of an element or removing it altogether would not be upwardly compatible.

7.10 Major and Minor Versions

A reader of XML that is designed to interpret an FpML document of one major version number is not in general able to correctly interpret FpML of any other major version number. A transformation or redesigned reader would be required.

---

[8]The adoption of XML Schema will also require a new version of FpML (probably a major version).
[9]Occasionally, a three-letter code is reused for a different currency (albeit after a long period of non use).

Minor version numbers, possibly using multiple levels, should be used where it is appropriate to relax this requirement. As an illustration (not a recommendation), a version 2-5 reader might, with no extra code, accept versions 2-0 through 2-4.

Users of FpML should be aware that the numbers of levels of versioning might change from version to version, so the version after "2-1" may be "2-2-0".

In practice, we expect version numbers to have only a major version and a single level of minor version, for example 1-0 or 2-35. If FpML is versioned by a more complex Scheme than this, the standard should include a description of the rationale for each of the minor levels.

7.11 Interoperability Between Versions

A forthcoming Recommendation from the Architecture Working Group will address issues concerning:

● Transformation between different versions.

● Rules restricting changes between versions, including difference between major and minor.

● Extensibility within versions.

● Release documentation, describing changes from previous versions.

These outstanding issues do not prevent a first version of FpML from being delivered, but must be considered before a second version is delivered.

Appendices

Appendix A: Oasis Catalog File Format

 (© James Clark; this is available at: http://www.jclark.com/sp/catalog.htm)

The entity manager[10] generates a system identifier for every external entity using catalog entry files in the format defined by SGML Open Technical Resolution TR9401:1997. The entity manager will give an error if it is unable to generate a system identifier for an external entity. Normally if the external identifier for an entity includes a system identifier then the entity manager will use that as the effective system identifier for the entity; this behaviour can be changed using OVERRIDE or SYSTEM entries in a catalog entry file.

A catalog entry file contains a sequence of entries in one of the following forms:

PUBLIC pubid sysid

> This specifies that sysid should be used as the effective system identifier if the public identifier is pubid. Sysid is a system identifier as defined in ISO 8879 and pubid is a public identifier as defined in ISO 8879.

ENTITY name sysid

> This specifies that sysid should be used as the effective system identifier if the entity is a general entity whose name is name.

ENTITY %name sysid

> This specifies that sysid should be used as the effective system identifier if the entity is a parameter entity whose name is name. Note that there is no space between the % and the name.

DOCTYPE name sysid

> This specifies that sysid should be used as the effective system identifier if the entity is an entity declared in a document type declaration whose document type name is name.

LINKTYPE name sysid

> This specifies that sysid should be used as the effective system identifier if the entity is an entity declared in a link type declaration whose link type name is name.

NOTATION name sysid

---

[10]An entity manager is a notional component of an XML or SGML parser that is responsible for mapping an entity external identifier (public or system) to a physical file or datastream. The code in Appendix C would be considered an entity manager.

This specifies that sysid should be used as the effective system identifier for a notation whose name is name. This is relevant only with the -n option.

OVERRIDE bool

bool may be YES or NO. This sets the overriding mode for entries up to the next occurrence of OVERRIDE or the end of the catalog entry file. At the beginning of a catalog entry file the overriding mode will be NO. A PUBLIC, ENTITY, DOCTYPE, LINKTYPE or NOTATION entry with an overriding mode of YES will be used whether or not the external identifier has an explicit system identifier; those with an overriding mode of NO will be ignored if external identifier has an explicit system identifier.

SYSTEM sysid1 sysid2

This specifies that sysid2 should be used as the effective system identifier if the system identifier specified in the external identifier was sysid1. sysid2 should always be quoted to ensure that it is not misinterpreted when parsed by a system that does not support this extension.

SGMLDECL sysid

This specifies that if the document does not contain an SGML declaration, the SGML declaration in sysid should be implied.

DOCUMENT sysid

This specifies that the document entity is sysid. This entry is used only with the -C option.

CATALOG sysid

This specifies that sysid is the system identifier of an additional catalog entry file to be read after this one. Multiple CATALOG entries are allowed and will be read in order.

BASE sysid

This specifies that relative storage object identifiers in system identifiers in the catalog entry file following this entry should be resolved using first storage object identifier in sysid as the base, instead of the storage object identifiers of the storage objects comprising the catalog entry file. Note that the sysid must exist.

DELEGATE pubid-prefix sysid

This specifies that entities with a public identifier that has pubid-prefix as a prefix should be resolved using a catalog whose system identifier is sysid. For more details, see A Proposal for Delegating SGML Open Catalogs.

The delimiters can be omitted from the sysid provided it does not contain any white space. Comments are allowed between parameters delimited by -- as in SGML.

The entity manager will look for catalog entry files as follows:

1. a file called catalog in the same directory as the document entity, unless the environment variable SP_USE_DOCUMENT_CATALOG has the value NO or 0;

2. any catalog entry files specified using the -c option;

3. a list of files specified by the environment variable SGML_CATALOG_FILES; the list is separated by colons under Unix and by semi-colons under MS-DOS and Windows; if this environment variable is not set, then a system dependent list of catalog entry files will be used.

In fact catalog entry files are not restricted to being files: the name of a catalog entry file is interpreted as a system identifier.

A match in one catalog entry file will take precedence over any match in a later catalog entry file. A more specific matching entry in one catalog entry file will take priority over a less specific matching entry in the same catalog entry file. For this purpose, the order of specificity is (most specific first):

> SYSTEM entries;

> PUBLIC entries;

> DELEGATE entries ordered by the length of the prefix, longest first;

> ENTITY, DOCTYPE, LINKTYPE and NOTATION entries.

Appendix B: Hypothetical Oasis Catalog for FpML

```
PUBLIC "-//FpML//DTD Financial product Markup Language 1-0//EN" "fpml-1-0.dtd"

PUBLIC "-//FpML//DTD FpML Common Elements 1-0//EN" "D:/fpml/dtd_v1_0/common.dtd"
PUBLIC "-//FpML//DTD FpML Swap Products 1-0//EN" "D:/fpml/dtd_v1_0/swaps.dtd"
PUBLIC "-//FpML//DTD FpML Barrier Products 1-0//EN" "D:/fpml/dtd_v1_0/barriers.dtd"
PUBLIC "-//FpML//DTD FpML Exotic Products 1-0//EN" "D:/fpml/dtd_v1_0/exotics.dtd"

PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN" "D:/entities/iso-lat1.gml"
```

Appendix C: Sample Apache Xerces Code

Parser.java (partial):

```
    // create a DOM parser that will be used to parse the FpML document.
    //
    org.apache.xerces.parsers.DOMParser parser =
        new org.apache.xerces.parsers.DOMParser();

    // associate an entity resolver that will redirect DTD references to
    // the correct DTD files (see Catalog.java)
    //
     parser.setEntityResolver(new Catalog());

    // use the DOM parser to parse the FpML document
    //
     parser.parse(argv[0]);

    // get a pointer to the FpML document, then the root element
    // and then get the version string
    //
    org.w3c.dom.Document doc = parser.getDocument();
    org.w3c.dom.Element root = doc.getDocumentElement();
    String version = root.getAttribute("version");
```

Catalog.java (exception handling omitted):

```java
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;
import java.io.*;
import java.util.*;

public class Catalog implements EntityResolver
{
    // use the Java default constructor

    // resolveEntity is called every time an external entity is encountered,
    // directly or indirectly, in an XML document.  This includes the reference
    // to a DTD in the DOCTYPE statement and parameter entity references to
    // components parts of the DTD in the DOCTYPE external subset.
    //
    // This simple demonstration redirects only the recommended FpML public
    // identifier to a local file where a copy of the DTD resides.  This
    // corresponds with the first line of the Catalog file in Appendix B.
    //
    // A proper implementation should cater for all the Public Identifiers used
    // in the FpML standard.  The best way to do this would read the mappings from
    // a Catalog file such as the one in Appendix B.
    //
    public InputSource resolveEntity (String publicId, String systemId)
    {
        if (publicId.equals("-//FpML//DTD Financial product Markup Language 1-0//EN"))
        {
            // we are providing Xerces with an open file from which to read the DTD
            //
            return new InputSource(new FileReader("D:/fpml/dtd_v1_0/fpml.dtd"));
        }
        else
        {
            // returning null indicates that Xerces should attempt to resolve
            // the entity using its own devices
            //

            return null;
        }
    }
}
```

Appendix D: A Hypothetical FpML DTD

```
<!--This DTD is normally referenced using the following DOCTYPE statement:
<!DOCTYPE FpML PUBLIC "-//FpML//DTD Financial product Markup Language 1-0//EN">
-->

<!ENTITY % common PUBLIC "-//FpML//DTD FpML Common Elements 1-0//EN" >
%common;
<!ENTITY % swaps PUBLIC "-//FpML//DTD FpML Swap Products 1-0//EN" >
%swaps;
<!ENTITY % barriers PUBLIC "-//FpML//DTD FpML Barrier Products 1-0//EN" >
%barriers;
<!ENTITY % exotics PUBLIC "-//FpML//DTD FpML Exotic Products 1-0//EN" >
%exotics;

<!ELEMENT FpML (...) >
<!ATTLIST FpML version (1-0) #REQUIRED >

...
```