Financial products Markup Language

FpML® 4 User Guide

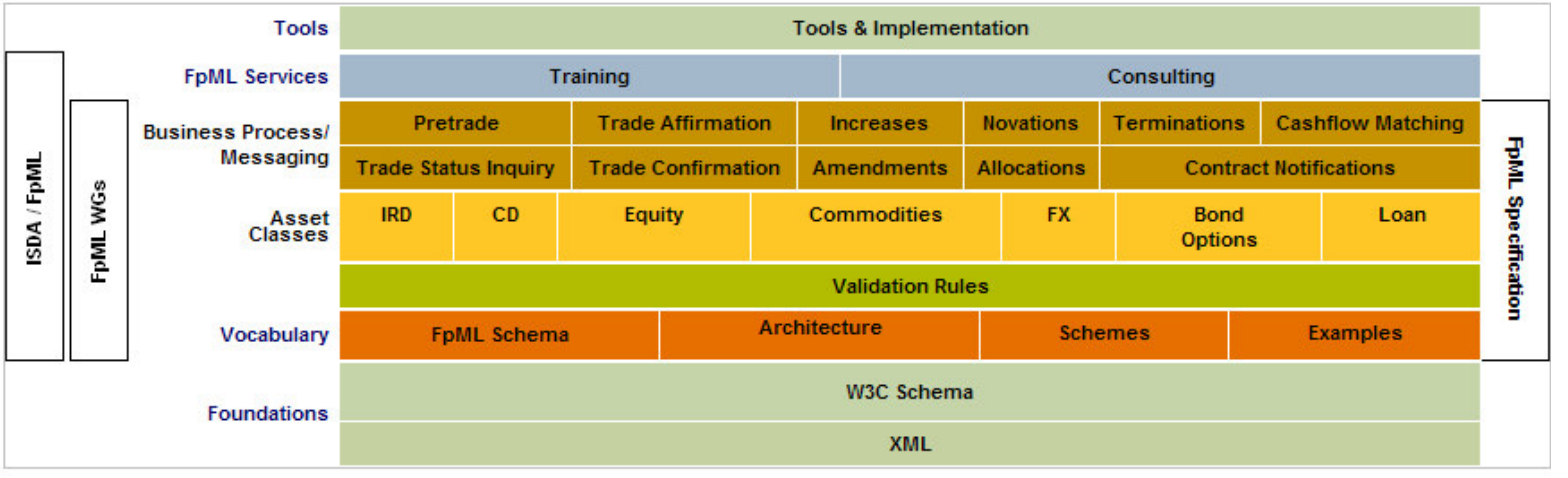
2010 Edition

ISDA®

INTERNATIONAL SWAPS AND DERIVATIVES ASSOCIATION, INC.

FpML at a glance

The following diagram illustrates the scope of the FpML 4 specification in relation to various dimensions such as technology, industry coverage, governing bodies, and implementation activities.



FpML® 4 User Guide

2010 Edition

ISDA®

INTERNATIONAL SWAPS AND DERIVATIVES ASSOCIATION, INC.

Published June 2010

Contents

Copyright Notice

1. Introduction

1.1. History of FpML

From the early 1980s when the interest rate swaps market began developing, the privately negotiated derivatives have grown tremendously in volume. According to the report on Derivatives Market Activity from the Bank for International Settlements, the notional principal outstanding of swaps and other over-the-counter (OTC) derivatives, stood at $95 trillion in 2000, doubled by the end of 2003 ($197 trillion), and reached $614 trillion in December 2009. This corresponds to a 23% annualized growth rate over this nine-year period.

To lower the cost of processing derivatives and thereby increase the profitability of the business, JP Morgan (now JPMorganChase), in 1997, established a research project to develop the methodology by which these instruments can be traded using e-commerce technologies. PricewaterhouseCoopers was brought on board as a resource, and in 1999 the organizations announced a draft standard for interest rate swaps. At that time, other industry firms were contacted and an independent organization – FpML.org – was formed to develop and promote the Financial products Markup Language (FpML) as an XML-based "lingua franca" for derivatives trading.

The FpML standard is freely licensed and is intended to automate the flow of information between derivatives participants, independent of the underlying software or hardware infrastructure, supporting activities related to these transactions.

On November 14, 2001 ISDA and FpML.org announced their intention to integrate the development process of the FpML standard into the ISDA organizational structure. This combined the organizational strengths of ISDA with FpML's technology base and allowed the FpML standard to be leveraged using the membership base and experience ISDA has built up since its formation in the mid 80s. The change is an indication of the increased importance of operations, automation and straight-through processing for the ISDA membership.

Expansion of the FpML standard to new products (e.g., correlation swaps, commodities) and business areas (e.g., regulatory reporting, collateral management, syndicated loan) and adoption by new users has continued under ISDA's sponsorship. Because of the rapid expansion of the standard and applications of the standard, and because of the increasing number of newcomers to FpML, there has been a growing need to provide introductory material to help users new to FpML understand how to use the standard. This user guide to FpML is intended to address this need.

1.2. About this User Guide

1.2.1. Purpose

The User Guide to FpML provides guidance to implementers about how FpML may be used by derivative market participants. It suggests applications for FpML, describes at a high level how to write FpML, and provides guidance on related topics.

The User Guide is complementary to the FpML standard specification, and is not a replacement for that specification. It describes specific instances and examples of using FpML, where the FpML specification provides a comprehensive reference and full schema description.

In case of any discrepancy between the documents, the FpML standard specification shall be regarded as correct.

1.2.2. Audience

This user guide is intended to be used primarily by technologists with some exposure to XML and some familiarity with OTC derivatives. Following are some suggestions on sections of particular interest to people in different roles:

- Technology managers: Section 2, Application Examples would be particularly relevant. Section 7, Customizing FpML may also be relevant.
- Business analysts/developers: Section 3, Writing FpML should be useful as an introduction to how FpML looks.
- Technical architects/standards developers: All sections, with a particular emphasis on sections 4 to 7.

1.2.3. Content Overview

The User Guide is organized as follows:

- **Section 2** contains examples of different applications that might use FpML, with a discussion about which features of FpML could be used. For each potential application, there are references to particularly relevant parts of the user guide and the FpML Specification, to help users quickly find related information.
- **Section 3** provides some examples of how to write simple FpML instance documents, to illustrate in basic terms how FpML is constructed.
- **Section 4** highlights some of FpML's key architectural underpinnings, and describes some cross-product components in FpML.
- **Section 5** briefly describes how FpML instruments are specified, and describes the instrument extension mechanism.
- **Section 6** covers support for business processes in FpML 4.7.
- **Section 7** introduces how to customize FpML for business-specific requirements.

1.2.4. Applicability

The FpML User Guide has been developed for FpML 4. In most cases, concepts, applications, and examples described in the user guide could be supported by previous versions of FpML. The examples were developed based on version 4.7; they may work as is with earlier versions or require small modifications.

 FpML 5	A separate user guide (FpML 5 User Guide – 2010 Edition) covers the latest changes introduced in version 5. The publication is also available from the ISDA Bookstore (www.isda.org).

1.2.1. Materials for Download

	The examples developed in this user guide are available for download at: http://www.fpml.org/userguide/fpml4ug2010a7

1.3. FpML Documentation

All published FpML documents can be found on the FpML website (http://www.fpml.org).

FpML Specifications (http://www.fpml.org/spec)

- The FpML Specifications section contains all the versions of the FpML Specification, including Recommendations and Working Drafts.
- Schemas and examples are available for download.
- As of this writing, the most recent version is FpML 4.7 Recommendation. The examples in this user guide were developed using FpML 4.7.
- The FpML Specifications section requires a simple, one-time registration that grants full access to all published versions of FpML.
- The section also contains the FpML Architecture Specification, the normative reference for the architectural rules under which FpML 4.x is developed. The latest version of the architecture is 2.2.

FpML Documents (http://www.fpml.org/documents)

- The FpML Documents section contains technical papers and proposals (e.g., validation, versioning, messaging, extensions)

Other information that can be found on the FpML website includes general background information on FpML schema, events, tools, consulting services. The FpML website is also where to sign up for FpML announcements, find further information about the FpML Working Groups (http://www.fpml.org/wg), or access the online FpML Subversion Source libraries.

1.4. Related Resources

In addition to the ISDA documentation, there are a variety of other resources that implementers can use to learn about FpML and related technologies. Some of these are described below.

1.4.1. XML Documentation

FpML is based on XML, the Extensible Markup Language. The FpML Specification and to a lesser degree the user guide assume familiarity with XML and with XML schema. The following are references to learn more about XML and XML Schema:

- XML
  - The XML specification: http://w3.org/XML/Core/#Publications
  - XML tutorial: http://www.w3schools.com/xml/default.asp
  - The Guide to the XML Galaxy: http://www.zvon.org/comp/r/tut-XML.html#
  - An index of XML publications: http://www.oasis-open.org/cover/xml.html
  - Brief article by an XML expert on how XML works: http://www.xml.com/pub/a/w3j/s3.walsh.html
  - Two-page quick reference guide to XML (assumes some familiarity with XML): http://www.mulberrytech.com/quickref/XMLquickref.pdf
- XML Schema
  - The W3C XML Schema specification: http://w3.org/XML/Schema.
  - A tutorial: http://www.xml.com/pub/a/2000/11/29/schemas/part1.html
  - Links to XML Schema quick reference guides: http://www.xml.dvint.com/

1.4.2. XML Terminology and Tools

Following are some common terms and tools used when working with XML:

- **Parser**: A parser is a software tool used to break XML documents into pieces that are convenient to be processed by a computer program. Most XML parsers use one (or both) of the following types of interfaces:
  - **DOM (Document Object Model)**: DOM parsers convert the XML document into a tree representation that is convenient for computer programs to process.
  - **SAX (Simple API for XML)**: SAX parsers convert the XML document into a sequence of "events" representing the different components in the document. SAX parsers allow more efficient programs to be written, especially for processing large documents, but are less convenient than DOM parsers.

  Common XML parser implementations include Apache Xerces (http://www.apache.org) for Java, or C++, or Microsoft MSXML ActiveX component.
- **Well-formedness**: An XML document that follows all of the standard XML rules is described as "well-formed". An XML parser can parse a well-formed document. An XML parser will report a parsing error if the input document is not well-formed.
- **Validating Parser**: A parser that can validate an XML document against a schema or DTD. Most modern XML parsers can validate, although validation is not used for some performance-intensive applications.
- **XSLT (Extensible Style Language - Transformations)**: A template-driven (pattern matching) language for processing XML. XSLT is particularly strong at reformatting XML, for example into HTML. See http://www.w3.org/Style/XSL/ for more on the XSLT specification. Common implementations of XSLT include MSXSL from Microsoft, Apache Xalan (http://xml.apache.org/xalan-j/), and Saxon (http://saxon.sourceforge.net/).
- **XML Schema**: An expressive, XML-based definition of an XML document's vocabulary and syntax. FpML 4 uses W3C XML schema to define its structure.
- **DTD (Document Type Definition)**: An older, more compact but less expressive (compared to XML Schema) way of defining an XML document's vocabulary and syntax. Versions of FpML prior to version 4 used DTDs.
- **Instance Document**: An XML document that contains business data expressed using the syntax rules defined by either a DTD or XML Schema.
- **Namespace**: An identified collection of XML component names. A namespace is typically defined by a schema. Namespaces are used to allow an instance document to specify which vocabulary is meant when a particular name is used.

1.4.3. Web-based Resources

There are a variety of resources on the web for implementers interested in learning more about FpML and FpML applications. By using a web search tool to look for FpML and related terms, you can find up to date information about solutions related to your application needs. In particular, for most of the applications described in section 2 there are web-based resources. Some of the keywords that can be searched, in addition to "FpML", include "confirmation service," "matching messages" and "reconciliation".

Currently the FpML website (http://www.fpml.org) serves as the main source for FpML related information and applications. Along with all published documentation and the specification itself, comprehensive listings of participating organizations and vendors supporting FpML products are available.

2. How to Use FpML – Sample Applications

2.1. Introduction

2.1.1. Purpose

This section provides examples of generic types of applications that users may wish to implement using FpML. All of these applications are known to have been implemented in some form, and many have been implemented by a number of firms. For each application, there are some suggestions about which FpML messages or structures to use, as well as a brief discussion of issues the implementer may face and resources for addressing them.

2.1.2. Overview

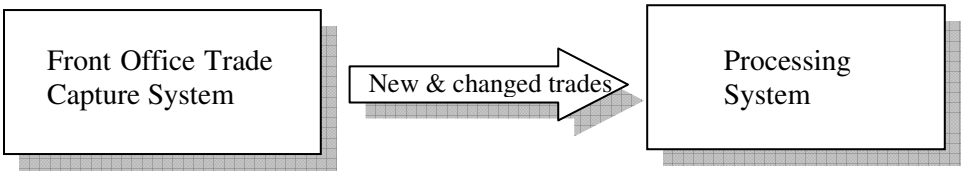
The example applications covered in this section include:

- A straight-through-processing (STP) data feed from a front office system to a processing system.
- A multilateral confirmation service.
- A trade repository/archive.
- Activity reconciliation between related systems/parties.
- Position reconciliation between related systems/parties.
- Reporting applications.

2.2. Straight-Through Processing (STP) Data Transfer

2.2.1. Objective

A common application for FpML is to send trade information automatically from front office systems to downstream processing systems, to reduce re-keying efforts, error rates, and operational risk.



2.2.2. Requirements

To support the application, the following is required:

- The ability to send new, modified, and cancelled trades from the front office to the processing system.
- The ability to record trade details for a wide variety of products.
- The ability to support internal data, such as revenue allocation information.

2.2.3. Messages and Structures Used

To support the workflow requirements, the FpML Execution messages can be used, i.e.:

- TradeExecution (formerly TradeCreated in FpML 4.3)
- TradeExecutionModified (formerly TradeAmended)
- TradeExecutionCancelled (formerly TradeCancelled)

These messages are described in Section 6.

The product coverage is addressed by the FpML product model, outlined in Section 5. If additional products or product characteristics are required, these can be customized as described in Sections 5 and 7.

Additional data items not covered by FpML can be covered through extensions as described in Section 7.

2.2.4. Tools and Technology

To implement this application, the following is needed:

- The ability to produce FpML in the upstream system. This can be done in a variety of ways and in different technical environments. Some techniques suited for this include:
  - "print" statements. Since XML is just text, it can be produced by most programming languages.
  - Constructing XML via a DOM (Document Object Model) tree. This allows the XML to be constructed in memory and then produced as one operation.
  - Using various FpML API frameworks, for example code generated from the schema.
- The ability to transmit FpML from the upstream system to the downstream system. Different ways of doing this include:

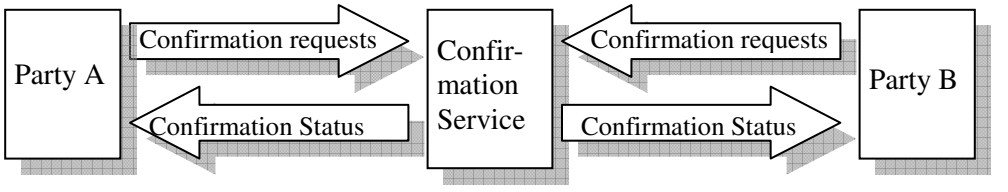
  - File transfer, e.g., using shared network drives or file transfer software such as FTP.
  - Message oriented middleware, such as IBM Websphere MQ.
  - Direct socket connections.
  - Web services, such as SOAP over HTTP.
- The ability to extract the business information from the FpML instance document in the downstream system. This is typically done in one of the following ways:
  - by "parsing" the FpML using an off the shelf XML parser, and writing custom code to process the data.
  - By using a data extraction or conversion program to convert the data into a format recognizable by the downstream system, such as a delimited file format. The extractors or converters can be written using technologies such as XSLT.

2.3. Automatic Confirmation

2.3.1. Objective

To provide automatic confirmation of a trade, there are two models in general use in the industry: matching and affirmation. The model described here and diagrammed below is a confirmation matching model; some services may implement an affirmation model, in which one party affirms (agrees to) the other party's trade. In the affirmation model there is only one representation of the trade. The FpML specifications detail messages and business process descriptions for both models.

In this application, each party sends trade confirmation requests to the confirmation service; the service matches the requests and sends resulting status updates to the parties, indicating whether the trades have been confirmed.



2.3.2. Requirements

Following are the high level requirements for the application:

- The ability for parties to request confirmations, modify the information and resubmit when there is a problem; and cancel confirmation requests in case of error.
- The ability to tightly control the list of valid trades/products to reduce the probability of error or unintended mismatch.
- The ability for the confirmation service to communicate the status of the confirmations (e.g., whether the trades match, have differences (mismatch), or where no matching trade is found).

2.3.3. Messages used

To make requests to the confirmation service, the following FpML messages can be used:

- **Request Trade Confirmation**: Initiate a trade confirmation process
- **Modify Trade Confirmation**: Change trade details for a trade currently being confirmed
- **Cancel Trade Confirmation**: Stop a trade confirmation process
- **Confirm Trade**: Indicate acceptance of a confirmation
- **Request Trade Status**: Determine the current status of a trade or of a number of trades

For confirmation services to return trade statuses, the following messages can be used:

- **Trade Matched**: Trades from two different sources are matched
- **Trade Mismatched**: Trades from two different sources have some differences
- **Trade Unmatched**: No matching trade was found
- **Trade Alleged**: A trade was submitted by a counterparty, but no matching trade can be found from your side

- **Trade Confirmed**: A confirmation acceptance has been received from both sides.
- **Trade Status Message**: A generic message that can be used to indicate the status of a trade.

These messages are described in Section 6.

2.3.4. Customizations / Restrictions

To restrict the range of acceptable products, and to add service-specific details, the schema can be customized to add fields and/or eliminate options. See Section 7 for more information on how to customize FpML.

2.3.5. Validation

To validate that the confirmation message meets basic FpML business rules and possibly platform-specific business rules, the FpML validation framework can be used. This allows specific business constraints to be enforced for all input messages. See Section 4 for more information.

2.3.6. Tools and Technologies

To implement this application, parties will need the following:

- The ability to create the FpML, as described above in section 2.2.4. Parties need to ensure that the generated FpML meets the confirmation service's specific requirements and constraints.
- The ability to transmit the FpML to the confirmation service. The mechanism to do this will typically be defined by the service, and may include technologies such as:
    - Custom Application Programming Interfaces (APIs), typically based on sockets over dedicated networks
    - Message-oriented middleware on dedicated networks
    - Web services

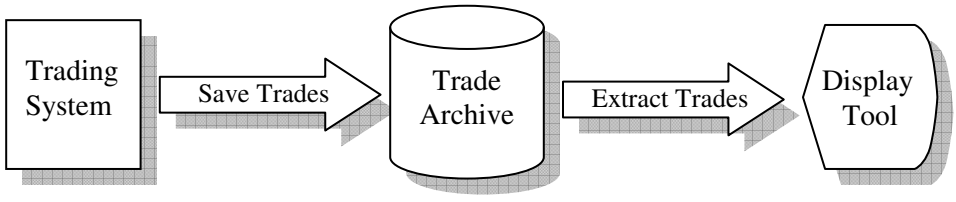
Services will need to create the following, among other items:

- A message delivery infrastructure
- A trade validation mechanism
- A trade matching and persistence mechanism

2.4. Trade Archiving

2.4.1. Objective

Another possible application is the use of FpML to store trades, e.g., to a file system or database. The flexibility of FpML allows a variety of trades with different details to be saved without ongoing database maintenance. Also, FpML is well-suited for saving multiple versions of trades; each trade version can be stored in a different file or database record. A simple application for this type of trade archive allows trades to be pulled from the repository and displayed.



2.4.2. Structures Used

In this application, although it is possible to represent the trades using messages, there is less benefit in doing so, because the application is not fundamentally a messaging application. If messages are desired, the A2A messages described in Section 2.2 can be used. Alternatively, the FpML "DataDocument" format, a non-message format, can be used. → See Section 4 for more details on DataDocument.

In addition, there may be some desire to organize or group trades into collections of related trades, e.g., based on underlyer, trading desk, counterparty. If so, the FpML "Portfolio" structure, described in Section 4, can be used to record portfolio contents.

2.4.3. Tools

To implement this type of application, the following is needed:

- The ability to generate FpML from the source system. This can be done in a variety of ways, as described above.
- The ability to store and organize the files. Different ways of doing this include:
  - Using a basic file system-based storage scheme
  - In a relational database, with one "blob" (or large text buffer) record per trade version stored in the database
  - Using an XML document management tool
- A retrieval mechanism. This is usually tied to the storage mechanism, but may in addition offer features such as:
  - The ability to query trades based on content
  - The ability to display multiple versions of trades, and possibly to compare them
  - Web-based navigation or query tools

- A mechanism for displaying the trades. This can be done in several ways, such as:
  - A web browser-based display tool such as the *FpML Editor/Viewer*, available from the FpML web site

- A Java application based display application
- A system for rendering the trades into a report format (e.g., into PDF files) as required

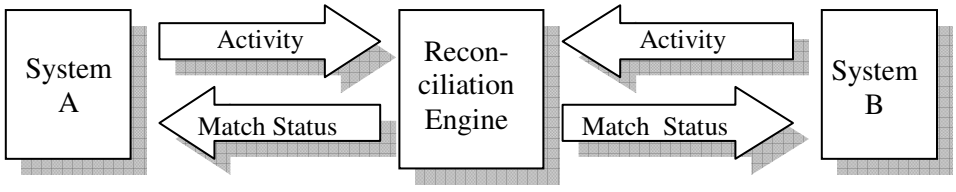
2.4.4. Issues/Notes

- One concern often cited by developers with the use of XML (and especially FpML) for this type of application is the large file size. However, this concern is usually not a serious limitation, for the following reasons:
  - FpML trade representations are typically not really that large relative to modern technology. A typical parametric view of a trade is about 10-20KB. 100,000 trades will require about 1-2 GB of storage. 100 versions each of 100,000 trades will require about 100 GB. This will fit on a single large disk.
  - If this is too much (for example, because additional information such as cash flows is being kept, or because storage is limited), FpML is highly compressible. A single trade is likely to compress by a factor of 4 to 10, and more if it is large.
  - A large collection of trades and trade versions is likely to compress by a factor of at least 10 and up to 100 fold or more when compressed as a single unit, depending on the commonality between trades. Also, one compressed large archive file typically uses computer file systems more efficiently than a large number of small files. This technique is useful for creating and distributing archival snapshots of large numbers of trades. For example, a single CD or even flash drive could hold a dealer's entire current derivatives inventory, possibly even with cash flows, when represented as a compressed archive.
- There are tools on the marketplace for comparing XML or FpML instance documents and reporting differences between them. For reporting differences between successive versions of FpML instance documents, a number of XML differencing tools could be used. For more information, search the web using key words "XML differencing" or "XML diff".

2.5. Intra-day Activity Reconciliation

2.5.1. Objective

FpML can be used to reconcile intra-day activity as recorded in one system with that in another. This type of check could be used to monitor manual dual-keying processes, as a check on automated STP processes, or as a check between separate entities. In general, the idea is to provide a real-time electronic comparison of state changes across systems.



2.5.2. Messages Used

Following are some of the messages that can be used to implement this workflow:

- From source systems to reconciliation engine:
  - Request Trade Match
  - Modify Trade Match
  - Cancel Trade Match
  - Request Trade Status
- From reconciliation engine to source systems:
  - Trade Matched
  - Trade Mismatched
  - Trade Unmatched
  - Trade Status

See Section 6 for more information about the matching messages.

2.5.3. Reconciliation Approaches

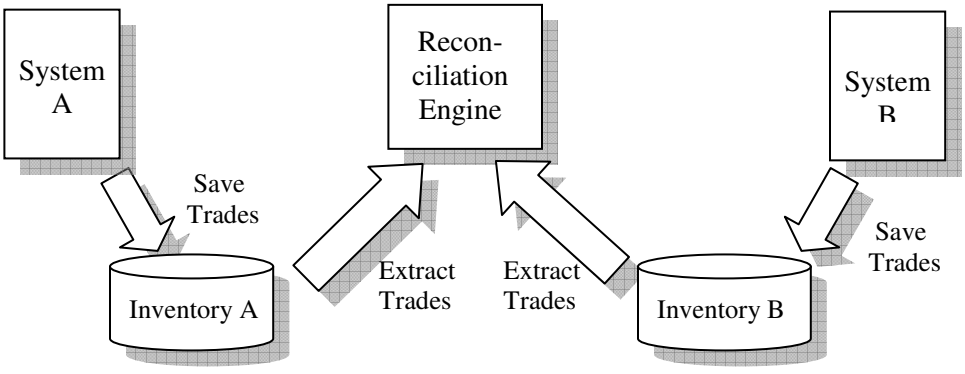
To implement the reconciliation engine, firms have employed different strategies, including:

- Internally developed systems: Some firms choose to develop custom FpML matching and reconciliation technology.
- Third party systems: Some firms choose to acquire and deploy third party matching/reconciliation technology, including
  - XML differencing: There are some generic XML tools for comparing documents that can be used.
  - FpML-specific tools: There are some FpML-specific or FpML-aware tools that can be used for matching/reconciling trades represented in FpML.

2.6. Inventory Reconciliation

2.6.1. Objective

Even when automated intraday reconciliation and confirmation processes exist, one might reconcile the current trade inventory as recorded in one system with that in another. This can be done on a periodic or incremental basis. The objective is to confirm that the trade population in the two places is within tolerable differences, if not identical.



2.6.2. Messages/Structures Used

In this application, the key is representing the trade economics; the workflow/activities are not important. For this reason, the preferred formats used to save the trades into FpML files are similar to those used in the Trade Archive application discussed above.

In addition, it is common to organize the trades in the inventories into partitions or portfolios. This allows the reconciliation to work on smaller subsets of the inventory at a time. This typically provides performance and reliability advantages.

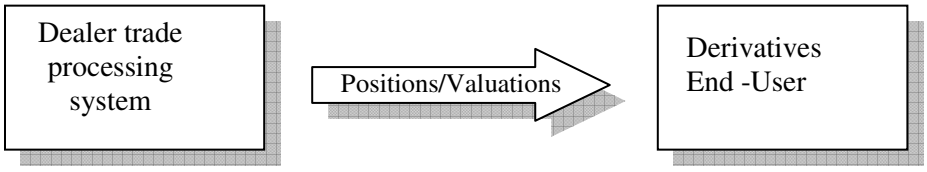
2.6.3. Matching/Reconciliation Approaches

Matching and reconciliation approaches are similar to those for intra-day reconciliation, but the message processing and management requirements are simpler. On the other hand, the data volume requirements are typically much higher.

2.7. Trade Reporting Applications

2.7.1. Objective

Yet another use of FpML might be to periodically (e.g., daily) generate reports of trade inventories and valuations for specific counterparties and send these to the counterparties to support processes such as collateral calls. Typically these reports will include multiple trades from multiple product types. Below is an example of this type of feed:



2.7.2. Messages/Structures Used

The **PositionReport** message fully supports this application by incorporating trade as well as valuation information. The trade data content is modeled reusing the existing FpML trade and product representations, discussed in Sections 4 and 5. The valuation information is modeled using the FpML Pricing and Risk framework, discussed in Section 6.6.

2.7.3. Tools and Technologies

To implement this application you will need technologies similar to those described above in Section 2.2. However, because this type of application is typically a periodic (e.g., daily) process, there are some differences in the types of suitable technologies. Following are some of the technologies that are likely to be required:

- The ability to generate FpML, as discussed in previous sections.
- The ability to transmit the FpML from dealers to clients. This could be done in a variety of ways, including:
  - File transfer
  - Download from a secure website
  - Some form of secure electronic mail, e.g., PGP / GPG
- The ability to validate the FpML. This could be done using technologies including:
  - Custom XSLT style sheets
  - Business rules written using the FpML validation framework and implemented on a commercially available validation processor
- The ability to read the incoming FpML. This could be done by converting the file to a format suitable for processing in existing systems, for example via:
  - XSLT-based style sheets
- An XPath-based data extractor written in Java or C++, for example.

3. How to Write FpML – Examples

3.1. Introduction

This section demonstrates some of the key FpML concepts by building two examples that demonstrate how and why FpML is formed.

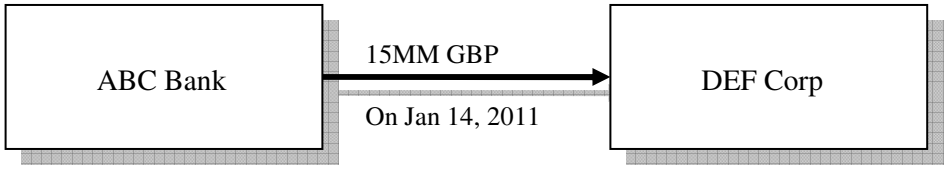
The first example (section 3.2 Forward Payment) demonstrates some basic aspects of FpML by representing a trade with very few economic details.

The second example (section 3.3 IR Swap) concerns a vanilla interest rate swap confirmation message, and demonstrates how a somewhat more complex FpML instance document is formed.

3.2. Basic Example: Forward Payment

3.2.1. Introduction

On Jan. 14, 2010, ABC Bank agrees to make a payment to DEF Corporation in one year's time, on Jan. 14, 2011. This date will be adjusted if it falls on a holiday. The payment will be 15,000,000 GBP.



In this example we develop an FpML 4.7 document that represents this transaction.

3.2.2. Required Data Attributes

The following attributes are required to represent this transaction:

- The parties to the trade: represented by SWIFT BIC codes ABCDUS33 and DEFGGB2L.
- The trade date: 2010-01-14
- ABC's reference number: A001.
- The type of trade: "bullet payment"
- Who's paying: ABC
- The payment amount: 15,000,000
- The payment currency: GBP
- The payment date: 2011-01-14
- Date adjustment rules: "Following" convention, NY and London banking days

3.2.3. Developing the FpML

3.2.3.1. Root Element

First, specify (in XML) that FpML 4.7 is being used to create the FpML instance document.

All XML documents have what is called a "root" element or "document" element.

In FpML, the root element is called, naturally, "FpML", and describes the FpML version, schema, and some other information.

Tags in XML are defined using angle brackets ("<" and ">" signs). There is an opening tag (<mytag>) followed by a matching closing tag (</mytag>).

The following is a simplified version of the starting and ending FpML tags:

```
<FpML version="4-7" xsi:type="DataDocument" . . . >

 <!-- party and trade details will go here -->

</FpML>
```

- The **FpML** keyword says that this is the top ("root") of the FpML instance document.
- The **version** attribute says that we are using FpML 4.7.
- The **xsi:type** attribute say that this holds FpML data, not an FpML message.
- Some additional attributes (not shown here) will specify exactly how the "FpML" namespace is defined. This will link to the FpML schema.
- The **<!--** and **-->** symbols surround an XML comment; the comment is ignored by the receiving program.
- The **</FpML>** tag ends the FpML.

 FpML 5	In FpML 5, the root element will change. Type substitution on the root element is no longer used to select the message type and its associated content model. In FpML 5 every message has been given its own global element which can appear as a document root element.

The FpML element, the xsi:type attribute, and the additional attributes on the FpML root element are discussed in more detail in Section 4.

3.2.3.2. Parties

Second, model the participants in the trade in FpML. In FpML, the participants are represented by the "party" element.

Following is an FpML representation of the two parties in this transaction:

```
<FpML version="4-7"…>
 <party id="abc">
   <partyId>ABCDUS33</partyId>
 </party>
 <party id="def">
   <partyId>DEFGGB2L</partyId>
   <partyName>DEF Corp</partyName> <!-- optional -->
 </party>
</FpML>
```

- The **party** keyword introduces a party definition
- The attribute, **id = "abc"** creates an identifier that can be used throughout the document for referring to this party
- The **partyId** keyword specifies the party's ID (default is a SWIFT BIC code, but other coding schemes can be used).
- The **partyName** keyword, can optionally be used to hold the legal name of the party.

The **party** element is also discussed in Section 4.

3.2.3.3. Trade

Next, represent the business transaction itself. In FpML, this is represented by a "trade" element with a "tradeHeader" that provides some reference information:

```
<FpML version="4-7"…>
 <trade>
   <tradeHeader>
     <partyTradeIdentifier>
       <partyReference href="abc"/>
       <tradeId>A001</tradeId>
     </partyTradeIdentifier>
     <tradeDate>2010-01-14</tradeDate>
   </tradeHeader>
   <!-- product info skipped -->
 </trade>
 <-- party information omitted -->
</FpML>
```

- The **trade** keyword introduces the trade.
- The **tradeHeader** keyword introduces identifying (non-economic) details.
- The **partyTradeIdentifier** block holds identifying information specific to party ABC.
- The **tradeId** element hold's party A's reference number for this trade.
- The **tradeDate** tag says when the trade was done.

The "trade" element is discussed in more detail in Section 4.

Please note that in FpML, the "party" elements typically go at the end of the FpML, after the trades. The locations of an *href* and its corresponding *id* in the instance document do not matter since XML references can work frontward or backward.

So far from the list of required data elements, we have represented the following:

- The parties:ABCDUS33 and DEFGGB2L, in the "party" elements,
- The trade date: 2010-01-14; and
- ABC's reference number: A001

The trade date and reference number appear in the "tradeHeader" element.

3.2.3.4. Product

Next, represent the actual trade economics; in FpML this is called the "product". The product for a single payment in FpML is called a "bullet payment" and is represented by the "bulletPayment" tag.

```
<FpML version="4-7" …>
 <trade>
   <!-- trade header skipped -->
   <bulletPayment>
     <payment>
       <!-- payment details go here -->
     </payment>
   </bulletPayment>
 </trade>
 <-- party information omitted -->
</FpML>
```

- **bulletPayment** is a kind of FpML "product"; this specifies what type of trade is being done.
- Different products (e.g., swap, fxSwap, equityOption) could go here. See Section 5 for more details on product substitution.
- **bulletPayment** is a product that consists only of a single "payment".
- Note that **payment** can be used in other places in FpML as well. For example, in option premium payments.

Products are covered in more detail in Section 5.

Next, specify the amount and direction of the payment. This is done by filling in part of the <payment> element, specifically the paying and receiving parties, and the amount of the payment:

```
<FpML version="4-7" …>
 <trade>
   <!-- trade header skipped -->
   <bulletPayment>
     <payment>
       <payerPartyReference href="abc"/>
       <receiverPartyReference href="def"/>
       <paymentAmount>
         <currency>GBP</currency>
         <amount>15000000.00</amount>
       </paymentAmount>
       <!-- payment date information skipped -->
     </payment>
   </bulletPayment>
 </trade>
 <-- party information omitted -->
</FpML>
```

- **payerPartyReference** and **receiverPartyReference** specify respectively which party pays and which party is on the receiving side. (ABC pays, DEF receives).
- They link to (reference) the party IDs (i.e., abc and def) created earlier.
- **paymentAmount** introduces the size of the payment.
- **currency** and **amount** provide details on the payment.

Finally, within the bullet payment the date of the payment and the rules for adjusting it are specified:

```
<!-- FpML and trade header information skipped -->
<payment>
<!-- party and amount info skipped -->
  <paymentDate>
      <unadjustedDate>2011-01-14</unadjustedDate>
      <dateAdjustments>
            <businessDayConvention>FOLLOWING</businessDayConvention>
         <businessCenters>
            <businessCenter>GBLO</businessCenter>
            <businessCenter>USNY</businessCenter>
         </businessCenters>
      </dateAdjustments>
   </paymentDate>
</payment>
<-- rest of trade and party information omitted -->
```

- **paymentDate** introduces the payment date and its adjustment rules
- **unadjustedDate** gives the date prior to any adjustments
- **dateAdjustments** gives the adjustment rules
- **businessDayConvention** specifies what adjustment type is used
- **businessCenters** specify what holiday centers to use, using a SWIFT coding scheme (these are London and New York)

The concept of date adjustment is discussed in more detail in Section 4. This type of "adjustable date" structure, and the "dateAdjustments" contained within it, is widely used throughout FpML.

3.2.3.5. Completed Example

Below is the completed FpML corresponding to the previous fragments:

```
<FpML version="4-7" xsi:type="DataDocument">
  <trade>
    <tradeHeader>
      <partyTradeIdentifier>
        <partyReference href="abc"/>
        <tradeId>A001</tradeId>
      </partyTradeIdentifier>
      <tradeDate>2010-01-14</tradeDate>
    </tradeHeader>
    <bulletPayment>
      <payment>
        <payerPartyReference href="abc"/>
        <receiverPartyReference href="def"/>
        <paymentAmount>
          <currency>GBP</currency>
          <amount>15000000.00</amount>
        </paymentAmount>
        <paymentDate>
          <unadjustedDate>2011-01-14</unadjustedDate>
          <dateAdjustments>
            <businessDayConvention>FOLLOWING</businessDayConvention>
            <businessCenters>
              <businessCenter>GBLO</businessCenter>
              <businessCenter>USNY</businessCenter>
            </businessCenters>
          </dateAdjustments>
        </paymentDate>
      </payment>
    </bulletPayment>
  </trade>
  <party id="abc">
    <partyId>ABCUS33</partyId>
  </party>
  <party id="def">
    <partyId>DEFGB2L</partyId>
  </party>
  </FpML>
```

Trade Identifying Information

Product Information

Party Information

3.2.3.6. Discussion

The above example illustrates some of the key concepts in writing FpML:

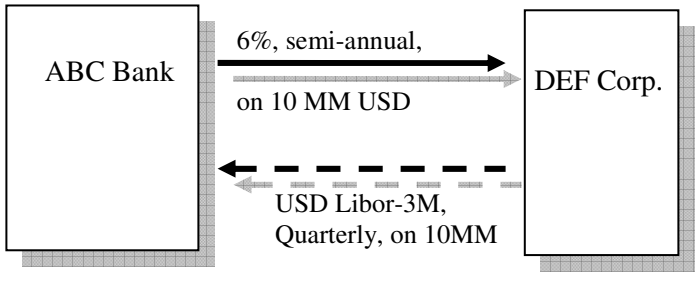
- FpML instance documents are wrapped in an FpML keyword (<FpML> root element) that specifies the FpML version, schema location, and other information. The example omits at least one important attribute, the XML namespace (xmlns) attribute, that specifies how to find the FpML schema. (See Section 4 for more information on linking schemas to FpML instance documents).
- Parties are represented using the <party> tag. See Section 4 for more information on parties.
- Transactions are described using the <trade> tag, and the <tradeHeader> tag is used to provide identifying information. See Section 4 for more information on trades.
- Products are represented by a variety of tags under the <trade> tag, following the <tradeHeader>. See Section 5 for more information about how products are specified.
- Adjustable dates are frequently used to represent date information. See Section 4 for more information on commonly used date building blocks.
- Payments are represented using a variety of reusable types. See Section 4 for more information on commonly used building blocks such as payment.

3.3. IR Swap Confirmation Message Example

3.3.1. Introduction

ABC Bank wishes to send a message to DEF Corp. confirming an interest rate swap. The swap is a 5-year, $10 million notional fixed/float swap; ABC pays 6% semi-annually, and DEF pays 3 month Libor quarterly.

The diagram below represents the cash flows of the deal:



3.3.2. Required Data Attributes

Following are some of the attributes that need to be captured:

- The sender of the message: ABC
- The recipient of the message: DEF
- The parties to the trade: represented by SWIFT BIC codes ABCDUS33 and DEFGGB2L
- The trade date: January 14, 2010
- The effective and termination dates: Jan. 16, 2010 to Jan. 16, 2015
- The date adjustment conventions used for the effective and termination dates and the calculation periods: Modified Following convention

- The business days applicable for the date adjustments: NY business days
- The roll date: calculation periods roll on the 16th of the month
- The calculation and payment frequency: semi-annual on the fixed side, quarterly on the floating side
- Rate set in advance or in arrears: rates set in advance (beginning of calculation period)
- Reset adjustments: modified following, using NY and London business days.
- Fixing lag: 2 days (fixing 2 days before rate reset)
- Payment in advance or in arrears: payment in arrears (end of calculation period).
- The notional: 10,000,000 USD
- The fixed rate: 6.00%
- The fixed rate day count fraction: 30/360
- Who is paying fixed (ABC) and float (DEF)
- The floating rate index (a.k.a floating rate option): USD Libor, sourced from Telerate, 3 month tenor
- The floating rate day count fraction: ACT/ACT.ISDA

3.3.3. Developing the FpML

3.3.3.1. Headers

FpML root element

The FpML root element is similar to that in the bullet payment example, except that in this case the FpML instance document type is a "Request Trade Confirmation" message. The xsi:type attribute notifies the recipient of the purpose of the message and enforces the content of the message.

```
<FpML version="4-7" xsi:type="RequestTradeConfirmation" . . . >
  <!-- FpML content will go here -->
</FpML>
```

Message header

Since this is an FpML message, an FpML message header must be supplied. Among other things, the message header will specify the sender and the recipient.

```
<FpML version="4-7" xsi:type="RequestTradeConfirmation" . . . >
 <header>
   <messageId messageIdScheme="http://abc.com/msg">A01</messageId>
   <sentBy>ABCDUS33</sentBy>
   <sendTo>DEFGGB2L</sendTo>
   <creationTimestamp>2010-01-14T15:38:00-00:00</creationTimestamp>
 </header>
</FpML>
```

- The **messageId** provides an identifier that can be used to refer to the message, for example in a reply
- The **sentBy** field says who sent the message
- The **sendTo** field says who the message was sent to, i.e., the recipient

- The **creationTimestamp** says when the message was created. It is expressed as a date, time, and time zone (offset from UTC)

The message header is discussed in more detail in Section 4.

Parties, Trade and Trade Header

The party, trade and trade header information is similar to that for a bullet payment, except for the values of the fields (e.g., the reference number), and therefore are not repeated here. Please see the bullet payment example in the previous section for more detail on how to fill these elements.

3.3.3.2. Swap Product

The interest rate swap itself is represented with the FpML "swap" keyword in the product slot of the trade. A swap can contain several swap streams in FpML. The typical number of swap streams is 2, as in this example. Each stream represents a payment stream from one party to the other.

```
FpML version="4-7" …>
 <trade>
   <!-- trade header skipped -->
   <swap>
     <swapStream>
       <!-- fixed stream details go here -->
     </swapStream>
     <swapStream>
       <!-- float stream details go here -->
     </swapStream>
   </swap>
 </trade>
 <-- party information omitted -->
</FpML>
```

In this example, the first swap stream will represent the fixed payments from ABC to DEF, while the second will represent the floating payments from DEF to ABC.

3.3.3.3. Fixed Stream

Each stream starts with elements indicating which party is paying the cash flows represented by the stream. In the case of the fixed stream, DEF is paying and ABC is receiving:

```
<swapStream>
  <payerPartyReference href="def" />
  <receiverPartyReference href="abc" />
  <!-- etc… -->
```

Key remaining components of fixed interest rate streams include:

- **Calculation period dates**, specifying when calculation periods occur;
- **Payment dates**, specifying when payments occur; and
- **Calculation period amounts**, specifying the amount of the payments.

Calculation Period Dates

The "calculationPeriodDates" component specifies when the calculation periods occur, including the effective and termination dates, the frequency, and the roll convention:

```
<calculationPeriodDates id="fixDates">
   <effectiveDate>
      <unadjustedDate>2010-01-16</unadjustedDate>
      <dateAdjustments>
         <businessDayConvention>NONE</businessDayConvention>
      </dateAdjustments>
   </effectiveDate>
    <terminationDate>
      <unadjustedDate>2015-01-16</unadjustedDate>
      <dateAdjustments>
           <businessDayConvention>MODFOLLOWING</businessDayConvention>
        <businessCenters id="busCtrs">
           <businessCenter>USNY</businessCenter>
        </businessCenters>
      </dateAdjustments>
   </terminationDate>
    <calculationPeriodDatesAdjustments>
      <businessDayConvention>MODFOLLOWING</businessDayConvention>
      <businessCentersReference href="busCtrs"/>
    </calculationPeriodDatesAdjustments>
    <calculationPeriodFrequency>
       <periodMultiplier>6</periodMultiplier>
       <period>M</period>
       <rollConvention>16</rollConvention>
    </calculationPeriodFrequency>
</calculationPeriodDates>
```

The above specifies:

- The **effective date**: 2010-01-16, with no adjustments.
- The **termination date**: 2015-01-16, adjusted using the modified following convention, using New York business days.
- The **adjustments** to be applied for each calculation period: modified following convention, and the same business centers as the termination date, i.e., New York business center.
- The **calculation frequency:** every 6 months, with a roll date on the 16th of the month.

Payment Dates

The payment dates structure specifies how frequently and when the stream cash flows are paid. In this case, the payments are semi-annually, at the end of each calculation period.

```
<paymentDates>
  <calculationPeriodDatesReference href="fixDates" />
  <paymentFrequency>
    <periodMultiplier>6</periodMultiplier>
    <period>M</period>
  </paymentFrequency>
  <payRelativeTo>CalculationPeriodEndDate</payRelativeTo>
  <paymentDatesAdjustments>
      <businessDayConvention>MODFOLLOWING</businessDayConvention>
    <businessCentersReference href="busCtrs"/>
  </paymentDatesAdjustments>
</paymentDates>
```

- The **calculationPeriodDatesReference** element explicitly links the payments to the fixed stream calculation periods.
- The **paymentFrequency** element defines that payments occur every 6 months.
- The **payRelativeTo** element states that payments are made at the end of the calculation period.
- The **paymentDatesAdjustments** indicates that payment dates are to be adjusted using the modified following convention, using the set of business centers defined above (i.e., New York business days).

Calculation Period Amounts

The calculation period amounts element specifies how much money is paid during each calculation period.

```
<calculationPeriodAmount>
  <calculation>
    <notionalSchedule>
      <notionalStepSchedule>
        <initialValue>10000000.00</initialValue>
        <currency>USD</currency>
      </notionalStepSchedule>
    </notionalSchedule>
    <fixedRateSchedule>
      <initialValue>0.06</initialValue>
    </fixedRateSchedule>
    <dayCountFraction>30/360</dayCountFraction>
  </calculation>
</calculationPeriodAmount>
```

- The **calculation** element wraps the calculation details.
- The **notionalSchedule** element specifies the notional. In this case, it is 10,000,000 USD, and does not change during the life of the swap.
- The **fixedRateSchedule** element specifies the fixed rate that is paid, 6%.

- The **dayCountFraction** element specifies that the day count fraction used is 30/360.

3.3.3.4. Floating Stream

With all the elements for the fixed stream defined we can now turn to the definition of the floating stream. A number of elements are similar to those in the fixed stream. There are however, several new elements describing the rules for the periodic rate reset; also, the calculation amounts are different than in the fixed stream.

Elements similar to fixed stream

Many of the elements in the floating stream are very similar to those in the fixed stream, and therefore are not shown here:

- The "payerPartyReference" and the "receiverPartyReference" are similar to those in the fixed stream, except that the parties are reversed in role.
- The calculation period dates element is very similar to that in the fixed stream. The only significant difference is that the calculation frequency is 3 Months (i.e., quarterly) instead of 6 Months.
- The payment dates element is also very similar to that in the fixed stream, except that the frequency is quarterly and the payments are based on the floating calculation periods.

Reset Dates

One element that is new in the floating stream is the resetDates element, which defines rules for resetting the floating rate. Within it, the fixingDates element describes how the rates are observed relative to the reset dates.

```
<resetDates id="resetDates">
   <calculationPeriodDatesReference href="fltDates" />
  <resetRelativeTo>CalculationPeriodStartDate</resetRelativeTo>
   <fixingDates>
      <periodMultiplier>-2</periodMultiplier>
      <period>D</period>
      <dayType>Business</dayType>
      <businessDayConvention>NONE</businessDayConvention>
      <businessCenters>
        <businessCenter>GBLO</businessCenter>
        <businessCenter>USNY</businessCenter>
      </businessCenters>
      <dateRelativeTo href="resetDates"/>
   </fixingDates>
   <resetFrequency>
      <periodMultiplier>3</periodMultiplier>
      <period>M</period>
   </resetFrequency>
   <resetDatesAdjustments>
      <businessDayConvention>MODFOLLOWING</businessDayConvention>
      <businessCentersReference href="busCtrs"/>
   </resetDatesAdjustments>
</resetDates>
```

- The **resetDates** element describes resetting rules, and the id attribute allows it to be referenced elsewhere in the document.
- The **calculationPeriodDatesReference** explicitly indicates which calculation periods these resets are related to, i.e., the floating calculation period dates.
- The **resetRelativeTo** says that the resets are done at the beginning of each calculation period (“in advance”).
- The **fixingDates** element describes how the rates are observed relative to the reset dates.
    - There is a 2 business day lag from rate observation to reset (it is expressed as -2 days because fixing is before resetting).
    - The business days used for determining the lag are New York and London business days.
    - The fixing dates are relative to the reset dates. See Section 4 for more on the “dateRelativeTo” element.
- The rate **resetFrequency** is quarterly (i.e., there is no averaging).
- Reset dates are **adjusted** using the modified following convention and the previously defined business centers (i.e., New York).

Floating calculation amounts

The calculation amounts element in the floating stream is similar in terms of notional, but is different in terms of the floating rate calculation. In this case the floating rate calculation is quite simple.

```
<calculationPeriodAmount>
  <calculation>
    <notionalSchedule>
      <notionalStepSchedule>
        <initialValue>10000000.00</initialValue>
        <currency>USD</currency>
      </notionalStepSchedule>
    </notionalSchedule>
    <floatingRateCalculation>
      <floatingRateIndex>USD-LIBOR-Telerate</floatingRateIndex>
      <indexTenor>
        <periodMultiplier>3</periodMultiplier>
        <period>M</period>
      </indexTenor>
    </floatingRateCalculation>
    <dayCountFraction>ACT/ACT.ISDA</dayCountFraction>
  </calculation>
</calculationPeriodAmount>
```

- The **notionalSchedule** element is the same as in the fixed stream.
- The **floatingRateCalculation** element describes how the floating rate is computed.
- The **floatingRateIndex** specifies the index (ISDA term, "Floating Rate Option") used for observing the rate. In this case it is USD-LIBOR from Telerate.
- The **indexTenor** element specifies the tenor of the index, in this case 3 Months.
- The **dayCountFraction** element specifies the day count fraction to be used for computing the calculation period, in this case ACT/ACT.ISDA basis.

3.3.3.5. Discussion

- A number of the building block elements described above are covered in more detail in Section 4 below, including message headers, trades, trade headers, date adjustments, and the dateRelativeTo element.
- The interest rate swap product itself is covered in detail in the Interest Rates section of the FpML Specification.
- The example represents a vanilla interest rate swap, omitting much of the complexity that is possible in a full FpML swap. Some of the items that were omitted include:
    - Compounding: This is represented by setting the payment frequency less than the calculation frequency, and adding elements to describe how compounding is done.
    - Averaging: This is done by setting the resetting frequency higher than the calculation frequency, and adding elements to specify the averaging method.
    - Stubs: Both long and short stubs can be specified at the beginning of each stream, and short stubs at the end. This is done by specifying extra dates in the calculationPeriodDates element, and by specifying stub rate calculation details in the calculationPeriodAmount element.
    - Uneven notionals (i.e., amortizing, accreting, or roller coaster): This can be done by supplying steps in the notional schedule.

- Uneven fixed rate: This can be done by supplying steps in the fixed rate schedule.
- Rate treatment, spreads, multipliers: A variety of calculations can be specified for floating rates.
- Cash flows: It is possible to list each cash flow explicitly, including the associated calculation periods and rate observations.
- Features: A variety of swap features (a.k.a. “break clauses”) can be specified, including:
    - Mandatory and optional early termination
    - Extension
    - Cancellation

4. The organization of FpML

4.1. Introduction

4.1.1. Purpose

This section summarizes key concepts underlying the design of the FpML standard, in particular some of the architectural principles. In addition, it describes some of the more important cross-product structures that are used widely throughout the standard.

4.1.2. Overview

The section is organized as follows:

- Architecture: a summary of FpML 2.2 architecture principles and how FpML is built on XML.
- Document/messaging structure: an introduction to FpML messaging.
- Key transaction structures: key structures used to represent trades in FpML.
- Reusable components: commonly reused building block components.
- Validation: an introduction to the FpML validation framework.

4.2. Architecture

4.2.1. Architecture Principles

The way that FpML 4 uses XML is documented in the FpML Architecture 2.2 available at http://www.fpml.org/spec.

Some key points that the FpML Architecture defines include:

- How the data is represented in XML. For example, business data should be held in XML element content, not in XML attributes. There is also coverage for “schemes”, which allow list values to be described.
- Naming conventions. For example, element and attribute names should be in “lowerCamelCase” (mixed capitalization, starting with a lower case).
- How to reference within and between documents. This is typically done with “id” and “href” attributes that link to them.
- How versioning is to be represented. This is done in part using the “version” attribute.
- How namespaces are to be used.
- How extensions are to be created.

Following is an example of how XML can be written, following the FpML architectural rules:

```
  <lowerCamelCaseElement>
   <anotherElementWithData>DataInElement</anotherElementWithData>
   <anElementReference href="target"/>
   <theTargetElement id="target">Some Data</theTargetElement>
  </lowerCamelCaseElement>
```

Following is an example of well-formed XML that doesn't follow the FpML architectural rules:

```
<ILLEGAL_ELEMENT_NAME>
  <BadElemName BadAttribute="Data Not Allowed Here"/>
</ILLEGAL_ELEMENT_NAME>
```

The errors in the last example include:

- Illegal naming conventions for elements and attributes (all should be lowerCamelCase)
- Use of attributes to hold business data

4.2.2. Schema and Instance Documents

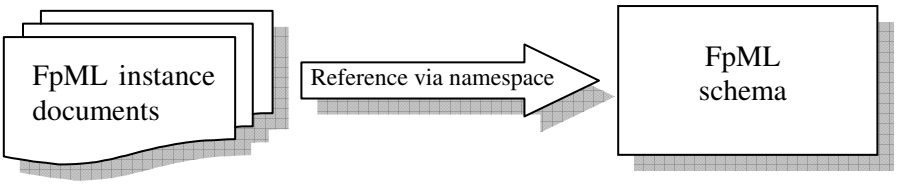
So far in this user guide we have focused on examples of FpML instance documents. In XML terms, these are called "instance" documents. These documents are instances (or if you prefer, examples) of documents whose structure is defined by a "schema" document. The schema defines the rules for building an instance document, i.e., "grammar" and "vocabulary", or "syntax". In much of the FpML specification documentation, the emphasis is on the schema, as opposed to the instances. This is because the instances will be different for every use, while the schema is the same for all.

The schema and the instances are linked as follows: The FpML root element in an FpML instance document has an "xmlns" (XML namespace) attribute, not described so far, that specifies the "namespace" (i.e., the vocabulary) that is used by the instance. The example below shows all of the attributes that the FpML root element will typically contain:

```
<FpML
 version="4-7"
 xmlns= "http://www.fpml.org/2009/FpML-4-7"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="RequestTradeConfirmation"
 xsi:schemaLocation="http://www.fpml.org/2009/FpML-4-7 ../fpml-main-4-7.xsd">
```

- The **version** attribute is an FpML attribute to define the FpML version that was used to create the document.
- The **xmlns** attribute specifies the namespace (in this case the FpML 4.7 namespace) that is followed by the document. This is the official link between the instance document and the schema. The XML document processor is supposed to use this namespace to find the schema that was used.
- The **xmlns:xsi** attribute defines the xsi namespace, so the parser will be able to understand what to do with the xsi:schemaLocation attribute described below (i.e., to recognize it as a special, pre-defined attribute).
- The **xsi:schemaLocation** attribute is an *optional* attribute that gives a hint to the XML document processor of where to find the schema document. Note that the XML document processor is free to ignore this and use its own version of the schema file.

The relation of these documents can be diagrammed as follows:



Two standard schema-related namespaces worth describing include the following:

- For XML schema itself, "*http://www.w3.org/2001/XMLSchema*". This namespace is used to indicate that the associated elements are being used to define an XML schema. This namespace is usually identified with the prefix "xsd" or "xs". It is normally found only in XML schema definition documents, not in instance documents.
- For XML schema instances, "*http://www.w3.org/2001/XMLSchema-instance*". This namespace is used within instance documents for hints or advice to XML document processors about how to use the XML schema associated with the instance document. It is usually identified with the prefix "xsi". It is normally found in instance documents and not in schema documents. In the case of FpML 4.7, we use the xsi namespace to tell the parser which message type to validate the document against.

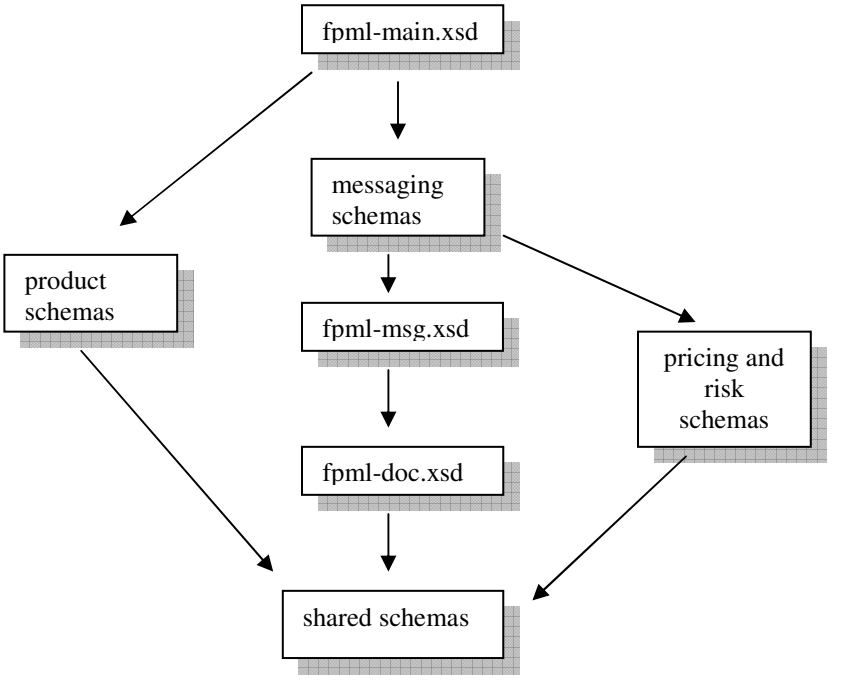
The FpML schema is also divided into a number of smaller subschema files for maintainability:

- The overall collection of subschema files is defined in fpml-main-4-x.xsd.
- Top level structures are defined in fpml-doc-4-x.xsd.
- Shared components and enumeration types are defined in fpml-shared-4-x.xsd and fpml-enum-4-x.xsd.
- Messaging is defined in fpml-msg-4-x.xsd and in several business process specific schema files, discussed in Section 6.
- Derivative products are defined in a variety of asset class-specific files (e.g., -irs-, -cd-, -eqd-, -fx-), and underlying assets are defined in fpml-asset-4-x.xsd.

where x stands for the minor FpML version. E.g., FpML 4.7.

This organization is documented in more detail in the introduction to the architecture in the FpML 4.7 specification.

Following is a summary of the dependencies of the different FpML sub-schemas.



4.2.3. Architectural Changes Between Versions

Over the years, FpML's architecture has changed slightly. The following gives an overview of the more significant changes that have occurred between the different versions. More detailed information on these changes and the rationale for them can be found in the respective versions where this change was first used.

- Up to version 3.0, the FpML syntax was defined in a document called a DTD (Document Type Definition). This format is more compact than XML schema, but not as powerful nor as descriptive. With the introduction of schema, there were a large number of changes in the internal organization of the schema.
- Top level structural changes. Between versions 1 and 3, a couple of structural changes were introduced, in particular:
    - In version 1.0, products were wrapped in a "product" tag. This was dropped from subsequent versions.

Version 1.0:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <product>
   <swap> . . . </swap>
 </product>
</trade>
```

Version 2.0+:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <swap> . . . </swap>
</trade>
```

    - Up to version 2.0, parties were within the trade element. They were moved out of the trade element in version 3.0.

Versions 1.0 – 2.0:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <swap>. . .</swap> <!-- or "product", or other product -->
 <party id="abc">
   <partyId>ABC123</partyId>
 </party>
</trade>
```

Versions 3.0 and 4.x:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <swap> . . . </swap> <!-- or other product -->
</trade>
<party id="abc">
 <partyId>ABC123</partyId>
</party>
```

- There have been some changes in the handling of list values :
    - Up to version 3.0, all of these were done using FpML "schemes", with scheme defaults in the FpML root element. Schemes look like the following:

```
<currency currencyScheme="http://schemeURI">USD</currency>
```

- From version 4.0 onward, some of the smaller and more stable schemes were moved to XML Schema "enumerations". Also, scheme defaults were removed from the FpML header element and put into the schema. Enumerations look like the following:

```
<businessDayConvention>FOLLOWING</businessDayConvention>
```

- Intra-document referencing syntax has changed slightly:
  - *Versions 1.0 – 2.0:*

```
<payerPartyReference href="#abc"/>
```

  - *Versions 3.0 – 4.x:*

```
<payerPartyReference href="abc"/>
```

- From version 4.0 onward, the majority of the elements are defined locally to avoid name collisions.

- Each FpML version has its unique namespace.

FpML 4.0 REC	http://www.fpml.org/2003/FpML-4-0
FpML 4.1 REC	http://www.fpml.org/2004/FpML-4-1
FpML 4.2 REC	http://www.fpml.org/2005/FpML-4-2
FpML 4.3 REC	http://www.fpml.org/2007/FpML-4-3
…	
FpML 4.7 REC	http://www.fpml.org/2009/FpML-4-7
FpML 4.8 REC	http://www.fpml.org/2010/FpML-4-8
FpML 5.0	http://www.fpml.org/FpML-5/confirmation (confirmation view) http://www.fpml.org/FpML-5/reporting (reporting view)

- Nothing significant changed from FpML 4.6 to 4.7 architecturally, other than the version numbers.

4.3. Document / Messaging Framework

FpML instance documents are all based on the "Document" type, in other words the "FpML" element is defined to be of type "Document". There are two main classifications (subtypes) of Documents:

- Data documents are documents that contain only data, not messages. These are relatively unstructured and are intended primarily for non-messaging uses or uses within proprietary messaging frameworks. They are represented using the "DataDocument" type.
- Message documents are intended to be used for communicating between firms or systems. They are represented using a variety of types, as described below.

Following are common characteristics of messages:

- Message documents are divided into three main types:
  - Notification messages are used to send unsolicited information.
  - Request messages are used to ask for something to be done.
  - Response messages are used to reply to "Request" messages.
- All messages have a message header. A message header contains information about the sender, the recipient, and various pieces of message identification information. The interest rate swap example in Section 3.3.3.1 shows an example of a simple message header. In addition, the message header can hold information such as:
  - "copyTo" addresses, for recipients that should get a copy of information.
  - An "inReplyTo" element, which can be used to indicate the message that this replies to.
  - A "conversationId", which can be used to link together a series of messages.
  - An "expiryTimestamp", for indicating when the message should be ignored as too old.
  - A digital signature, for authenticating the message sender.
- The user of an FpML instance document specifies the message type (or the use of DataDocument) by using an xsi:type attribute as described in section 3.3.3.1. For example, to specify that a document is a data document, you would set xsi:type="DataDocument". To specify that a document is a TradeStatus message, you would set its xsi:type="TradeStatus".

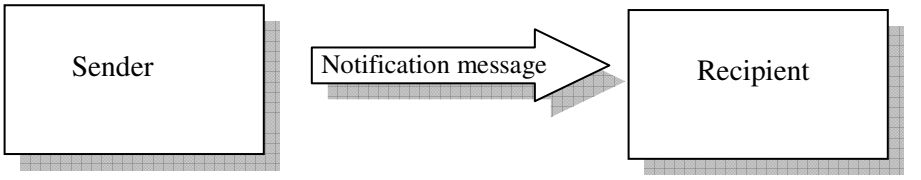
Sample message header

```
<FpML version="4-7" xsi:type="TradeConfirmed" ...>
 <header>
 <messageId messageIdScheme="http://abc.com/msg">A1</messageId>
 <inReplyTo messageIdScheme="http://def.com/msg">B2</inReplyTo>
 <sentBy>ABCDUS33</sentBy>
 <sendTo>DEFGGB2L</sendTo>
 <creationTimestamp>2010-04-02T15:38:00-00:00</creationTimestamp>
 <expiryTimestamp>2010-04-02T18:38:00-00:00</expiryTimestamp>
 </header>
 <!-- message content goes here -->
</FpML>
```

4.3.1. Sample Message Flows

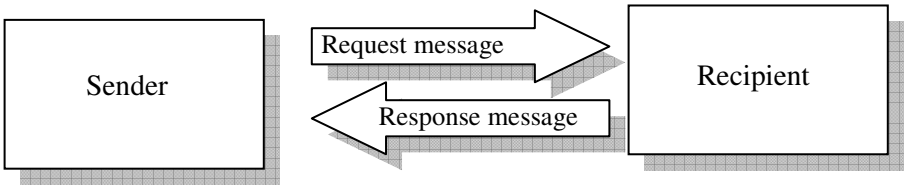
Notification Message Flow:

The standard notification message flow is a one way flow from the sender to the recipient. However, notification messages may have been triggered by a previous message, so they are allowed (but not required) to contain an "inReplyTo" element to identify that original message.



Request/Response Message Flow:

The standard request message flow is a two-way, request-response interaction. Typically, the response will indicate the message that is being responded to in the "inReplyTo" field.



4.3.2. More Information

The FpML specification provides more information in the message architecture section. It discusses the message types and structures in detail and describes several business processes using interaction diagrams.

It is important to note that the FpML Messaging framework is message transport independent. In other words, it is not linked to any specific messaging transport protocol. Thus it can be used with any message transport, from simple ones like e-mail or HTTP, to message-oriented middleware.

4.4. Other Top Level Structures

Some of the key, top level structures in an FpML instance document include:

- Party
- Trade, Trade Header
- Portfolio

4.4.1. Party

- The Party element is used to identify a participant in a transaction. This participant does not need to be a principal to the transaction. Instead, it could be a broker, arranger, agent, etc.
- Since version 4.2 the Party structure contains information about accounts.
- Until version 4.1, the roles of the parties were typically identified using party references. Since version 4.2 a complete set of roles are defined within the tradeSide structure (see following Trade section).
- For example, the following parties may be defined in an FpML instance document:

```
<party id="abc">
  <partyId>ABCDUS33</partyId>
  <account id="abcacc">
    <accountId>1234345</accountId>
  <account>
</party>
<party id="def">
  <partyId>DEFGGB2L</partyId>
  <account id="defacc">
    <accountId>789123</accountId>
  <account>
</party>
```

4.4.2. Trade

- The Trade element is used to hold transaction information. The Trade element includes
  - Identification information in a tradeHeader.
  - A product in the second slot, providing economic details of the transaction
  - Optional additional information in subsequent slots, including items such as:
    - otherPartyPayments, for fees and commissions
    - brokers
    - calculation agents
    - collateral
    - documentation
    - governing law
    - allocations
    - tradeSide, for party roles

Example trade structure, with many (not all) slots filled

```
<trade>
  <tradeHeader>
    <partyTradeIdentifier>
      <partyReference href="abc"/>
      <tradeId>A001</tradeId>
    </partyTradeIdentifier>
    <tradeDate>2010-01-14</tradeDate>
  </tradeHeader>
  <!-- product info skipped -->
  <collateral>
    <!-- collateral info goes here -->
  </collateral>
  <documentation>
    <!-- documentation info goes here -->
  </documentation>
  <governingLaw>GBEN</governingLaw>
  <tradeSide id="abc_side">
    <executor>
      <party href="xyz" />
    </executor>
    <creditor>
      <party href="abc" />
    </creditor>
    <calculater>
      <party href="abc" />
    </calculater>
  </tradeSide>
</trade>
```

4.4.3. Portfolio

The portfolio structure allows a collection of trades to be grouped together, identified by party trade identifier:

```
<portfolio id="port1">
   <tradeId id="t1" tradeIdScheme="http://b.com/tids">12</tradeId>
   <tradeId id="t2" tradeIdScheme="http://b.com/tids">23</tradeId>
   <tradeId id="t3" tradeIdScheme="http://b.com/tids">34</tradeId>
   <tradeId id="t4" tradeIdScheme="http://b.com/tids">45</tradeId>
   <tradeId id="t5" tradeIdScheme="http://b.com/tids">56</tradeId>
</portfolio>
```

4.5. Building Block Components

FpML has a large number of building block components that are used widely throughout the schema. The following types of components are described in more detail in this section:

- Currency and location related components
- Date-related components
- Payment

There are of course many other shared components. Some of these, not described here, include components for identifying legal entities and instruments and components for describing settlement instructions and documentation.

4.5.1. Currency and Location Related Components

There are a number of building block components for holding currency and location-related information. Some of these include:

- Currency
- Money
- Business Center, Business Centers
- Exchange ID
- Address

Several of these have been used in earlier examples. “Currency” is used to hold a currency code (by default an ISO currency code), while “Money” holds a currency and an amount. In the following, “paymentAmount” is of type “Money”:

```
<paymentAmount>
  <currency>USD</currency>
  <amount>1000000</amount>
</paymentAmount>
```

BusinessCenter holds a city or other business location, and BusinessCenters holds a collection of these. The default coding scheme for BusinessCenter is a four character SWIFT code in which the first two characters represent the country, and the last two represent the city. The following example represents London (GB + LO) and New York (US + NY) holidays:

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

This collection can be referenced by a BusinessCentersReference:

```
<businessCentersReference href="pmtBusCtrs"/>
```

This allows one set of business centers to be defined and then reused throughout a document. Most places where a list of business centers can be provided, a reference can be used instead.

For identifying exchanges, the “ExchangeId” type is used, e.g.,

```
<exchangeId>NYSE</exchangeId>
```

FpML does not currently standardize the values used in this scheme, so it is up to implementers to decide the coding scheme that they wish to use for this element.

The “Address” type and types contained within it can be used to identify locations. An example address is:

```
<routingAddress>
  <streetAddress>
    <streetLine>123 E 4 St.</streetLine>
  </streetAddress>
  <city>New York</city>
  <state>NY</state>
  <country>US</country>
  <postalCode>10003</postalCode>
</routingAddress>
```

4.5.2. Date-related Components

A number of components in FpML are used frequently for representing dates and date-related information. Some are described in the following sections.

4.5.2.1. Adjustable date, business day conventions

The Adjustable Date type holds an unadjusted date and adjustment rules (which are held in the "BusinessDayAdjustments" type). A number of elements throughout the schema are defined to be of type "AdjustableDate", or of type "BusinessDayAdjustments". The adjustments rules include a business day convention and a list of business centers. The following example states that the termination date is January 14, 2015, subject to the *Following* business day convention, using the payment business days defined in this FpML instance document:

```
<terminationDate>
  <unadjustedDate>2015-01-14</unadjustedDate>
  <dateAdjustments>
    <businessDayConvention>FOLLOWING</businessDayConvention>
    <businessCentersReference href="pmtBusCtrs" />
  </dateAdjustments>
</terminationDate>
```

In addition to the "AdjustableDates" type, there are several similar types that hold variations on this. These include:

- AdjustableDate2: Date adjustments can be omitted or referenced
- AdjustableDates: Holds several adjustable dates, using the same adjustment rules.
- AdjustableOrRelativeDate(s): Holds either AdjustableDate(s) or Relative Date(s) (described below).

4.5.2.2. Period, Frequency, Offset

In addition to date types that allow specific dates to be defined, there are date types that allow time periods to be defined. These include:

- **Period**: A defined number of periods.
- **Frequency**: A defined number of period but used for specifying payment or calculation frequencies at which the value T (Term) is applicable.
- **Offset**: A defined number of periods, additionally allowing Business or Calendar to be specified when the offset is in Days.

Period is commonly used to hold tenors, expressed in numbers of days, weeks, months, or years. The following expresses that the index tenor is 3 months:

```
<indexTenor>
  <periodMultiplier>3</periodMultiplier>
  <period>M</period>
</indexTenor>
```

The following example states that the payment days offset is 5 business days:

```
<paymentDaysOffset>
  <periodMultiplier>5</periodMultiplier>
  <period>D</period>
  <dayType>Business</dayType>
</paymentDaysOffset>
```

4.5.2.3. Date Relative To

The "dateRelativeTo" element allows a reference to a specific date to be included in a structure. This allows an unambiguous definition of the base date used to compute a relative date. The following states that something is to be defined relative to the trade date:

```
<tradeDate id="TradeDt">2010-01-02</tradeDate>
```

Then:

```
<dateRelativeTo href="TradeDt" />
```

4.5.2.4. Relative Dates

Leveraging the Offset and dateRelativeTo components described above, it is possible to define a date as being calculated relative to another date.

The RelativeDateOffset extends the Offset to include:

- The date adjustments used to compute the relative date.
- An explicit reference to the date upon which the relative date is based.

An example of this is the following:

```
<fixingDates>
  <periodMultiplier>-2</periodMultiplier>
  <period>D</period>
  <dayType>Business</dayType>
  <businessDayConvention>NONE</businessDayConvention>
  <businessCenters id="fixingBusinessCenters0">
      <businessCenter>EUTA</businessCenter>
  </businessCenters>
  <dateRelativeTo href="resetDates0"/>
</fixingDates>
```

This states that fixing dates are 2 Target (EUTA) Business Days before the reset dates identified by "resetDates0". No adjustment is necessary because "business" days are used for the offset.

RelativeDateOffset is used by the "RelativeDate" and "RelativeDates" type to allow a date or a series of dates to be defined relative to another date or date series.

4.5.3. Payment

Another structure used frequently throughout FpML is the "Payment" structure. It is used to represent a payment of a specified amount of money from one party to another at a specified time. It uses many of the components described above. This type is applied for defining payments such as the bullet payment in Section 3.2, premium payments, commission payments, and others.

A payment consists of:

- References to the payer and receiver
- The amount
- The unadjusted (and optionally adjusted) payment date
- Optional categorization and settlement information

An example payment is the following:

```
<premium>
    <payerPartyReference href="partyA"/>
    <receiverPartyReference href="partyB"/>
    <paymentAmount>
       <currency>EUR</currency>
       <amount>100000</amount>
    </paymentAmount>
    <paymentDate>
       <unadjustedDate>2010-08-30</unadjustedDate>
      <dateAdjustments>
         <businessDayConvention>FOLLOWING</businessDayConvention>
         <businessCenters>
            <businessCenter>EUTA</businessCenter>
         </businessCenters>
     </dateAdjustments>
   </paymentDate>
</premium>
```

Description: Party A pays Party B EUR 100,000 on August 30, 2010; subject to the Following convention using Target business days.

4.6. FpML Validation Framework

The FpML business rules validation framework allows business constraints to be applied to FpML instance documents to ensure that they are meaningful. This section discusses how FpML business rules validation relates to XML Schema validation, what is provided with the FpML validation framework, and how validation enhances the FpML specification.

4.6.1. What Does Validation Add?

To understand what FpML validation does, it will be useful to follow an example document as it passes through successive levels of increasingly stringent checking. In the following, we consider the case of a payment that will be made on January 14, 2011, subject to adjustment by the Following business day convention.

4.6.1.1. Well-formedness

A first attempt to represent this payment might be as follows:

```
Payment-date: January 14, 2011
Adjustment: Following
```

An XML parser will report errors with this document, as it is not well-formed XML (in fact, it is not XML at all). To resolve these errors, the document could be changed as follows:

```
<PaymentDate adjustment=”Following”>January 14, 2011</PaymentDate>
```

This is well-formed XML, and a non-validating XML parser will be able to parse this.

However, this is not FpML, so an FpML-based application trying to process this document will not be able to understand it, because the document structure doesn’t use FpML terminology and structuring concepts.

4.6.1.2. Syntactic Validation

Continuing with the above example, the document could be restructured into schema-valid FpML as follows:

```
<paymentDate>
  <unadjustedDate>2011-01-14</unadjustedDate>
  <dateAdjustments>
    <businessDayConvention>FOLLOWING</businessDayConvention>
  </dateAdjustments>
</paymentDate>
```

This document will now pass FpML schema validation when processed with a validating XML parser. This type of document can be described as "syntactically" valid or "schema valid". It says that the document uses names and structures in accordance with those defined in the schema.

4.6.1.3. Semantic Validation

A document can be syntactically valid and still not completely understandable, because some business meaning is missing or incorrect. In the preceding example, the sample document meets the requirements of the FpML schema, but the adjustments cannot be understood fully because the business days to be used for the adjustments are not specified. The FpML schema does not require the business days to be specified, because if the business day convention is "None" the business days are meaningless. However, in this case the business adjustments can only be fully understood if the business days are specified.

This type of validation is called "semantic" validation - it determines whether the FpML instance document is understandable. There are a series of business rules that an FpML instance document must pass to be meaningful. These business rules are in addition to the syntax rules defined by the schema.

The corrected document is as follows:

```
<paymentDate>
  <unadjustedDate>2011-01-14</unadjustedDate>
  <dateAdjustments>
    <businessDayConvention>FOLLOWING</businessDayConvention>
    <businessCentersReference href="primaryBusinessCenters"/>
  </dateAdjustments>
</paymentDate>
```

4.6.2. What Information Does Validation Provide?

FpML's validation rules are defined in the Validation Architecture section within the 4.7 Specification http://www.fpml.org/spec.

The FpML business validation rules define a large number of constraints on FpML instance documents similar to the above. For each constraint, the following is provided:

- An English-language version of the rule
- A technical expression of the rule
- Optionally, examples of documents that pass the rule (valid test cases)
- One or more examples of documents that fail the rule (invalid test cases)

A number of implementations of the rules have been developed in a variety of languages.

The rules that have been defined fall into a variety of categories. Some of the types of rules that have been defined include the following:

- **Date constraints**: dates must be in a particular order or pattern. For example, termination date must be after effective date.
- **Structural constraints**: if A is present, then B must also be. For example, if a stub period is defined, the payment amount calculations for it must also be defined.
- **Data value based constraints**: if element A has a given value, then …. For example, if the business day convention isn't NONE, the business centers must be supplied.
- **Special cases**: unusual handling / requirements for particular specialized products/ markets. For example, the SFE (Sydney Futures Exchange) has special date roll rules.

In addition, the FpML specification allows an FpML instance document to record which validation rule set(s) it is asserted to follow. This is done with the <validation> tag as follows:

```
<FpML version="4-7" xsi:type="RequestTradeConfirmation" . . . >
  <header>
    <messageId messageIdScheme="http://abc.com/msg">A01</messageId>
    <sentBy>ABCDUS33</sentBy>
    <sendTo>DEFGGB2L</sendTo>
    <creationTimestamp>2010-01-14T15:38:00-00:00</creationTimestamp>
  </header>
  <validation validationScheme="http://abc.com/rules">IRD</validation>
  <validation validationScheme="http://abc.com/rules">DEF-MSTR</validation>
</FpML>
```

This states that the document is believed to be valid according to ABC's IRD and DEF-MSTR rule sets. Presumably this is ABC's rules for interest rate products and its rules for its master agreement with DEF.

This information may be used by a recipient to process the message, but in many cases the recipient will apply its own validation policies independent of the sender's.

4.6.3. How Does Validation Work Together With the Specification?

The FpML Validation Rules defined in the specification are intended to be respected by all applications generating FpML, and as guidance to applications processing FpML on what validation rules to implement. These validation rules are considered to be required to be followed by an FpML application.

In addition, custom validation rules can also be defined in a similar way for other purposes. These might include:

- enforcing system constraints
- enforcing business policies
- enforcing rules defined by master agreements
- enforcing platform-specific policies

In this way, implementers of FpML-based applications can leverage tools using the FpML validation framework to identify and process validation constraints.

Since FpML 4.2, validation rules are published as a part of the FpML specification. Versions prior to FpML 4.2 have the validation rules and framework included in a separate package.

5. The FpML instrument framework

5.1. Introduction

This section describes FpML's framework for identifying financial products, both derivatives and simpler financial products used in defining derivatives. It is intended to explain how the type of product is specified and how new products are added, but not to explain any particular product in detail. The section is organized into two subsections:

- Derivative products
- Underlying instruments

5.2. Derivative Products

5.2.1. Product Substitution Framework

Derivative products form the core of what FpML is typically used to represent. They represent OTC derivatives products such as interest rate swaps, swaptions, FRAs, credit default swaps, equity swaps, commodity swaps, equity options, commodity options and FX options.

Where the trade element in FpML represents the non-economic information about a transaction (e.g., identifying information, documentation, etc...), the "Product" type represents the economic details of a transaction. While the possible "trade" details are similar for most or all transactions, the "Product" details differ depending on the type of financial product that was traded.

The basic trade structure is as follows:

```
<trade>
  <tradeHeader> . . . </tradeHeader>
  <product> . . . </product>
  <!-- more trade information -->
</trade>
```

However, after version 1.0 the "product" keyword described above never actually appears in the FpML instance document. Instead, the "product" tag is an "abstract" tag that is a member of a "substitution group". In practical terms, this means that instead of putting in a "product" tag, you can put in any element that is a member of the "product" substitution group.

For example, “creditDefaultSwap” is a member of this group, as is “equityOption”. That means that either of the following is allowed in FpML:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <creditDefaultSwap> . . . </creditDefaultSwap >
 <!-- more trade information -->
</trade>
```

Or:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <equityOption> . . . </equityOption>
 <!-- more trade information -->
</trade>
```

The first signals that the trade is a credit default swap, while the second signals that the trade is an OTC equity option.

Strategies

In addition, there is a special kind of product called a “Strategy” that combines other products. This allows the creation of complex structures by combining existing products.

The following is an example of a strategy, combining an FX option with an FX forward to create a delta-hedged option:

```
<trade>
 <tradeHeader> . . . </tradeHeader>
 <strategy>
   <productType>Delta-Hedge</productType>
   <fxSimpleOption>
     <productType>European FX Option</productType>
     <buyerPartyReference href="DEF"/>
     <sellerPartyReference href="ABC"/>
     <!-- more fx option details here -- >
   </fxSimpleOption>
   <fxSingleLeg>
     <!-- FX spot/forward details here -->
   </fxSingleLeg>
 </strategy>
</trade>
```

5.2.2. Product Summary

Following is a summary of the products covered by FpML, along with the FpML versions that the products were introduced in:

	Asset Class	Product	Product Variants	Since
	n/a	Strategy		3.0
Interest Rate Derivatives	**IRD**	bulletPayment		2.0
		capFloor		2.0
		fra		1.0
		swap	break clauses (cancelable, extendible, early termination), asset swap (since 4.2), inflation swap (since 4.2) Brazilian swaps (since 4.4)	
		swaption	American, European, Bermuda, Cash/Physical	2.0
Foreign Exchange	**FX**	fxSingleLeg	Forwards, Non-Deliverable Forwards	3.0
		fxSwap		3.0
		fxSimpleOption	knock-in and knock-out options	3.0
		fxAverageRateOption		3.0
		fxBarrierOption	dual and window barrier options	3.0
		fxDigitalOption	dual digital option	3.0
		termDeposit		4.0
Credit Derivativ	**CD**	creditDefaultSwap	CDS index (since 4.1), CDS basket (since 4.2), Loan CDS (since 4.3), CDS on Mortgage (since 4.3)	4.0
		creditDefaultSwapOption		4.3
Equity Derivatives	**Equity**	equityOption	various option features/exercise types	3.0
		equityOptionTransactionSupplement		4.1
		brokerEquityOption		4.1
		equityForward		4.1
		returnSwap (formerly equitySwap)		4.0*
		equitySwapTransactionSupplement		4.1
	Dividend	dividendSwapTransactionSupplement		4.3
	Variance	varianceSwap		4.3
		varianceSwapTransactionSupplement		4.3
		varianceOptionTransactionSupplement		4.6
	Correlation	correlationSwap		4.3
	Bond Options	bondOption		4.3
Commodities		commoditySwap	Physically-settled trades (since 4.6)	4.5
		commodityOption	Physically-settled options (since 4.8)	4.5
		commodityForward		4.6

* Indicates that the product was first defined in this version as equitySwap, however, the structure was changed substantially in 4.1 and specially 4.2, when it was renamed returnSwap. New implementations are strongly advised to use 4.2 for this product.

 FpML 5	The FX product model is being extensively enhanced and refactored in FpML 5.1. New implementations should consider using the FpML 5 FX framework instead of the 4.x model.

5.2.3. Adding New Products

To create a new product in your own namespace that can be used in an FpML trade, you should do the following:

- Create a type derived from the "Product" type, e.g., "MyProduct".
- Create a new global element, e.g., "myProduct", whose type is "MyProduct". Make that element a member of the "fpml:product" substitution group.

See Section 7 for more on extending FpML.

5.3. Underlying Assets

5.3.1. Usage

In addition to representing complex derivative products, FpML has a representation of a fairly large number of simple, standardized financial instruments. These instruments, called "UnderlyingAssets" in FpML, can be used for a variety of purposes:

- As underlying assets in various derivatives, including:
  - Equity options
  - Equity swaps
  - Asset swaps
- As reference obligations in credit default swaps
- For a variety of purposes in pricing and risk, including:
  - For describing curve inputs
  - For describing benchmark asset prices

It is also expected that use of the underlying assets will increase in future versions of FpML.

5.3.2. Underlying Asset Substitution Framework

The underlying asset framework is very similar to the product framework. In places where underlying assets are used, a substitution group can allow the asset to be substituted as required. However, underlying assets are different from "products" in the sense that all are derived from UnderlyingAsset rather than Product. In addition, UnderlyingAsset defines some standard data fields available for all assets.

For example, "equity" is an FpML underlying asset, and here is a use of "equity" as a basket component:

```
<basketConstituent>
    <equity>
        <instrumentId instrumentIdScheme="http://www.fpml.org/coding-
           scheme/external/instrument-id-bloomberg">TIT.ME</instrumentId>
        <description>Telecom Italia spa</description>
        <currency>EUR</currency>
        <exchangeId exchangeIdScheme="http://www.fpml.org/coding-
      scheme/external/exchange-id-MIC">Milan Stock Exchange</exchangeId>
```

```
    </equity>
    <constituentWeight>
      <openUnits>432000</openUnits>
    </constituentWeight>
</basketConstituent>
```

However, if instead a “bond” a different FpML underlying asset, were desired, the FpML might look like:

```
<basketConstituent>
   <bond>
      <instrumentId
          instrumentIdScheme="http://www.fpml.org/spec/2002/instrument-id-
          ISIN-1-0">JP310860A032</instrumentId>
      <couponRate>0.0213</couponRate>
      <maturity>2011-03-08</maturity>
   </bond>
   <constituentWeight>
      <openUnits>432000</openUnits>
   </constituentWeight>
</basketConstituent>
```

5.3.3. Summary of Underlying Assets

The following table summarizes the underlying assets available in FpML. All of these assets are defined in "fpml-asset-4-7.xsd".

Underlying Asset	Description
bond	a security typically delivering interest coupon payments and requiring the repayment of a principal amount at its maturity
cash	an asset in monetary form, typically held in a bank account
commodity	a commodity underlying asset
convertibleBond	a bond that can under specified circumstances be converted into equity (e.g., common stock) in the issuer
deposit	a term deposit, a money market instrument of fixed duration yielding a specific interest rate
equity	an ownership share in an entity, typically common stock
exchangeTradedFund	a fund whose units can be traded on an equity exchange
future	identifies the underlying asset when it is a listed future contract
fxRate	a currency pair whose exchange rate can be quoted on the open market
future	a standardized, daily-settled contract traded on an exchange for the purchase or sale of an asset at some specified date in the future
index	an asset whose value is based on the value of a set of instruments, typically equities
loan	a simple underlying asset that is a loan
mortgage	a mortgage backed security
mutualFund	a pooled investment vehicle that takes positions in a variety of financial instruments, typically equities
rateIndex	an interest rate index, such as USD LIBOR
simpleFra	a simple, benchmark Forward Rate Agreement
simpleIrSwap	a simple, benchmark Interest Rate Swap
simpleCreditDefaultSwap	a simple, benchmark Credit Default Swap

6. The FpML Business Process Framework

6.1. Introduction

FpML has provided support for representing business processes since version 4.0. This section describes how business processes are represented in FpML 4, summarizing them into several categories, de-pending on where they happen within the trade lifecycle.

FpML models business processes using a combination of messages, whose content is normative, and workflow diagrams, whose sequence is non-normative. This means that differing implementations of the same workflow will use the same message structure. However, they may differ slightly in when those messages are sent and in the detailed message contents, for example in the message identification.

Because of the large number of messages, the messages are divided into a number of categories, according to whether the messages are sent prior to trade execution, during the confirmation process, or afterward. One other category of messages covers those that summarize a large number of trades.

The FpML messaging introduction in the FpML 4.x series contains extensive diagrams modeling how the interactions are intended to operate. Developers interested in better understanding these business process definitions are directed to that documentation.

The existing business processes mostly leverage the product substitution framework described in Section 5. This framework was originally developed for confirming trades, and may evolve somewhat in future versions of FpML to better accommodate the needs of the various business processes.

 FpML 5	The messaging framework has been streamlined and refactored in FpML 5.0. Firms new to FpML and interested in implementing FpML messaging should consider using the FpML 5 messaging framework.

6.2. Pre-trade

Pre-trade business processes are defined in the fpml-pretrade-4-7.xsd schema file. The main process is a Request For Quote (RFQ) process, with a mechanism for accepting and executing a quote. This process has been in FpML since version 4.0.

This area also defines a “QuotableProduct”, which is a simplified version of a product that is suitable for RFQ purposes. Currently QuotableProduct is implemented only for FX spots, but it is anticipated that this will be expanded to other products in future versions of FpML.

Pre-trade messages include:

- RFQ:
  - RequestQuote
  - Quote (formerly RequestQuoteResponse in FpML 4.3)
  - QuoteUpdated
- Quote acceptance/execution
  - AcceptQuote

  - TradeExecution (formerly QuoteAcceptanceConfirmed in FpML 4.3)
  - QuoteAlreadyExpired

The FpML messaging specification has diagrams explaining how the overall RFQ interaction works.

6.3. Trade Execution/Confirmation

Trade execution and confirmation messages are defined in the fpml-tradeexec-4-7.xsd file. These relate primarily to confirmation matching and affirmation. This area has been in FpML since version 4.0.

Messages include:

- Affirmation
  - TradeAffirmation
  - TradeAffirmed
  - TradeAlreadyAffirmed
- Confirmation
  - RequestTradeConfirmation
  - ModifyTradeConfirmation
  - CancelTradeConfirmation
  - TradeMatched
  - TradeMismatched
  - TradeUnmatched
  - TradeAlleged
  - ConfirmTrade
  - ConfirmationCancelled
  - TradeConfirmed
  - TradeAlreadySubmitted
  - TradeAlreadyConfirmed
- Matching
  - RequestTradeMatch
  - ModifyTradeMatch
  - CancelTradeMatch
  - TradeAlreadyMatched
  - TradeMatched
  - TradeMismatched
  - TradeUnmatched
  - TradeAlreadySubmitted

Example workflows for these business processes are described and diagrammed in the specification. To summarize, the processes are used as follows:

- Affirmation is used when one party generates the trade, and the other accepts or rejects the version supplied by the first party.
- Matching is used to compare trades from two sources and report matches and differences.
- Confirmation is an extended version of matching, intended specifically for confirming trades.

6.4. Post-Trade

The post-trade business processes describe events that happen after a trade has been confirmed. These include processes such as novation, termination, amendment, etc. The related post trade messages are defined in fpml-posttrade-4-7.xsd. Most of the messages in this area were first added in FpML 4.1.

Post-Trade messages in FpML 4.7 include:

- Application to application (A2A)
  - TradeExecution (formerly TradeCreated in FpML 4.3)
  - TradeExecutionModifiied (formerly TradeAmended in FpML 4.3)
  - TradeExecutionCancelled (formerly TradeCancelled in FpML 4.3)
- Novation - negotiation
  - NovationConsentRequest
  - NovationConsentGranted
  - NovationConsentRefused
- Novation - execution
  - NovateTrade
  - TradeNovated
  - RequestNovationConfirmation
  - NovationMatched
  - NovationAlleged
  - NovationConfirmed
- Termination
  - TradeTerminationRequest
  - TradeTerminationResponse
  - RequestTerminationConfirmation
  - TerminationConfirmed
- Trade Increase
  - TradeIncreaseRequest
  - TradeIncreaseResponse
  - RequestIncreaseConfirmation
  - IncreaseConfirmed
- Amendments
  - TradeAmendmentRequest
  - TradeAmendmentResponse
  - RequestAmendmentConfirmation
  - AmendmentConfirmed
- Credit Events
  - CreditEventNotification
- Allocations
  - RequestAllocation
  - AllocationAmended
  - AllocationCancelled
  - AllocationCreated
- Cash flow matching
  - TradeCashflowsAsserted
  - TradeCashflowsMatchResult
  - CancelTradeCashflows

- Portfolio reconciliation
    - RequestPortfolio
    - PositionsAsserted
    - PositionsAcknowledged
    - PositionsMatchResults

Contract Notification messages include:

- ContractCreated
- ContractIncreased
- ContractIncreasedCancelled
- ContractNovated
- ContractNovatedCancelled
- ContractCancelled
- ContractFullTermination
- ContractFullTerminationCancelled
- ContractPartialTermination
- ContractPartialTerminationCancelled
- ContractAmended
- ContractAmendedCancelled
- ContractChanged
- ContractChangedCancelled

These messages are documented in the FpML 4.7 Specification.

Business processes and the related messages is an area that has grown greatly in FpML 4.7 and is expected to continue to grow in upcoming versions of FpML. Some other post-trade business processes that are likely to be covered in future versions of FpML include:

- Option exercise, knock in/out notification, etc.

6.5. Reporting

The reporting business processes represent information about a collection of trades. This area was first added in FpML 4.1, and currently includes only valuation and position reporting. In future versions of FpML this area is expected to expand position reporting further and cover activity reporting, collateral reporting, client statements, etc. Reporting messages are defined in fpml-reporting-4-7.xsd.

Reporting Messages:

- Valuation messages
  - RequestValuationReport
  - ValuationReport
- Position messages
  - RequestPositionReport
  - PositionReport

Despite the small number of messages, this is a functionally rich area, as there is a great deal of flexibility in the valuation structures underlying these. For example, in both the requests and the responses, the following types of information may be specified:

- Lists of trades or portfolios, or portfolio definition criteria.
- Market data used in the messages, including items such as:
  - benchmark instrument prices
  - market inputs, such as yield or credit curves, and volatility matrices
- Valuation scenarios (to be) applied, such as valuation dates and shifts to market inputs.
- Sensitivity calculation definitions
- Valuation results, allowing:
  - a wide variety of valuation measures to be used;
  - a number of quotation characteristics to be supplied, such as side, quotation units, location, time of day, etc; and
  - sensitivities to be reported, linked to base values and to sensitivity definitions.

 FpML 5	• The reporting messaging framework is being enhanced in FpML 5.x to define data elements to be used for reporting OTC derivative positions between market participants and to regulators. • The improved reporting framework will support portfolio reconciliation, e.g. for collateral review, and regulatory reporting, including the industry-wide portfolio repository, industry wide collateral management approach, and regulatory and public reporting. • We advise using the confirmation representation of products for position reporting. • See the FpML 5.0 specification on the FpML.org website for the latest reporting developments.

6.6. Loan Syndication

The Loan messaging framework describes Syndicated Loan notifications between Agent Bank and Lenders. This area was first added in FpML 4.4, and currently includes the following notifications:

- Loan Contract Level Notifications
  - DrawdownNotice
  - InterestPaymentNotice
- Facility Level Notifications
  - OneOffFeeNotice
  - OnGoingFeeNotice
  - RepaymentNotice
  - LcIssuanceNotice
  - LcBalanceNotice
  - LcTerminationNotice
  - LcAmendmentNotice
  - PricingChangeNotice

 FpML 5	• The Loan messaging framework continues to be developed in FpML 4.x series but will be refactored in FpML 5. Firms who have not started implementation should consider loan support in FpML 5. • Note that FpML 5.0 does not include support for Syndicated Loans. • A new, refactored loan framework is being developed and will be included in FpML 5.1 and beyond.

7. Customizing FpML

7.1. Introduction

7.1.1. Purpose of the Section

This section discusses techniques for adjusting FpML to better meet specific application requirements. This includes adding information not included in FpML as well as preventing FpML features from being used when they are not required.

Many of the techniques described here are described in more detail in the FpML Architecture Specification 2.2, section 6 (http://www.fpml.org/spec).
These techniques apply to FpML 4.x and 5.x.

7.1.2. Overview

The section is organized as follows:

- **Wrapping**: a traditional approach for extending FpML.
- **Type extension**: a more modern approach for extending FpML.
- **Type restriction**: a more modern approach for constraining FpML.
- **Extending product coverage**: detail on adding product coverage.
- **Extending message/business process coverage**: detail on adding messages.
- **Migrating extensions**: a discussion on managing extensions to FpML as FpML evolves.

7.2. Wrapping

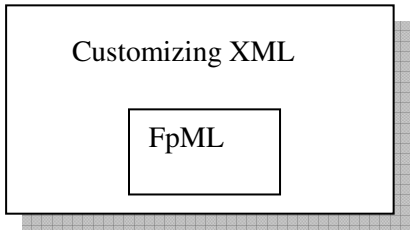
7.2.1. Explanation

Wrapping is a traditional technique for extending FpML. It has been used by a number of firms implementing FpML since the earliest versions of FpML.

In wrapping, the FpML is contained within an XML "wrapper" that contains elements that are not available in FpML but that are required for the application. Below is a simplified example:

```
<abcML version="0.1">
  <additionalData>123</additionalData>
  <moreAdditionalData>456</moreAdditionalData>
  <FpML version="4-7" xmlns="http://www.fpml.org/2009/FpML-4-7" >
    <trade>
      <!-- detail omitted -->
    </trade>
    <!-- parties omitted -->
  </FpML>
<abcML>
```

The following diagram illustrates the "wrapping" approach for extending FpML:



7.2.2. Advantages and Disadvantages

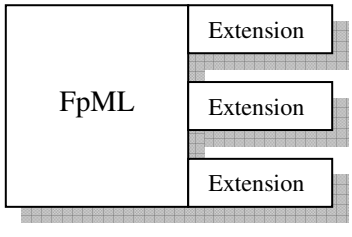
Some of the advantages and disadvantages of the "wrapping" approach include:

- Pros
  - This approach is straightforward and simple.
  - It has been used widely for several years by a number of dealers and has been proved to be successful.
  - The distinction between customized and standard FpML is clear.
- Cons:
  - It's not easy to add elements to structures that are repeated more than once.
  - Linking extensions to the original data can be hard.
  - This approach does not support restricting the contents of the FpML.

Because of the disadvantages listed above, the FpML Architecture 2.2 does not recommend using the "wrapping" approach for new implementations.

7.3. Extending Type Content

The recommended approach for extending FpML is to extend the types defined by FpML with new elements that are in the customizing firm's namespace. This approach is documented in the FpML Architecture 2.2 document in Section 6.4.



FpML Namespace Custom Namespace

7.3.1. Example

In FpML 4.x, the "AdjustableDate" type is defined as having an *unadjusted date* and *date adjustments*. Let's assume that ABC, a firm implementing an FpML-based application, would like to add a new element, "adjustedDate", to that type.

The type extension would be defined in the ABC's schema (AbcML) as follows:

```
<xsd:schema xmlns = "http://www.abc.com/AbcML"
    xmlns:fpml = "http://www.fpml.org/2009/FpML-4-7"
```

```
 xmlns:dsig = "http://www.w3.org/2000/09/xmldsig#"
 targetNamespace = "http://www.abc.com/AbcML"
 xmlns:abc = "http://www.abc.com/AbcML"
 xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
 elementFormDefault = "qualified"
 attributeFormDefault = "unqualified">

<xsd:import namespace= "http://www.fpml.org/2009/FpML-4-7"
   schemaLocation="fpml-main-4-7.xsd" />

    <!-- ***** Extension of AdjustableDate ********* -->

<xsd:complexType name = "AdjustableDate">
   <xsd:annotation>
      <xsd:documentation>ABC extension to FpML Adjustable Date
       type</xsd:documentation>
   </xsd:annotation>
   <xsd:complexContent>
      <xsd:extension base = "fpml:AdjustableDate">
         <xsd:sequence>
            <xsd:element name="adjustedDate" type="xsd:date"
                 minOccurs="0" >
               <xsd:annotation>
                  <xsd:documentation xml:lang="en">The date after
                   adjustment using the adjustment conventions.
                  </xsd:documentation>
               </xsd:annotation>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

Then an AbcML document could use the new type in existing FpML products by using the "xsi:type" attribute to indicate that the new type is being used. The new element is in the "abc" namespace, so it requires an appropriate prefix. For example, assuming that the AbcML schema were being used and the "abc" prefix pointed to that namespace, in a payment the user could do the following:

```
<FpML version = “4.7”
    xmlns = "http://www.fpml.org/2009/FpML-4-7"
    xmlns:abc = "http://www.abc.com/AbcML"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">
    …

    <payment>
      <payerPartyReference href=“abc”/>
          <receiverPartyReference href=“def”/>
            <paymentAmount>
              <currency>GBP</currency>
              <amount>15000000.00</amount>
          </paymentAmount>
          <paymentDate xsi:type=”abc:AdjustableDate”>
            <unadjustedDate>2011-01-14</unadjustedDate>
            <dateAdjustments>
            <businessDayConvention>FOLLOWING</businessDayConvention>
              <businessCenters>
                  <businessCenter>GBLO</businessCenter>
                  <businessCenter>USNY</businessCenter>
              </businessCenters>
            <abc:adjustedDate>2011-01-17</abc:adjustedDate>
          </dateAdjustments>
       </paymentDate>
     </payment>
```

7.3.2. Advantages and Disadvantages

- Pros
  - Allows extension to existing FpML elements in-place.
  - Supports extending repeated elements.
  - Extensions are closely linked to the original FpML, making it easy to determine the relation of the extension to the original FpML.
  - Extensions are clearly documented in the instance documents because they are in a different namespace.
  - It’s relatively easy to migrate extensions to a new version of FpML.
  - This approach is a recommended approach in the FpML 2.2 Architecture.
- Cons
  - Requires the use of multiple namespaces and namespace-aware applications.
  - XPath expressions may be slightly longer due to the need for namespace prefixes.

7.4. Restricting Type Content

In many cases, specific implementations may wish to restrict the ability of users to use specific elements or options. To implement these restrictions, implementers may use the schema “redefine” capability to remove optional elements, make optional elements mandatory, etc. This technique is described in detail in the FpML Architecture Specification 2.2, in Sections 6.5 to 6.7.

7.5. Extending Product Coverage

The first step in creating a new product is the creation of a new product type based on the FpML "Product" type.

7.5.1. Create a New Type

First, create a new product type based on the FpML "Product" type.

In the following example we create a type called "Forward" that is a forward purchase/sale of an FpML underlying asset (such as a bond or a stock). Note that this product is simplified compared what you might require for a real implementation, e.g., it does not include any support for settlement date or amount, ignores commissions and accrued interest, for example. However, it does convey some of the information one might require for such a product, such as the buyer and seller, the asset, the quantity, the price (and provision for the price units, etc…), and the forward sale date.

```
<xsd:complexType name = "Forward">
  <xsd:annotation>
    <xsd:documentation>A forward transaction on an
    underlying asset.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base = "fpml:Product">
      <xsd:sequence>
        <xsd:group ref="fpml:BuyerSeller.model"/>
        <xsd:element ref="fpml:underlyingAsset" />
        <xsd:element name="quantity" type="xsd:decimal"/>
        <xsd:element name="price" type="xsd:decimal" />
        <xsd:group ref="fpml:QuotationCharacteristics.model"/>
        <xsd:element name="fwdDate" type="fpml:AdjustableDate"/>
      </xsd:sequence>
    </xsd:extension>
  <xsd:complexContent>
</xsd:complexType>
```

Next create a new global element that is a member of the "product" substitution group. This will allow the newly created product to be included in FpML trades and other places that require FpML products:

```
<xsd:element name="forward" type="abc:Forward"
  substitutionGroup="fpml:product"/>
```

To use this product, reference it as “abc:forward”, for example:

```
<trade>
 <tradeHeader>
   <! -- details deleted -->
 </tradeHeader>
 <abc:forward>
   <buyerPartyReference href=”abc” />
   <sellerPartyReference href=”def” />
   <bond>
     <!-- details omitted -->
   <bond>
   <abc:quantity>100</abc:quantity>
   <abc:price>100.5</abc:price>
   <quoteUnits>PctOfParValue</quoteUnits>
   <abc:fwdDate>
     <unadjustedDate>2011-01-14</undadjustedDate>
     <dateAdjustments>
       <!-- details omitted -->
     </dateAdjustments>
   </abc:fwdDate>
 </abc:forward>
</trade>
```

Note that in the above example, the document is assumed to have its default namespace to be the FpML namespace, so no prefixes are required for FpML elements. This includes any elements that are included using FpML groups (such as the BuyerSeller.model or the QuotationCharacteristics.model), or referencing FpML substitution groups (such as the underlyingAsset substitution group).

A different example is provided in the FpML Architecture Specification, Section 6.2.

7.6. Extending an Existing Product

The process for extending an existing product is similar to that described above, except that instead of extending “fpml:Product”, one extends the specific product, for example “fpml:Swap”. Then one has the choice of either creating a new member of the “product” substitution group, as described above, or not bothering, and just using the “xsi:type” attribute to indicate that a proprietary extension was used. In the first approach, the trade would have the following structure:

```
<trade>
 <tradeHeader>
   <! -- details omitted -->
 </tradeHeader>
 <abc:swap>
   <!-- details omitted -->
 </abc:swap>
</trade>
```

In the second approach, the trade would look something like this:

```
<trade>
 <tradeHeader>
   <! -- details omitted -->
 </tradeHeader>
 <swap xsi:type=”abc:Swap”>
   <!-- details omitted -->
 </swap >
</trade>
```

In either case the extended elements would normally be in the “abc” namespace and therefore would require a namespace prefix.

This topic is also discussed in the FpML Architecture Specification, Section 6.2.

7.7. Extending Messages

To create a new message, one must create a new message Complex Type extending one of the existing base message types, namely:

- NotificationMessage
- RequestMessage
- ResponseMessage

For example, assume that one wants to create a new notification message called “Trade Report”, with a report date and a collection of trades, plus the associated parties. The message definition would be as follows:

```
<xsd:complexType name = "TradeReport">
 <xsd:complexContent>
  <xsd:extension base = "fpml:NotificationMessage">
   <xsd:sequence>
    <xsd:element name="reportDate" type="xsd:date"/>
    <xsd:element name="trade" type="fpml:Trade"
     maxOccurs="unbounded"/>
    <xsd:element name="party" type="fpml:Party"
       maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Then, to use the new message type, one creates an FpML instance document whose xsi:type matches that of the new message, e.g.:

```
<FpML version = "4.7"
    xsi:type="abc:TradeReport"
    xmlns = "http://www.fpml.org/2009/FpML-4-7"
    xmlns:abc = "http://www.abc.com/AbcML"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">

   <header>
     <messageId messageIdScheme="http://abc.com/ms">A1</messageId>
     <sentBy>ABCDUS33</sentBy>
     <sendTo>DEFGGB2L</sendTo>
     <creationTimestamp>2010-01-14T15:38:00-00:00</creationTimestamp>
   </header>
   <abc:reportDate>2010-01-01</abc:reportDate>
   <abc:trade>
        <!-- trade details omitted -->
   </abc:trade>
   <abc:trade>
        <!-- trade details omitted -->
   </abc:trade>
   <abc:party>
        <!-- party details omitted -->
   </abc:party>
   <abc:party>
        <!-- party details omitted -->
   </abc:party>
</FpML>
```

Another example of this is provided in the FpML Architecture Specification, Section 6.3

7.8. Versioning and Version Migration

When customizing FpML, one needs to be aware that both FpML and the customizations will evolve over time. In the proprietary schema, a mechanism for tracking the version of the customizations needs to be specified, in addition to the mechanisms specified in FpML to record the FpML version.

To migrate to a new version of the customizations without changing the FpML version, update the customized schema and its version mechanisms, preserving the references to FpML unchanged.

To migrate to a new version of FpML, one will normally issue a new version of the customized schema that references the updated version of FpML.

FpML 4 Underlying Assets

Underlying Asset	Description
bond	a security typically delivering interest coupon payments and requiring the repayment of a principal amount at its maturity
cash	an asset in monetary form, typically held in a bank account
commodity	a commodity underlying asset
convertibleBond	a bond that can under specified circumstances be converted into equity (e.g., common stock) in the issuer
deposit	a term deposit, a money market instrument of fixed duration yielding a specific interest rate
equity	an ownership share in an entity, typically common stock
exchangeTradedFund	a fund whose units can be traded on an equity exchange
future	identifies the underlying asset when it is a listed future contract
fxRate	a currency pair whose exchange rate can be quoted on the open market
future	a standardized, daily-settled contract traded on an exchange for the purchase or sale of an asset at some specified date in the future
index	an asset whose value is based on the value of a set of instruments, typically equities
loan	a simple underlying asset that is a loan
mortgage	a mortgage backed security
mutualFund	a pooled investment vehicle that takes positions in a variety of financial instruments, typically equities
rateIndex	an interest rate index, such as USD LIBOR
simpleFra	a simple, benchmark Forward Rate Agreement
simpleIrSwap	a simple, benchmark Interest Rate Swap
simpleCreditDefaultSwap	a simple, benchmark Credit Default Swap

(See page 52)

FpML 4 Products

	Asset Class	Product	Product Variants	Since
	n/a	Strategy		3.0
Interest Rate Derivatives	**IRD**	bulletPayment		2.0
		capFloor		2.0
		fra		1.0
		swap	break clauses (cancelable, extendible, early termination), asset swap (since 4.2), inflation swap (since 4.2) Brazilian swaps (since 4.4)	
		swaption	American, European, Bermuda, Cash/Physical	2.0
Foreign Exchange	**FX**	fxSingleLeg	Forwards, Non-Deliverable Forwards	3.0
		fxSwap		3.0
		fxSimpleOption	knock-in and knock-out options	3.0
		fxAverageRateOption		3.0
		fxBarrierOption	dual and window barrier options	3.0
		fxDigitalOption	dual digital option	3.0
		termDeposit		4.0
Credit Derivatives	**CD**	creditDefaultSwap	CDS index (since 4.1), CDS basket (since 4.2), Loan CDS (since 4.3), CDS on Mortgage (since 4.3)	4.0
		creditDefaultSwapOption		4.3
Equity Derivatives	**Equity**	equityOption	various option features/exercise types	3.0
		equityOptionTransactionSupplement		4.1
		brokerEquityOption		4.1
		equityForward		4.1
		returnSwap (formerly equitySwap)		4.0*
		equitySwapTransactionSupplement		4.1
	Dividend	dividendSwapTransactionSupplement		4.3
	Variance	varianceSwap		4.3
		varianceSwapTransactionSupplement		4.3
		varianceOptionTransactionSupplement		4.6
	Correlation	correlationSwap		4.3
	Bond Options	bondOption		4.3
Commodities		commoditySwap	Physically-settled trades (since 4.6)	4.5
		commodityOption	Physically-settled options (since 4.8)	4.5
		commodityForward		4.6

(See page 49)