Financial Products Markup Language

Overview

FpML Working Draft 1.0 b2

Abstract

The Financial Product Markup Language (FpML) is a business information exchange standard for electronic dealing and processing of Financial Derivative instruments. FpML is based on the Extensible Markup Language (XML) and initially focuses on Interest Rate Derivatives, FX Spot and FX Derivative Products. FpML has been designed to be modular, easy to use and in particular, intelligible to practitioners in the financial industry. This working draft is presented in three parts: (1) Overview, (2) DTDs and associated information components, and (3) Corresponding reference FpML for a representative set of Financial instruments in Interest Rate and FX markets. The overview includes business and design goals, the scope and architectural framework of FpML.

AUTHORS

J.P.Morgan:
Tayo Ibikunle, Keri Jackson, Axel Kramer, Colin McDonald, Ramesh Parmar, Bob Sieber

PricewaterhouseCoopers:
Waqar Ali, Philip J. Lee, Mahesh Panjwani

Status of this document:

This is the FpML Working Draft for review by the FpML Technical and Standard Committees. It is a draft document and may be updated, replaced, or made obsolete by other documents at any time. The FpML Standards committee will not allow early implementation to constrain its ability to make changes to this specification prior to final release. It is inappropriate to use the FpML working draft as reference material or to cite it as other than "work in progress".

Acknowledgements:

J.P.Morgan:
Michael Ashworth, Lev Berlin, Sunil Biswas, Guy Gurden, Mahmood Hanif, Ross Hamilton, Oliver Herzefeld, Mary Hurd, John O'Hara, Thorkild Juncker, Nigel Khakoo, Philippe Khuong-Huu, Kerry LeClue, Mike Liberman, Stuart Marker, Benjamin McGill, Andrew Mead, Andy Miller, Gordon Middleton, Chris Molanphy, Eun Ju Park, Nicola Pearce, Neil Penny, Heather Rice, Jeff Saltz, Keith Salway, Ian Shectman, Oliver Stevens, Vlad Torgovnik, Ed Volkwein, Phil Weisberg

PricewaterhouseCoopers:
Aleksandr Barg, Chris Curt, Fred Fuchs, Norbert Gehrke, Adam Greissman, Ed Hoofnagle, Stacey Joyce, Kwong Li, Tom Little, Sanjeev Midha, Andrew Price, Juan Pujadas, Lon McQuillin, Vijay Pant, Sam Salameh, Doug Summa, Shyam Venkat

TABLE OF CONTENTS

1 INTRODUCTION

Financial Product Markup Language, abbreviated FpML, is a new information exchange standard for Internet-based electronic dealing and processing of financial instruments, initially focusing on interest rate and foreign exchange derivative products. Based on XML (eXtensible Markup Language), the emerging Internet standard for data sharing between applications, FpML enables Internet-based integration of a range of services, from electronic trading and trade confirmations to risk and sensitivity analysis of trade portfolios.

The highly conditional and time-dependent characteristics of derivative products have often required complex processes and/or systems to support their pricing, trading, risk management, settlement and confirmation. Over time people have become more adept at mastering the complexity of derivative products in single proprietary, single business processing systems. However, there continues to be considerable difficulty in determining the structure and content of what can be exchanged between systems in order to support various forms of end-to-end processing of these products. FpML has been defined to help manage such complexities.

By specifying the exact standard descriptions of financial derivative products, FpML will not only facilitate the proper processing of these products within financial services firms, but will also open up new business-to-business e-commerce opportunities for derivatives between such firms and their clients and partners.

FpML provides the following advantages:

- Specifies derivatives information in a format that is readable to both computers and humans, and can therefore be readily understood and interpreted by those familiar with the business and products.
- Enables the exchange of financial information between diverse sets of applications. This will also help to streamline the information exchange within financial services firms, especially as they seek to implement straight through processing of complex financial instruments.
- Leverages the innovation and standardization work from the Internet community, as XML is continually advanced by the W3C and increasingly embraced by the technology vendors.
- Supports the globalization of financial markets and the cross-border flow of capital, as the internet becomes an increasingly cost-effective and secure network for linking global business and financial centers.

This overview document describes the business and architectural goals of FpML and sets a context for its use including the scope, high-level business scenarios and the future direction. It also presents the architecture of FpML. In total, there are three documents that comprise the FpML specification:
(1) Overview
(2) Document Type Definitions (DTD) and components
(3) Sample FpML

2 GOALS

2.1 Business Goals

On the business side, the goals of FpML are to enable changes related to many aspects of the derivatives trading lifecycle including Pricing, Confirm Generation, Risk Management and Settlement. The confirm process for Interest Rate Market products, which records the details of the transactions that dealers engage in with their counterparties, is a key process which can benefit from electronic information exchange. The current process is manually intensive and involves considerable fax and paper-based information exchange between dealers and their counterparties. As volumes of derivatives such as Vanilla Swaps rise, the inefficiencies of the Confirm process can become intractable for the dealers who have to maintain a large headcount to support the process. An FpML-based version of the trade permits an efficient structure for confirm processing, enabling electronic validation of the details of the trade.

The increasing pressure on profit margins for the vanilla products also require better efficiencies of client service. FpML can make it easier for clients to check on the status of their transactions with dealers in a standard manner, through intelligent query facilities provided by dealers. This *client self service* model can reduce the number of low value calls that have to be handled by personnel, enabling them to perform other, more value-added operations.

2.2 Architectural goals

A key goal of the FpML architecture is to ensure that the standard is independent of any underlying transport, session or message protocol. Although FpML specifies the instrument, counterparty and trade XML components, it does not require a specific message transmission. FpML should be able to be integrated with other session level protocols (e.g. FIX) that may already have the necessary facilities for guaranteed and/or reliable financial message delivery.

Another important goal of the architecture is that FpML be an extensible standard. It will allow the addition of new financial products over time as well as support additional business services such as those outlined in the previous section. FpML enables this extensibility via its use of building blocks of financial information, from which complex financial instruments can be constructed.

The standard must specify the financial data in an intelligible manner so that people familiar with the business can recognize the FpML for the various products. While such users may not be familiar with the XML syntax, the elements and the structure of those elements should be evident. The semantics are not contained within the FpML per se; however, the language and the structure should conform to the semantics implied by the name of the product. This also allows a given financial product to be readily distinguishable from another product.

FpML is positioned to take advantage of emerging advances in XML and avoid lock-in to the current state of the art. An example of this is the use of namespaces to support modularity and reuse. This will also facilitate the anticipated future move to schemas, which are expected to provide more capabilities than DTDs in terms of expressing complex structures and validation rules. Details of the FpML conventions, some of which were adopted to provide this flexibility, are given in an appendix.

3 CONTEXT

3.1 Scope

This draft of the FpML standard focuses on the instrument definitions for a subset of Interest Rate Market Derivatives and Foreign Exchange spot and Derivative products along with the associated information that is necessary in a Trade. This additional data includes Counterparty information, and Trade Identification, as well as very limited market data for the FX products. This draft does not specify the set of business events or transactions to which the Trade will be applied. The products included in this working draft are detailed below.

3.1.1 Interest Rate Market Derivatives

Products

- Swap Product: This is a generic swap trade, which allows one or more swap streams in the trade. Each swap stream can represent a fixed, a floating leg, or a more complex kind of stream.
- Vanilla Fixed Float: This is the Vanilla Swap Trade with exactly one fixed leg and exactly one floating leg.
- Cancelable Swap Product: This represents an option on a swap that is already in existence. The option allows one or both parties to cancel the swap once it has started. It models the exercise and notification periods flexibly.
- Swaption Product: This also represents an option on a swap, but here the swap does not come into existence until the user decides to exercise the option. Again, the exercise and notification dates are modeled in a flexible manner.
- Cap/Floor Product: This represents a cap/floor stream versus a fixed stream.

Streams

- Vanilla Fixed Stream: This must have a fixed rate. It is vanilla and therefore does not include an amortizing notional, or step up coupons for instance.
- Vanilla Floating Stream: This is identical to the Vanilla Fixed Stream except that it has a floating rate.
- Swap Stream: This is a flexible stream, which allows many more features than either vanilla stream. Both vanilla streams are restrictive versions of this stream. The swap stream models the following:
  - Amortizing (accreting) notional
  - Step up coupon
  - Payment lag
  - Stream can set in arrears or not
  - Reset maturity can be different than the payment/accrual frequency
  - Calculation period dates can be different than the payment dates which can be different than the reset dates (even if frequency of all of these is the same)
  - Floating strike
  - Stubs; both implied and explicit
- Cap/Floor Stream: This stream allows us to flexibly model optionality. Each cap/floor stream represents an option based on the reset rate set against an upper/lower limit for each caplet.
- Option Stream: This represents an option on an underlier, possibly with multiple notification and exercise dates. This stream is used in the cancelable and swaption Products.

3.1.2 Foreign Exchange Products

- FX Spot: The exchange of two currency amounts at an agreed rate and date between two parties. The settlement date is normally two business days after the trade date.
- FX Outright: The exchange of two currency amounts at an agreed rate and date between two parties. The settlement date can be any date other than the market 'FX Spot' date.
- FX Swap: The exchange of two currency amounts at an agreed rate on two different dates between two parties. The direction of payments changes on the second date.
- American FX Option: American option gives the option holder the right to buy or sell a given notional during the life span of the option at a pre-determined exchange rate.
- European FX Option: European option gives the option holder the right to buy or sell a given notional at maturity at a pre-determined exchange rate.
- Knock-out FX Option: These options have two strike rates: a conventional strike against which the intrinsic value of the option is measured, and a Barrier strike. A knock-out option is automatically cancelled if the underlying spot market moves through a pre-determined level or barrier.
- Knock-in FX Option: The buyer of a knock-in option, holds a dormant option with an agreed strike and maturity that is only activated by the movement of spot through a fixed barrier strike. On activation, the option becomes a conventional European option with the agreed strike and maturity.
- Double Knock-out FX Option: There are two barriers associated with this type of option. The option becomes a European option provided that the spot rate has stayed between two barriers during the life of the option.
- Binary FX Option: An American style binary (or one touch) option is an option that pays out when a pre-defined level is touched or not during the option's life. Unlike standard options whose ultimate payout varies depending upon final spot, the payout of a binary is stated at the outset. The initial premium is paid at entry, like a standard option, and the payout is paid at expiry.
- Digital FX Option: A European style binary (or one touch) option is an option that pays out when a pre-defined level is touched or not at option maturity. Unlike standard options whose ultimate payout varies depending upon final spot, the payout of a binary is stated at the outset. The initial premium is paid at entry, like a standard option, and the payout is paid at expiry.
- Binary Knock-out Option: Binary knock-out options combine the features of a binary option along with those of a barrier option. A standard binary option payout depends on touching a pre-defined spot level prior to expiration (American), or being at or beyond a pre-defined level at expiration (Digital), regardless of the path taken. In contrast, a binary knock-out depends upon touching a certain level without having touched a different level.
- Average Spot Rate Option (ASRO): An average spot rate option cash settles at expiry the difference between a strike and an average spot rate. The averaging can take place over any defined window and with any frequency desired.
- Time Trigger FX Option (TT): It is basically a sum of Digital Options that are only different in their expiry date.

3.2 Business Scenarios

This section describes at a high level several possible business scenarios in which FpML could be utilized. Note that these are only possible scenarios; they are not exhaustive by any means, nor are they real in the sense that no implementation exists today that corresponds to any of these scenarios. They are simply intended to give an idea of how the FpML standard *might* be used.

3.2.1 Scenarios

Since v1.0 beta of the standard described in this document covers the Client View of the trade, the possible business scenarios are also described relative to the Client View, in particular focussing on the confirmation process. A possible set of services could evolve around this confirmation process, performing functionality described below, within the context of the particular scenarios.

The general scenario for confirmations is that a trade has been "done"; that is, the trade has been verbally agreed to by two parties, over the phone. The next step is to produce a tangible confirmation that is signed by both parties. Today, in the area of interest rate derivatives, this happens by describing the trade with an ISDA confirm form, possibly referencing an ISDA master agreement, and faxing documents back and forth until both parties are satisfied and sign the document. This is not a simple process due to the complexity of the transactions in addition to the complexity of the agreement itself, even for relatively simple vanilla swaps. Much human interaction in terms of checking the document (or documents), following up with the other party to ensure that the necessary changes are put in place, and faxing back and forth is necessary to carry out this confirmation process.

Following are three descriptions of possible scenarios for the confirmation process using FpML.

Scenario 1

The first scenario describes what might happen when each party produces an FpML document. In this scenario, there is a central confirmation service to which both parties submit their FpML documents and the service is responsible for matching the two and notifying each party of success or failure.

- Party XYZ and Party ABC independently produce FpML documents describing a Trade that has been agreed to over the phone.
  - ⇒ Both documents, according to the FpML specification, describe the Trade in the same way; that is, they describe it from a neutral perspective. There is no need therefore, to do any translation between the documents to describe it for XYZ versus ABC.
  - ⇒ Both parties will have made use of a tool (see below) that takes the language in which their system is written (e.g. Java, C, C++) and produces FpML.

- Both XYZ and ABC send their FpML document to a 'Confirmation Service'.
  - ⇒ There could be several scenarios, which dictate the technical details of how an FpML document gets sent to another institution. FpML has been designed to be used independently of whatever process is chosen. For example, a web server could sit in a central location and each party could send the document to the web server, using http. Alternatively, CORBA could be utilized to do the actual sending. In either case, issues such as security, guaranteeing the message gets to its intended recipient, etc. need to be worked out. Those details will not be described here as there are many issues that need to be addressed
  - ⇒ Each party electronically signs the document they send.
  - ⇒ The Confirmation Service will acknowledge that it received the documents.

- The Confirmation Service will attempt to match the two documents.
  - ⇒ There are various rules that are implied right now and that would need to be made explicit by such a service in order to carry out the matching. For instance, when there are two possible ways

of expressing the same concept within FpML, when is a match valid? This is up to the service. Some services might perform exact element matching, others might employ semantic matching. Therefore, even when two descriptions are semantically equivalent, they might not match. The following are possible areas in which this is applicable:
  - ❖ Vanilla fixed float swap expressed using the FpML for the generic Swap.
  - ❖ Notional schedule that makes use of explicit dates on the one hand versus parametrically describing them on the other.
- ⇒ How reference data is looked up needs to be resolved.
- ⇒ How the expression of parties is done needs to be resolved. Currently, the name describes the party. It is possible to have slightly different names when the same party is actually meant by both sides.

- A match occurs and the Confirmation Service notifies each party that the submitted trades match.
  - ⇒ Details of the guarantee by the Service need to be defined. One possibility is that the Service attaches its signature to the document in addition to the counterparties signatures' and sends it back to each party.
  - ⇒ What the Service sends back represents a legal document describing the trade.

- If a match does not occur, the service needs to inform each party of the place where the mismatch occurred. This ideally would be an exhaustive list, and so not stop at the first problem, but continue through the rest of the documents.
  - ⇒ Several possible scenarios could happen here in terms of only one party making changes, both parties making convergent changes, or possibly one party making changes that cause another mismatch to occur.
    - ❖ One party makes changes which now match with the original version of the other party. Since the version is part of the document that gets sent back by the Confirmation Service indicating a match, this is not a problem (again, the caveat of versioning still needs to be worked out holds).
    - ❖ Both parties make convergent changes, a match occurs, and again the Service notifies both parties of the match and indicates which versions match.
    - ❖ One possibility is that after a match occurs, one party submits a later document which does not match (e.g. the match and the additional version criss-cross in the ether). Protocol would need to be worked out with users of the service, but one possibility is that this is rejected and if the party wants to initiate another change, an edit must be submitted.

Scenario 2

The second scenario describes what might happen when both parties produce an FpML document, but there is no central confirmation service. Both parties are responsible for confirming the others' FpML and the trade is confirmed when both parties sign the others' document. Many details are the same as in the above scenario and are not repeated for scenario 2.

- Again, Party XYZ and Party ABC independently produce FpML documents describing a Trade that has been agreed to over the phone. The details are the same as scenario 1.

- Both XYZ and ABC send their FpML document to the opposite party.
  - ⇒ Again, the technical details of how an FpML document gets sent to another institution need to be decided but in practice could be different here than in the above scenario.
  - ⇒ Each party electronically signs the document they send.

- Independently, each party will make use of some service to verify that the FpML document received from their counterparty describes the same Trade as the one described by their own FpML document.

- A match occurs if both parties notify the other that this particular version of the FpML document matches.
  ⇒ This particular scenario may have slightly different implications for versioning; in any case, this is an area that needs to be addressed.
  ⇒ Each party sends back a signed version of the FpML document; there should now be two copies signed by both parties. Presumably, either or both would represent a legal document describing the trade.

- If a match does not occur, again a list is produced by each of the mismatches.
  ⇒ Again, several possible scenarios could happen here in terms of only one party making changes, both parties making convergent changes, or possibly one party making changes that cause another mismatch to occur. Since this is a two way process with no "middleman service" these will possibly converge earlier. In any case, eventually a match is reached and as in the previous step, both parties sign the others' document.

Scenario 3

The third scenario describes the case when only one party produces an FpML document, and the other party signs the document when they are satisfied that it describes the trade according to their understanding. Again, details that are the same as in the above scenarios are not repeated.

- Party XYZ and Party ABC agree to a Trade over the phone. Only Party XYZ produces an FpML document.

- XYZ sends the FpML document to ABC.
  ⇒ XYZ electronically signs the document they send.
  ⇒ This could be sent either in the form of an FpML document or, using an XSL style sheet, in a form more closely resembling the confirms used today.

- ABC either makes use of some service to verify that the FpML document received from XYZ describes the trade to their satisfaction or alternatively, they could look at an English (or other natural language) version and manually make that decision.

- A match occurs and ABC signs the document which has already been signed by XYZ.
  ⇒ Versioning in this scenario is simpler because it is a one way process.
  ⇒ ABC sends back a signed version of the FpML document to XYZ.

- If a match does not occur, the document is rejected, with a list of what doesn't match.
  ⇒ Only XYZ makes changes to the document and sends a new signed version to ABC.
  ⇒ The above is repeated until ABC signs the document.

3.2.2 Tools and Services

As FpML develops to support a wide range of business activities, it is anticipated that there will be a corresponding growth in the development of tools and services around FpML, both by vendors and financial institutions. Having one standard will hopefully encourage competitive tools and services to be implemented, allowing market participants to choose the best one. For instance, some kind of validation tool that incorporates some of the simpler syntactic rules would be very useful and would centralize some of the work that would otherwise need to be done by individual applications. This could be invoked by an application prior to doing anything else, or alternatively, prior to sending an FpML document to an application, the document could be run through this validator tool, thereby preventing sending the document in the first place if it is not valid. In sum, this kind of tool would not try to encode any complex semantic rules. It would focus on the kinds of validation that currently occur in sophisticated user interfaces.

In part, as mentioned elsewhere in this document, the need for such a tool arises due to the limitations on the amount and kinds of validation that can be done purely by a parser in conjunction with the DTDs. Even with Schemas however, this tool could conceivably be quite useful by offering validation beyond the level enabled by schemas. It would embody rules such as “effectiveDate must be before terminationDate”. It could also ensure that references were valid, and that in the places where FpML assumes that the values of two tags are equal (for instance several occurrences of date), that is indeed the case. For example, in the simple vanilla fixed float swap, the effective and termination dates in the two streams are assumed to be the same, even though they are described twice, in the context of each stream.

The above description of a validation tool implies that the rules that this tool (or competing tools developed by multiple organizations) encompasses are described somewhere in the FpML specification, or possibly in an ancillary, but still central place that everyone can access. In other words, it must be possible to determine exactly what being ‘FpML compliant’ means with regard to such rules. Currently, these rules have not been made explicit; they need to be prior to v1.0. It will no doubt be the case though that these rules, as well as the core standard itself, do not get fleshed out completely until the development of various tools and applications transpires.

On the next level, services can be developed either by vendors or within financial institutions. If developed by vendors these would require some input from the dealers that will make use of them.

3.3 Future Directions

This section describes what still needs to be done within FpML, broken down into two major sub-sections. The first presents what needs to be done before version 1.0 is complete (i.e. non-beta); that is, what is necessary to complete before FpML gets used in a real implementation. These are either issues that were not completely solved for v1.0b2 or subjects that we felt were better addressed by working groups, which could draw on expertise from other market participants as well as vendors and service providers. The second describes future work, which is basically the direction in which we see FpML heading. The latter will be addressed in a manner which is consistent with the priorities as defined by the steering committee.

3.3.1 What is missing in v1.0b2

1. Business Event Transactions
   A working group from the technical committee will be assigned to look at this. These would include events such as a new trade, an exercise event, or an unwind. Because of events such as edits, this will encompass related issues such as history (what is necessary to keep around from one version of a trade to the next), identifying changes, trade versioning, etc.

2. Validation
   Ideally as much validation as possible should be done upfront, that is, within the DTDs (for the first release, schemas when we switch to them). However, DTDs can only handle a limited amount of validation and there will therefore need to be some at the application level. This could either happen within each application or possibly within a tool, which is invoked by applications. The rules that the applications or tools must follow to be "FpML compliant" need to be defined somewhere.

3. Matching
   In various places within FpML, there are different representations for equivalent structures, both at a high level (e.g. vanilla fixed float vs. general swap) as well as the lower level (e.g. different date structures). Where these occur, rules need to be defined which dictate how translation between them should be done. Alternatively, in certain cases, it may be that the rule states that two structures only match if they are identical, not equivalent.

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

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

6. Framework for interacting with other standards
   FpML will concentrate on defining the instrument and product definitions for Derivatives. Other standards exist that address other products, such as FIX for equities. A framework needs to be defined that will allow interaction with other standards, utilizing namespaces and other concepts within FpML.

7. Fees
   These are currently missing from v1.0b and need to be added.

8. Compounding
   Again, this needs to be added.

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

3.3.2 Future Work

1. Product Coverage
   The products that are covered by FpML will be expanded to include other derivative products. The scope will include additional Interest Rate and FX Derivatives, as well as other areas such as Credit Derivatives and possibly Equity Derivatives. Our focus is strongly on Derivatives, and does not include plans for other markets such as equities.

2. Views
   The first release of FpML is a description of the 'Client View'. Other Views will be added in the future, such as the 'Risk Management' view. Also, work to define Views explicitly within FpML needs to be done. The related idea of being able to add proprietary data needs to be addressed as well.

3. Market Data
   Market Data is only touched on for the first release, specifically in the area of implied volatility for some of the FX products. The FX instruments also require data in an area that sits between an instrument definition and market data proper. That subset is defined in FpML to be outside the product but within the trade. However, the area of market data in general is a very large issue that is not fully represented in v1.0b2 by any means and will be addressed in the future.

4. Technical work
   The W3C efforts on XML Schemas are expected to have a considerable influence on the evolution of FpML. A standard schema-based approach for capturing the structure and allowed values will be of considerable benefit to FpML. With schema support, it will become easier for technology vendors to provide off-the-shelf tools for validating FpML.

   Digital signatures will be important for authenticating XML documents as part of e-commerce and business-to-business commerce dealing. FpML will take advantage of this capability once the signature work has become standard.

   The FpML Standards group will work closely with other standards groups to develop full session-level capabilities on which FpML can be exchanged.

While the first version of this standard focuses on Interest Rate Market and FX Derivatives, going forward, FpML will support other classes of derivative products such as Credit Derivatives and Equity Derivatives.

4 ARCHITECTURE

This chapter describes the architecture behind FpML and it is divided into three sections. The first section describes the key factors driving the design of FpML and the resulting design principles. The second section gives a brief overview of the conceptual framework. The third section describes in detail the concrete approach that was taken.

4.1 Driving Factors

From an architectural point of view, the determining factors driving the design of FpML are three-fold: the quest for standardization in the financial derivatives domain, the need to de-couple systems in order to leverage a common service infrastructure, and the desire to exploit business opportunities enabled through Internet based new technologies.

Up to now most communication between financial parties in the derivatives domain has been done via phone and fax. This has made the further processing of trades and trade related information into the information systems of each party a manual process of data entry and data comparison. FpML must facilitate that process in electronic form.

If financial institutions have straight-through processing systems in place they typically consist of a number of different applications that exchange data in various ways, including flat files, shared databases, publish/subscribe mechanisms, RPC, and CORBA. FpML should provide a language and system independent way to express the relevant financial data de-coupled from the particular processing required.

The Internet has already had a big impact on the financial retail market, reducing the transactional friction between consumers and financial institutions and enabling new kinds of services that were either non-existent beforehand or very expensive. It is clear that the Internet will have a similar impact on the wholesale financial market, and FpML is the vehicle in the derivatives domain to open those opportunities. This requires that FpML fits well in the realm of existing and new internet technologies.

4.1.1 High level Design Goals

The design goals behind FpML can be categorized into four areas: financial, structural, processing, and evolution. The following lists the goals within each section.

Financial

- Capture the interest rate and foreign exchange derivatives domain
- Support front, mid, and back office activities
- Be useful between parties and within one institution
- Use business terminology

Structural

- Reuse common components
- Create coarse, orthogonal, components
- Embed and be Embedded
- Divide trade element into orthogonal components
- Allow proprietary data for internal usage
- Dynamic trade related data

Processing

- Be transport independent
- Be programming language independent
- Enable web and internal processing
- Enable data validation
- Minimize leap of faith for adopting existing systems
- De-couple data from API's

Evolution

- Plan for evolution of Internet standards
- Use consistent conventions
- Plan for evolution of financial domain
- Push standard out early and revise with others

4.1.2 Design Tradeoffs

In order to focus the development process surrounding FpML and in order to resolve some conflicting design goals design tradeoffs were made. The main ones are listed and explained in this section. Additional background information on such tradeoffs can be found in section 1.

Validation

There are various ways to validate XML documents. FpML uses validation facilities of DTDs as much as possible, with a view that more sophisticated features will be available once schemata languages are standardized. There will always be degrees of validation that can only be done on an application level. For FpML that currently is true for referential integrity and combinations of dependent element values (e.g. start date less than end date).

One possibility could have been not to use DTDs at all and jump directly on one of the current schemata proposals, but since these are not stable enough yet, this was rejected. Instead, a tool, which can later convert to a schema language easily was utilized. Total proprietary application level validation was rejected because it is too far from the main-stream of technology right now and would present acceptance, evolution and deployment problems.

Signatures could be combined with validation, so that coarse elements get signed after they have been semantically validated. Future users of those elements only need to check the signature. This is an interesting option and might still be utilized.

Element composition

FpML elements are decomposed into two kinds: small components that are reused everywhere and coarser components that describe more complex financial concepts. Processing services are written in terms of those coarse components.

The approach of using one big component per product or trade was rejected. Such a component would capture all relevant information in one unit. This was decided against because it does not allow for thinning down the data for processing services that do not need all the information.
The idea that coarse components can be enriched over time was also rejected because the idea of signing components was appealing (and that is not sensible if they change over time). In addition, making the processing as "type-safe" as possible was seen as advantageous.

Element structure

A component-based element structure instead of a flat parameter structure is used in FpML. Experience with existing systems has shown that flat parameter structures can yield to incomprehensible data for complex instruments. Although some of the conciseness of a flat approach for simple instruments was sacrificed, the extensibility the component based approach offers for complex instruments outweighs this.

Instrument representation

FpML uses strongly typed components that have well defined plug-and-play behavior.

4.2 Conceptual Framework

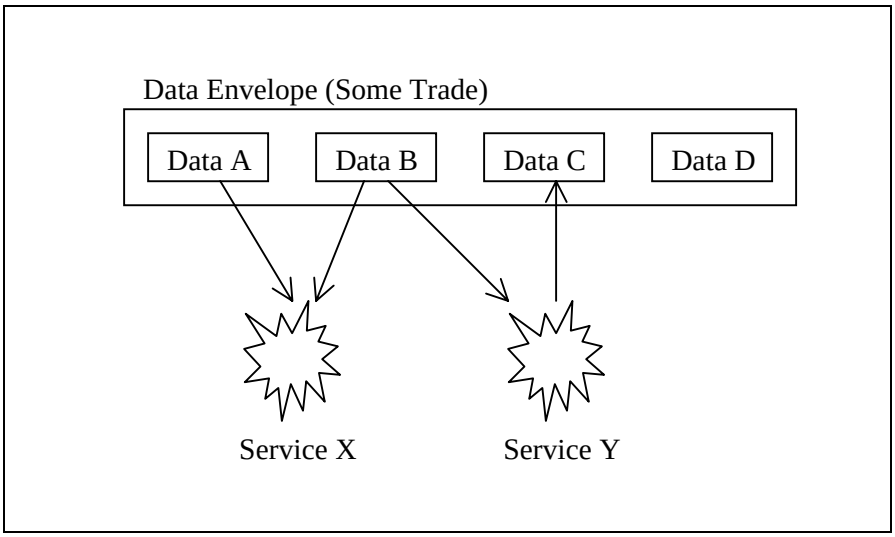
FpML is based on a framework that enables the goals mentioned above. In generic terms, the main idea is the use of coarse data components and data envelopes to represent descriptions of relevant data.

As described above, in the domains of derivatives and foreign exchange, this includes data associated with the ownership / exchange of a product. For instance the counterparties, settlement instructions, fees, and terms, as well as other data that is required for particular kinds of processing (e.g. market data and pricing parameters). Data elements defined in the standard are generally applicable and not specific to the usage and requirements of particular organizations.

Separating the data into coarse data components means that instead of creating a flat data representation of all the data available in the derivatives domain, the data is factored into different components. Each of those components serves a particular semantic aspect. These components are meant to be as orthogonal as possible. Some of the components might be immutable once they exist, and they can be signed separately. Examples for such components are counterparty information, trade identification, product, fees, and settlement information.

Data envelopes are a means of embedding the coarse data components into a structure that denotes a whole. The data components inside an envelope might have to refer to the content of other data components within the same envelope. For example, the product description for a trade refers to elements within the party information for that trade.

Data envelopes can be seen as configured for a particular processing service. Such configuration is called a view. A view identifies the minimal set of coarse data elements that need to be present in a data envelope. An example of this is the trade element. Not all kind of processing requires all the data possible in a trade element. For instance, a confirm service might only require tradeID, product and party information. Yet, the standard ideally describes all coarse data elements that could be inside a trade. The processing service will work in conjunction with a particular configuration. That is, it will require a given set of data elements in order for the processing to be performed (it might also describe which new elements are produced, if any).



The figure exemplifies this concept. The data envelope describing some trade contains four data components. Two services process the same trade, but make use of difference data components.

FpML explicitly does NOT state how the processing of those data elements is to be done. This applies to processing like pricing, matching, or confirm generation.

The initial version of FpML will cover only a small slice of the possible data elements one could define: the data required in the client view which represents the commercial terms of the trade both parties agree to. A future version of FpML might describe data related to other kinds of processing of derivatives, e.g. risk management, settlement, portfolio management, market data.

Any standard evolving in the financial industry has to manage the tension between making data and services available to the public or selected clients or making other data and services remain proprietary in order to preserve competitive advantage. The extensibility permitted through the data and processing framework of FpML, helps manage this tension. The data elements defined in the standard can be exchanged among organizations or used within systems of one organization and are rich enough to serve a diverse set of requirements. At the same time, using the concept of the trade as an envelope of information, there is enough flexibility provided in FpML to allow organizations to use and associate additional proprietary information with trade information.

The same flexibility holds for the processing. Organizations can use standard FpML data elements within their proprietary software, potentially enriching it by additional proprietary data elements. They can offer services with FpML while keeping their processing proprietary. At the same time the standard data elements will be rich enough to create a market for software component developers.

4.3 Approach

This section describes in more detail the approach we have taken in defining the FpML standard.

4.3.1 Overview of Approach

FpML has taken the approach of breaking various pieces down in order to concentrate on one area of concern at a time. This has been done along two dimensions: the reduction of the whole into various objects and the cohesive grouping of those objects. This idea of what concrete structure is optimal,

including the issue of granularity, is an important one within FpML. Central to the approach is the idea of a 'View'. This encompasses two primary thoughts: firstly, that explicitly describing an entity with respect to a particular business purpose is important, and secondly that it is not simply the entities themselves that need to be considered, but also the relationships that they share. Finally, as is evident from the language used, concepts from the Object Oriented (OO) world have influenced the approach taken in FpML.

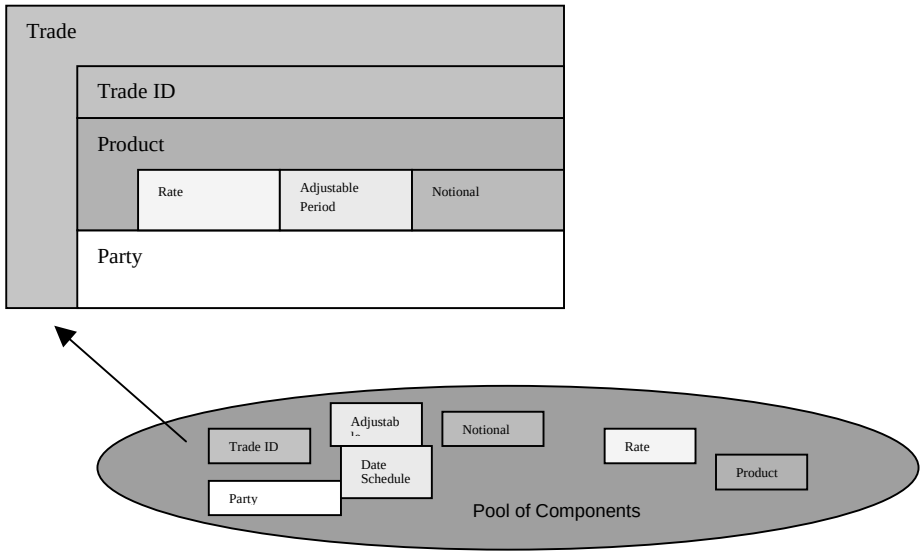
For the first release, FpML focuses on the instrument definitions for a subset of Interest Rate Derivatives and Foreign Exchange products and Derivatives. This encompasses a wide range of information. Structuring that information in a way that facilitates the FpML standard being used by applications to perform the processing that they need on those instruments was an integral part of the approach. It should therefore be clear what business process, or set of processes, are enabled from a given FpML document. Just as importantly, it should be apparent that the FpML definitions are correct and complete (or conversely, highlight any gaps), in order to ensure that the FpML standard is accepted by the community. The information that is represented is complex, and it is necessary to associate some amount of structure with the data in order to accomplish this.

Therefore, incorporated into FpML is a significant level of inherent structure that is not necessarily what one might expect from a 'data' representation. It is useful to consider concepts from the object world, in order to represent the data in a way that conveys more information than the sum of its parts. The OO community has successfully illustrated that objects play a useful role in this arena, and various concepts in FpML, from the naming conventions, to the structural representations reflect this. The level of composition shapes the way one thinks about these entities and will facilitate not only the acceptance of the FpML standard but also its use going forward. In the next section, descriptions of the integral structures that are incorporated into FpML, as well as more detail about Views, are given.

4.3.2 Structure

Trade

Trade is the top level object just under FpML, and is the entity that holds the various pieces of information, as this seemed an intuitive level to talk about information being exchanged between financial institutions. In sum, a Trade is an agreement between two or more parties to exchange some financial instrument(s) and to convey whatever information is necessary in order to carry out that exchange. Focussing on particular aspects of a Trade at one time is beneficial, as it is a broad category and could include many different kinds of things. Furthermore, explicitly differentiating between the various characterizations that a Trade has, depending on the business purpose for which it is being used, is advantageous. Finally, capturing all the different parts of a Trade is not sufficient; it is also necessary to convey something about the relationship that the constituent parts have. These lead to two important features of the FpML Trade definitions: the association of a Trade with a View, and the specific structure of a Trade.

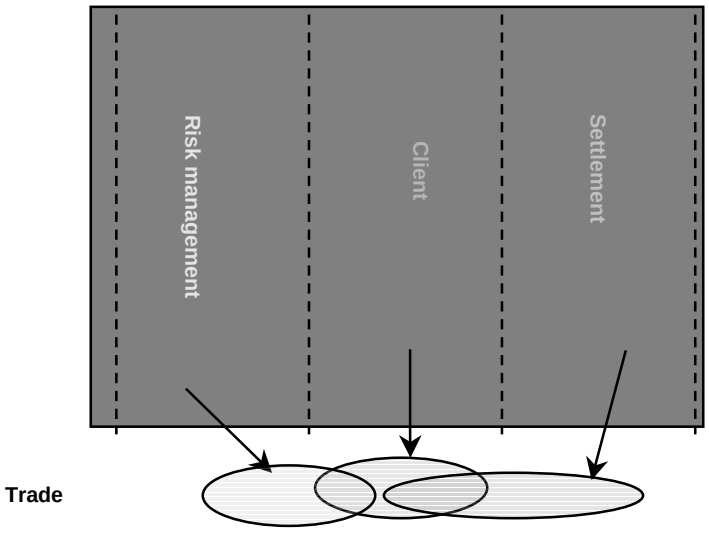


View

In order to incorporate the ideas above, a Trade is associated with the concept of a View, which encompasses the idea of a Trade existing for a specific business purpose or activity (or set of activities). The view imparts a particular context and therefore also implies something about the relationship between the Trade's constituent parts. Note that View is a *conceptual* entity; it is not explicitly modeled within FpML. For release 1.0, the Trade definitions for one view only are included, namely, the Client View. However, it may make sense as additional Views are incorporated, to explicitly include this concept as part of the FpML definition.

Trades can have multiple Views altogether, each of which captures a specific intention and context for the Trade. Within a given FpML document however, each Trade is associated with one and only one View. In the first release of FpML, the Client View of a Trade is represented. The Client View represents the information about a Trade exchanged between two financial institutions for a specific business purpose. One particular business process that this View could be used for is Confirmation, although it is not limited to that activity. While version 1.0 of FpML does not include other views, such as a Risk Management View or a Pricing View, the approach is extensible by allowing additional information to be added (through enrichment) as well as possible transformations of the information to take place. This will be explained in more detail below after the other primary structural components are described.

An area that still needs work is that of business transactions. It was consciously decided not to represent the business transaction as part of the first working draft release. This is a complex subject in its own right and seems a probable subject for a working group of the technical committee. It is relevant to mention here though because of the View concept, which specifically links a Trade with a business activity or set of activities. For example, the initial View of a Trade is the Client View. One business activity that the Client View enables is the Confirmation of a Trade. This involves exchanging FpML between parties, and it is therefore necessary to include some transactional information in order to fully define the confirm.[1] This information is missing from the initial FpML definition. It is possible that this area of the business transaction is where it makes sense to make explicit the concept of View within FpML.



Core Trade Component

The constituent parts of a Trade are the next level of structure in terms of level of granularity; these are referred to as Core Trade Components. Although the specifics of exactly what core Trade Components will be included in a given Trade depend on the Trade View, the Core Trade Components are entities that make sense in their own right, and are therefore rather coarse grained. It is important to note though, that

[1] Note that what is meant here is different than the message protocol level information, which we are not defining, nor do we intend to define as part of the FpML standard. We expect that various protocols will be used to transport FpML and there is nothing within FpML that commits us to a specific one.

an FpML document must contain a Trade which in turn will contain the core components; FpML documents for Core Trade Components do not exist on their own. This does not mean that these components will not be able to be reused across different contexts; this is in fact precisely what is intended.

Because it implies a given business process or activities, the concept of a View is integral to determining what core components belong to a trade and what their relationship is to one another. For the Client View, the following Core Trade Components exist:

- Trade Identification: This component associates a transaction reference number for the trade for each party. It is not necessary for each party to have the same reference number.
- Product: This component is the heart of the instrument definition and is explained in more detail below.
- Party Organization: This component models the Counterparty information plus all the associated information that conveys the hierarchical organization that financial and other institutions might have.

These core components can reference each other as a whole or they can reference internal components within another core component. For example, both the Trade identification and the Product components reference the Party component.

Product

This is the area where the differences between the Interest Rate Derivatives and Foreign Exchange instruments appear, and is the area that represents the individual instruments within each domain. It is therefore here that many of the complexities of the different instruments arise. This component captures the economic details of the trade. Because of the complexity of the domain of instruments that FpML addresses, composing these products from various building blocks is a key aspect of the approach.

Interest Rate Derivatives

There are several layers of building blocks within Product, the highest of which is the Stream for Interest Rate Derivatives. This component represents the details that are required to calculate a set of cashflows. Underneath the stream level are various other components, which will be described more in the Components section. It is important to note though, that while a Stream can be taken wholesale and reused somewhere else, the other components cannot be randomly combined.

The product represents the combination of Streams that exist for a given Trade. For some products there is a specific combination of streams that must exist, and for others, any collection of streams is valid. This difference in products has been incorporated in order to provide flexibility for the places where complexity is required, yet to attempt to restrict the possibilities where the extra complexity is not required. Therefore, FpML has a Vanilla Fixed Float Swap and a more generic Swap. The Vanilla Fixed Float has exactly one Fixed stream and exactly one Floating stream. The generic Swap allows any combination of Streams. This difference is important because the swap can get extremely complicated, allowing various combinations of amortization schedules for the notional, payment lags, stubs with their own way of calculating the rate, and many other features. The Vanilla Fixed Float on the other hand is very restrictive. Partly because DTDs are limited in the amount of validation they allow, and partly because of some of the design decisions, there is only a certain amount of validation that will be performed at the DTD level. So, it is quite possible to have a valid FpML document that satisfies the DTDs, but yet does not make sense from a financial point of view. It may be that the application cannot in fact handle it, and will have to reject it. A fairly large amount of validation and checking will need to be done in the applications that are using FpML (and perhaps in an FpML parser). The idea of a more

restrictive Vanilla Swap facilitates more upfront checking, where many more requirements in terms of the specific structures and their relationships can be specified. It also has the benefit that not everyone will be required to have an application which can handle the more complex swaps, but will still be able to derive benefit from FpML. Clearly though, the Vanilla swap is not sufficient to handle all the complexities that are possible to specify as part of the definition of the swap. Therefore, FpML has a more generic one as well.

The generic swap can support many features, such as an amortizing or accreting notional, payment lags, and initial and final stubs that have their own rate associated with them. There is a distinction between functionality that can be handled by an additional 'feature' within the swap and things that require an additional kind of Stream or Product. Moreover, it is not always clear exactly on which side of the line something falls. For instance, compounding could introduce a new stream, or it could be incorporated into the generic swap. The latter seems to make the most sense. Averaging, on the other hand, makes more sense to represent with a new Stream. The important point here is that depending on exactly what it is, one could choose to handle it either way.[2]

There are some Products, such as the Cancelable and the Swaption that reuse existing Products. Each of these is an option on an underlying swap. The swap in both cases is the generic Swap Product, which therefore can contain any combination of streams. It is possible therefore to have an option on any kind of Swap.

Foreign Exchange Products

In this first release, the Foreign Exchange (FX) product category covers Foreign Exchange trades (FX Spot, FX Outright and FX Swap) and Options on Foreign Exchange (FXO) trades (European Option, American Option, Binary Option, Barrier Options, Time Trigger Option, ASRO (Asian) Option.). As many of the product types contain similar field groupings, component groupings were defined and are used across different products.

For Foreign Exchange trades the first component below Product is the product type itself (such as FXSwap), each of which implies certain components necessary to define the product. Although components are shared across many of the products it is not possible to combine them ad-hoc to create a new product because although the data may be valid within each component, it may not be valid across components or the component grouping itself may not make sense.

The next level of component is FXLegTemplate. This component is common to all FX Products; one template is used for FX Spot and FX Outrights, two are required for FXSwaps. This component contains low level elements and other common components such as Amount and Rate. DTD validation will check the data within each component; however, application logic will be needed to check data across the components used by the FXSwap product.

Within FXO products, high level product groupings were defined (such as FXVanillaOption, FXBarrierOption, and FXFixingOption) to group different products that shared the same basic parameter definition and hence components. FXVanillaOption uses the FXOptionCore component to group the parameters needed to define a basic American or European option. FXBarrierOption is used to define any type of barrier option that has a standard option (American or European) as its pay out (such as Knock-In, Knock-Out, Double Knock–Out, etc.). It contains the FXOptionCore and a container class containing any number FXOptionBarrierCore components. FXFixingOption contains FXOptionCore and

[2] We have not yet done either averaging or compounding and expect this to be an area of discussion going forward.

FXOptionFixings as main components. FXOptionFixings contains fixing reference information together with either FXOptionFixingAARef for ASRO (Asian) style options or FXOptionFixingTTRef for Time Trigger (TT) style options. Although ASROs and TT options are completely different products the parameters used to define them are similar and hence can be grouped together in one component. The OptionPremium component is common to all options, defining the parameters used to price the trade.

Forward Rate Agreements (FRA) Products

Since the parameters needed to define FRA products are based on a set of elements common to all FRA trade types, the ForwardRateAgreementTemplate component is used for all FRA products. This template component contains the FRACore component and optionally the AverageFRA component, which is only used in defining Average FRAs. Within the FRACore component, the floating rate is defined within the floatingRates collection. Normally only one floating rate is needed; however, two floating rates are defined for interpolated FRAs.

Component

At the next level in the structural hierarchy are general Components. These in fact can be arbitrarily nested, down to the point of primitive types. Components cannot randomly be combined and cannot be thought of as existing in their own right; they only make sense in a given context and in relationship to other components.

One set of components that figures prominently in FpML is the set in the Date namespace (see Modularization section for details on namespaces). Several core ways of expressing Dates and Periods are defined. These are divided into expressing the concept of a single Date or Period, and multiples of each. The latter are indicated by attaching the suffix 'Schedule'. Moreover, absolute dates and dates that are adjusted according to some business convention and set of business days are distinguished. Finally, dates can be related to Periods by extracting a set of dates in a specified manner.

Other components that are defined include Rates, which encompass Fixed, Floating, and OptionReset (i.e. a reset on which another component is contingent) rates.

Market Data

Market environment is an important high-level component in dealing with financial derivative products. It is on the same level of granularity as Trade, and can be broadly classified into the following:

- Observable market indications for actively-traded financial instruments (i.e. treasury yields)
- Data derived from observable market indications (i.e. implied volatility)
- Financial instrument's reference data (i.e. payment frequency)
- Pricing parameters

One cannot model pricing parameters without modeling other pieces of market data. However, it is more interesting to model data derived from observable market indication than other data (say reference data) as the data formats for them are relatively less standardized.

The first version of FpML defines only implied volatility for FX options. Future versions of FpML would be expanded to other market data relevant to FX and IRM products.

Implied Volatility

It is envisioned that market data interaction would be based on request (query) / reply (result) paradigm. User of the data would initiate a market data query and a market data processing engine would then satisfy the query and send only the relevant piece of market data.

Currently, only the result set for implied volatility is modeled, and is organized per currency pair. With in each currency pair, data is grouped based on term (or tenor). For each term, the volatility smile is shown for various delta and put/call combination. The delta neutral (also known as 'at-the-money volatility') is also shown.

4.3.3 Extensibility

This approach extends nicely. Various levels of structures (e.g. different levels of granularity) have been defined in order to achieve reuse at these different levels. Together, they outline a way in which FpML can be extended to handle additional products and business functions. In order to extend an existing Trade (which implies that one is extending an existing View of that Trade), one could add more components if it is discovered that something is missing or a new feature is required. An example of this is when core business components are added to support business transactions.

To add a new view of a Trade, one would enrich an existing view. This can be done by adding new Core Components. Alternatively, existing information could be transposed to a new representation in some predefined way through the use for example of XSL. This handles the case for instance of the risk management view needing essentially the same information, but organized in a different structure.

FpML has been defined in such a way that the extensions and translations that might be necessary in order to achieve other views have been considered upfront. The representations of existing products and views have been structured so they lend themselves to translation or enrichment.

1 APPENDIX: FREQUENTLY ASKED QUESTIONS

1.1 Introduction

This chapter presents some frequently asked questions (FAQ) on FpML and current answers. Some of the questions point to issues that are still outstanding, others provide an explanation for design decisions taken.

1.2 Focus

1.2.1 Why are there no messages defined?

The focus of FpML is on the basic data objects that particular kinds of processing require. FpML does not specify how the processing is done, nor does it talk about how the data gets to the processing services.

In some areas message protocols for describing transport and transactions between systems/users exist already and it is the hope that FpML descriptions will eventually be embedded in those. In other areas, especially for Internet based applications and XML based application services, such standards are much less developed. By not insisting on a particular protocol the standard enables and encourages more experimentation and a larger potential usage of FpML.

Future versions of FpML might standardize aspects of processing once a shared view in the user community develops on which ones those are.

1.2.2 Why Client View?

It is easier to define a standard for areas that many users agree on already. The Client View is something that is not proprietary to a particular financial institution, it is exchanged in confirms and codified through the ISDA agreement. Thus one hopes that there is less reason for conflict and a faster agreement on a standard, than on other areas, e.g. the financial event structure, pricing results.

1.2.3 Is there a difference between Business Transactions and Protocol?

In the context of this document, protocol means how bits are moved from one system to another (HTTP, CORBA, TCP/IP), and business transaction means the semantics and procedure related to the business related interchange between parties (sell, unwind, buy). FpML does not standardize on a protocol. Currently FpML does not include any standardization for business transactions, but it might in the future.

1.2.4 What is the relationship to other standards?

FpML is a standard driven by the financial industries with support from vendors. Although it does compete conceptually with some vendor driven standards there is a strong desire to come to consensus on a standard that is most beneficial to the financial industry. Some other standards, finance industry and vendor driven ones, are more complementary, in that they focus on other kind of financial instruments, or in that they focus on the business transactions between parties.

1.3 Conventions

1.3.1 Why are some element names in lower case and some in upper case?

An object oriented perspective benefits from a clear differentiation between classes and the state captured by a class. XML, coming originally from the world of document markup, does not facilitate such a view

natively. FpML therefore uses naming conventions to make those distinctions clear. An upper case element name always represents the name of a class. Lower case names always represent an instance variable/fields.

1.3.2 Why CamelCase?

The camel case is a naming convention that originated with Smalltalk and was later used within Java, JavaScript and VisualBasic. It has therefore quite some market share in the realm of naming conventions.

1.3.3 Why are collection elements wrapped in their own element?

The aim is to be as clear as possible about repeating elements, and the easiest way to accomplish this is to wrap them in their own structure and not mix them up with elements that only have a single occurrence. The same mechanism is used in many programming languages that allow the definition of data records/types. Most will use an extra instance variable to point to a collection of objects (one of the exceptions to this is LISP where the basic underlying structure is a list)

1.3.4 How are ID and IDREF used?

FpML currently does not use ID and IDREF. The main reason is that this would require names that are unique for one document, and since there might be multiple trades in one document, those unique names are not easy to come by. FpML favors a trade relative naming approach and is aiming to use Xpath once it becomes stable. There is also a proposal on the table on how to redo naming and references, which is a little more restrictive but more structured and would also eliminate the redundant data in documents right now.

1.3.5 How are attributes used?

Currently attributes are only used for two purposes: (1) to represent typing meta-information, and (2) to express naming. Even though initial versions of FpML made use of attributes for enumerated values that was dropped in favor of a consistent element approach.

1.3.6 Why are there so many files for defining the DTD's?

One of the goals is to provide a clear structure that allows definitions to be mostly independent of each other. This is important for development, evolution, and maintenance reasons. DTDs' might be sorted into subdirectories later.

1.3.7 Where are the allowed combinations and values of elements documented?

This still needs to be done; it should be part of the element description for the first public release.

1.4 Structure

1.4.1 Why are the trade related data components so coarse?

One goal in the structuring of FpML has been to be able to factor the data components in suitable ways for processing. In particular, it is a goal that a particular processing service can pick and choose the minimal set of trade data components it requires for a task.

1.4.2 Why not have flat data structures?

Within FpML data is grouped together in a fashion that can be thought of as a class / instantiation of a class. It also makes the relationship between data explicit where appropriate. Arbitrary nesting allows capturing different levels of granularity regarding grouping entities together.

This approach generally results from an object background. XML is typically not used to serialize object structures, but the benefits of doing that surpass the disadvantages of an unusual XML style.

1.4.3 Why isn't there a notion of a deal?

In contrast to a portfolio, a deal expresses a number of trades that are entered into in one business transaction. For now FpML avoids this issue, because it is close to the subject of business events. It should be revisited for the first public release.

1.4.4 Why isn't there a notion that products can be stand-alone?

It is clear that such a feature is desirable, but it raises the problem that one does want to have particular values of such a product filled in. E.g. if one has an auction like service or an electronic offering of products one probably wants to express that one can offer 5 year USD LIBOR swaps but leave particular other values up to the buyer to define. FpML is currently missing a mechanism to express incomplete products. A new reference proposal might help here. The issue needs to be revisited before the first public release.

1.4.5 What is a view, and can one define an arbitrary one?

A view is a particular, required, collection of the information components contained in a trade. Typically a view is processing service specific in that it is the subset of elements that must be in the Trade element for a service to function properly. For example, one might have a matching service that needs the trade ids, product, parties and fee elements in order to do a match, whereas a pricing service where clients can store their portfolios might only need the trade ids. Currently there is no mechanism to express a view. This issue should be revisited in the context of business events and processing before the first public release.

1.4.6 How are multiple products in one trade modeled?

Such a facility does not exist right now, but a simple mechanism would be to create a CompositeProduct that can include a list of products. This issue should be revisited before the first public release.

1.4.7 What is the difference between portfolio and generic transaction grouping?

The grouping and nesting of trades in portfolios, and the grouping and nesting of portfolios themselves is done to structure one's world of trades in appropriate ways for the purpose of communication and processing. A trade might actually be in multiple such portfolios. When trades are grouped within a business transaction it typically means the business transaction will fail if it fails for one of contained trades. FpML currently does not model business events and a transactional grouping of trades should be considered when business events are revisited.

1.4.8 Is there one and only one document for a trade?

No. There can be many documents that contain a particular trade. Furthermore, there might be documents that do not contain all the available information for a trade. (Note, in FpML a trade is seen as all available information associated with the ownership of a product). Sometimes the information that makes up a trade is generated on the fly and sometimes the different information components might come from multiple databases. This question relates to the view question above.

1.4.9 How can I add my proprietary data?

If the data you want to add is related to the ownership of a product, invent a new information component and make it a sub-element of the Trade element. If it is independent of a product, invent a new

information component and make it a sub-element of the FpML element. Use namespaces to avoid name clashes. Do not add new elements or attributes to other standard components.

1.4.10 How can I use my proprietary data?

For the particular processing service, define which data components are required (standard ones and proprietary ones). As mentioned above, this subset of components is called a view (and currently FpML does not have a formal mechanism to express this concept). When receiving FpML in the processing service, validate that it conforms to that view. However, in order to converse with others in standard FpML, you must construct views only from components that are defined in the standard.

1.4.11 Why are namespaces used?

Without namespaces all sub-elements end up in one global namespace. Coming from the object oriented world the analogy is that all instance variable names are in one global namespace instead of being local to the class hierarchy of the class they are defined in. Any time somebody in a development team makes a local change, there is a potential with conflict with existing elements or concurrent local changes in another area by another user. That is a current flaw with XML and DTDs and namespaces are used to avoid this problem.

1.4.12 Aren't elements declared many times?

Shared components are declared in their own namespaces, and the object-oriented approach allows defining them only once, but making use of them in different contexts. For instance variables that have the same name and are defined in multiple objects it might feel like a redefinition. Some of that is due to the fact that they really are different, some of that is due to the fact that current DTDs' cannot express inheritance, and thus require the redefinition of elements, which would have been factored into common super classes in an object-oriented data model.

1.4.13 Why Prefixing?

Prefixing achieves the advantage of namespaces with DTDs even though the existing DTD validation mechanisms do not support namespaces. Once FpML moves to schemata the need for prefixing every element goes away and one can use scoped namespace declarations instead. Documents that have been created under the first, DTD based version continue to be valid documents when FpML moves to schemata.

1.4.14 How are references from one data element to another done?

Currently this is done by name. Specifically, the element that is being referenced will have a name as an attribute (e.g. name="bar"). This will then be the value the referencing element uses. In general, if the value of an element foo can be expressed by a reference, there is an alternative element called fooReference, which contains the reference name. This issue needs to be revisited for the first public release. XPath, a shared proposal by the XSL and XPointer group of W3C is getting more stable and could be used instead.

1.4.15 Why is Money an object in itself?

It is an object because it is an interesting semantic unit in the financial domain. In that sense it is similar to Floating Rate and Fixed Rate. FpML opts for the structured approach and reifies those kind of objects instead of folding them flattened into the object that contains them.

1.5 Processing

1.5.1 Where is validation done and how much is necessary?

There is a scale of choices with respect to the validation of XML documents. On the rigid spectrum one uses all possible features of a DTD to do validation, on the other side, one gives up on DTDs because they are so weak anyway, and does all validation in the application. FpML takes a middle ground. As much validation with DTD's is done as possible without having that force unnecessary burden on the expressibility or consistency. Schemata languages will pick up some of the remaining validation issues, (e.g. to check types and enumerations, which FpML does not represent as attributes), but there is a requirement for doing validation in the application or a validation services (which could then sign the information component, so no repeat validation would be necessary). The main areas of validation that fall in that category are referential integrity of named and referred elements and the instrument specific appropriate values of elements. The latter needs to be described with the element description before the first public release.

1.5.2 How are signatures done?

This topic is a candidate for a working group in the technical committee. Currently FpML assumes signing and the storage of signatures in an information component is independent of the components that need to be signed.

1.5.3 Where are the business events?

This release does not define any business events. It does make some suggestions in the business scenario section as to what business events might be interesting ones for a particular domain, but the standard itself does not define them. It is assumed that their definition would make use of elements currently defined in this standard as part of their data and that processing related working groups in the technical committee will pick up this topic.

1.5.4 Should there be versions of a trade?

This issue is interesting and important for certain kind of processing but has been deferred to working groups in the technical committee.

1.5.5 How does FpML provide access to the history of the data?

This issue is interesting and important for certain kind of processing but has been deferred to working groups in the technical committee.

1.5.6 How does matching work if there are different ways of expressing the same concept?

This question is applicable for the high level matching (e.g. vanilla fixed float vs. general swap) as well as the lower level (e.g. different date structures). One possibility is to explicitly state which choice is being used and if there is not agreement at that level, then there is no match. Another possibility is to have a transformation from one to another. When translating from the simpler version to the more complex (e.g. vanilla to more general), the transformation should be able to be completely specified. If the transformation goes the other way, obviously it might not work (that is a non-vanilla swap might be the input which would not in fact be able to be transformed). Matching also presents a problem when some kind of algorithm is involved (e.g. dates and frequency). All parties must agree on the algorithm. If there are discrepancies, this would be problematic.
The standard must define the rules to generate a normalized form that can be compared.

1.5.7 Can an FpML document serve as the prime record?

There are different opinions about this right now. In the end, this issue will be driven by business needs/opportunities and the consensus of the technical committee.

1.5.8 Why isn't the document expressed from one party's point of view?

FpML uses a symmetrical document because that implies that the same document can be signed from either party's point of view. This means there is no 'owner' concept for the document, although there may be from a trade perspective.

1.5.9 How does FpML handle reference data?

There is a difference to how internal and external reference data should be handled: for external data it is reasonable to provide some kind of reference. For internal data that not all parties will see, it is not, except in proprietary data components.
Reference data that can be expressed with an international standard will be expressed in a notation, e.g. currencies, ISDA-rate option names, business centers. Dates are expressed in the ISO standard.

1.5.10 How are defaults handled?

"Defaults are usually wrong". FpML does not have any defaults in the standard. Applications can provide user interface defaults when constructing FpML.

1.6 Interest Rate Derivatives

1.6.1 Why are there rigid instruments and flexible ones?

The Vanilla fixed float will cover a significant number of trades while still providing a simpler structure. People will be able to understand the simpler structure more easily (e.g. only 10 things to look at instead of 100). Systems will be able to be simpler if some clients only want to deal with the vanilla case.

1.6.2 Why aren't there low-level components that can be plugged together for arbitrarily complex products?

This is a complex enough topic that it probably warrants a white paper. From a business point of view it scales better and makes processing for simple products easier, while still allowing the definition of more complex products.

1.6.3 Is there an alternative to parametric information for date schedules?

There are provisions in the standard for replacing some parametric information by explicit information. This has not been emphasized and used in the version of the document but it is facilitated by the object-oriented approach taken.

1.7 Foreign Exchange Instruments

1.7.1 What market data is supported in this release of FpML?

This release of FpML supports the definition of exchange rates, and implied volatilities for FX Options.

- The exchange rate definition includes an optional direction specifying bid, ask, or mid.
- The implied volatility data is organized per currency pair with multiple tenors per currency pair.

The support for market data will be extended in the next release to include other data such as zero curves, correlations, interest rates etc.

1.7.2 What is an FXLeg?

The FXLeg component is common to FX Spot, FX Outright and FX Swap products. It represents the exchange of two currency amounts at a specified rate and date. Two FX Leg components are combined to form an FX Swap trade where the same currencies are exchanged on two different dates. Application logic will be used to determine whether the data contained within the two FX Leg components is consistent.

1.7.3 Can templates be arbitrarily combined to create new products?

The FpML definition is hierarchical in nature. Products are built from pluggable components called Templates. Various templates such as FXOptionTemplate, FXLegTemplate, FXBarrierTemplate, FXBinaryTemplate are combined as required to define products. These templates cannot be arbitrarily combined because products could be created which are not valid instruments.

1.7.4 How are the exchange rates expressed?

FX exchange rates can be expressed by specifying currency1, currency2, quote basis and the rate. The quote basis provides context to the rate and can either be number of currency1 per unit of currency2 or number of currency2 per unit of currency1.

1.7.5 What are FXCore and FXOptionCore and FRACore components?

Although these components are not meaningful in business terms, they represent a collection of fields that are common amongst a set of related products. The ‘core’ components were created to collect a set of common fields into one component that is reused as needed.

1.7.6 Why is ‘base currency’ omitted from the FXSpot, FXOutright and FXSwap and included in FXOptions?

The ‘baseCurency’ element is used in FX Option instruments to indicate which of the two currencies is regarded as the base currency for this FX trade. In conjunction with the FXRate and its term field, the baseCcy indicates the market convention being used: American ('base per term' or 'base per risk') or European ('term per base' or 'risk per base'). The 'baseCcy' must correspond to one of the two currencies involved in the trade.

1.7.7 Why are multiple settlement dates allowed?

It is possible for the currencies of a foreign exchange transaction to be exchanged on different days, however because this is a very rare occurrence normally one settlement date will be needed.

1.7.8 Why are fixings supplied together with fixing reference information?

The fixing reference information is recorded when the deal is executed before any fixings have occurred. During the life of the trade, fixings are applied to the trade and can be sent along with the reference information in order to determine risk or mark-to-market information, as the fixing rates must be supplied.

1.7.9 From whose perspective is FpML defined (buyer or seller)?

The FpML definition is party neutral in the sense that element tags define trade components generically. For instance, notional related information is defined using ccy1Amount or ccy2Amount and not buyAmount or sellAmount.

It is possible that a trade information could be transferred multiple times between two parties before it gets confirmed. The above concept eliminates any need to change the FpML to reflect a party's perspective.

1.7.10 Can multiple barriers be defined?

The FXBarrierCore component can be used to define single barrier products: Knock-in, Knock-out or multiple barrier products, such as a Double Knock-out.

1.8 Party Component

1.8.1 Is this a complete set of Party information?

No. The approach has been to provide the information necessary to cover the current scope of FpML. Release 1.0 contains only the information necessary to exchange data for trade confirmation and settlement.

1.8.2 Does Party provide information for Risk and Exposure management?

Only to the extent that the information currently provided is used for this purpose. Supplemental information will be added in subsequent releases to the current Party framework to support detailed risk and exposure reporting, e.g. industry codes and country of assets.

1.8.3 Can payment instructions be provided by swap stream?

Not yet. While multiple payment instructions per currency can be provided into the Settlement Instructions, the links between streams and payment instructions have not yet been provided.

1.8.4 Why does the Contact element have the Phone Number as a required component?

In most contexts, the contact is an optional element. However, if it is defined, some information should be provided so the person can be reached. The phone number is considered mandatory since the address is an optional element and just providing the name was considered inadequate information to reach him/her.

2 APPENDIX: CONVENTIONS

2.1 Goals

This portion of the document describes the various syntactic conventions used in FpML that are common to the family of FpML languages. Various aspects of element naming, element and metadata representation, and structure are also covered. Each convention is described verbally and is illustrated with an example, followed by a rationale. The appendix contains a more comprehensive example of the conventions.

The following goals were established before standards and conventions were defined:

- Similar "look and feel" across FpML sub-domains
- Ease of maintenance FpML family of languages
- Ease of development in separate groups
- Plug-n-play of objects to drive functionality
- Extensibility of standard in subsequent releases
- Development path towards schemata/namespaces
- Facilitate application level validation and, in subsequent releases, schema validation

2.2 Conventions

2.2.1 Prefix element names

Elements with same name could carry different meaning based on the context. To avoid name collisions, all element names in FpML are prefixed with the "module" it is defined under. For instance, an element *position* in *FXOption* context would be named as *FXOption:position*. For a detailed discussion, refer to the section titled "FpML Modularization Proposal" (Appendix 3 on page 42).

Rationale:

- Avoid element name collision
- Prepare for namespace support
- Prepare for schemata
- Enable distributed development

2.2.2 Use Camel Case naming notation

Element names that contain multiple words, should have the each word capitalized. This is known as "camel case" notation. *An exception to this convention is made for element names that contain an abbreviated word(s) in which case the abbreviated letters are in upper case.*

The above notation is used to name all FpML element names. Other special delimiting characters (like '_', '-', '.') are not used.

Do

```
<FXOptionData>
<position>buy</position>
<callAmount>
<Money>
<currency>USD</currency>
<amount>5000000</amount>
</Money>
</callAmount>
<!-- … -->
</FXOptionData>
```

Avoid

```
<FX-Option-Data>
<position>buy</position>
<call_Amount>
<Money>
<currency>USD</currency>
<amount>5000000</amount>
</Money>
</call_Amount>
<!-- … -->
</FX-Option-Data>
```

Rationale:

- Provide concise naming conventions in line with other naming conventions used on the internet and programming languages (e.g. JavaScript, Java, BizTalk).

2.2.3 Use capitalized class names and lowercase instance variable names

If an element reflects a thing (an object, a class or a table name) the UpperCamelCase convention is used. For a property, a reference etc, lowerCamelCase is used

In order to serialize an object in XML the class name of the object is used for the top-level element name representing the object. The following describes the naming conventions.

Class names are in UpperCamelCase
Instance variables of a class are in lowerCamelCase

This leads to a very regular structure, in which upper case element names and lowercase element names alternate. The only exception to this rule is reified collections, see section 3.6.

Do

```
<FXOptionData>
<position>buy</position>
<callAmount>
<Money>
<currency>USD</currency>
<amount>5000000</amount>
</Money>
</callAmount>
```

Do

```
<!-- … -->
</FXOptionData>
```

Avoid

```
<FXOptionData>
<Position>buy</Position>
<callAmount>
<money>
<Currency>USD</Currency>
<Amount>5000000</Amount>
</money>
</callAmount>
<!-- … -->
</FXOptionData>
```

Rationale:

- Provides clear identification of data as there is no positional dependency
- Provides a clear indication of compositional elements as it is always clear if an object or the state of an object is described.

2.2.4 Use elements to represent data content

FpML uses clear distinction between the use of elements and attributes. Elements are used to always represent data fields and attributes are used to represent element metadata.

Do

```
<FXOptionData>
<position>buy</position>
<callAmount>
<Money>
<currency>USD</currency>
<amount>5000000</amount>
</Money>
</callAmount>
<!-- … -->
</FXOptionData>
```

Avoid

```
<FXOptionData>
<position value=”buy”/>
<callAmount>
<Money currency=”USD”>5000000</Money>
</callAmount>
<!-- … -->
</FXOptionData>
```

Rationale:

- Using elements provide easier extensibility than attributes
- Complex structures cannot be modeled using attributes

2.2.5 Make non-primitive objects explicit

If the object pointed to by an instance variable is non-primitive, complex, or can take objects of different types inheriting from the same super type, the nested object is made explicit.

Do

```
<ConditionalRate>
<condition>…</condition>
<rate><FixedRate>5.769</FixedRate></rate>
</ConditionalRate>

<ConditionalRate>
<condition>…</condition>
<rate>
<FloatingRate>
<isda-rate-option>USD-LIBOR-BBA</isda-rate-option>
<maturity><multiple>3</multiple><unit>M</unit></maturity>
</FloatingRate>
</rate>
</ConditionalRate>
```

Avoid

```
<ConditionalRate>
<condition>…</condition>
<rate>5.769</rate>
</ConditionalRate>
```

Rationale:

- Provides flexibility for modeling reusable structures
- Provides easier maintainability when a new specialization of a base class needs to be introduced. Only the grammar specification need to be changed without impacting the XML data file
- Enables use of plug-n-play to plug in different components

2.2.6 Use nested form for collections of objects

Explicit nesting of sequences of similar objects. If different kind of collections could be provided, the collection itself is made explicit. Note, the latter case leads to the only exception of the alternating upper case, lower case element structure.

Do

```
<ObservationRate>
<rate>…</rate>
<strikes>
<Strike><date>1996-11-13</date><rate>7</rate></Strike>
<Strike><date>1997-05-13</date><rate>7</rate></Strike>
<Strike><date>1997-11-14</date><rate>7</rate></Strike>
</strikes>
</ObservationRate>

<ObservationRate>
<rate>…</rate>
<strikes>
```

Do
<SimpleStrikes> <Strike><date>1996-11-13</date><rate>7</rate></Strike> <Strike><date>1997-05-13</date><rate>7</rate></Strike> <Strike><date>1997-11-14</date><rate>7</rate></Strike> </SimpleStrikes> </strikes> </ObservationRate>

Avoid
<ObservationRate> <rate>…</rate> <Strike><date>1996-11-13</date><rate>7</rate></Strike> <Strike><date>1997-05-13</date><rate>7</rate></Strike> <Strike><date>1997-11-14</date><rate>7</rate></Strike> </ObservationRate>

Rationale:

- Allows clear identification of repeatable elements.

2.2.7 Add data type information to elements

Each element may have multiple attributes to describe its metadata, which an application can then use to perform validation.

The following conventions are used for defining attribute metadata. *Note all attributes are optional except e-dtype*.

e-dtype

This attribute defines the element's data type. It could either be a NOTATION or a datatype. FpML currently uses following datatypes.

string	content is a text string.
integer	content is an integer number.
float	content is a floating-point number.
number	content is a number
boolean	content is a boolean (true or false).
date	content is a date.
datetime.tz	content is a date and time with time zone information.
time.tz	content is a time with time zone information.
uuid	content is an object id.

e-dsize

This captures element's storage size. The following meaning is assigned to values of e-dsize:

e-dtype	e-dsize	Example
string	Maximum number of characters	23
float	Bytes (4 or 8)	4
number	x.y	14.4

	where x = digits allowed to left of decimal y = digits allowed to right of decimal	
int	Bytes (1, 2, 4 or 8)	8

e-dvalue
Attribute enumerates value(s) for an element. This could either be a list of space-separated values or a NOTATION. The attribute type is a NOTATION when the content of the element being modeled is provided by a well-known source (i.e. list of currencies is published by ISO).

e-dmin
attribute defines element's minimum value

e-dmax
attribute defines element's maximum value

Examples
For instance an element that represents a currency can be modeled using NOTATION as

```
<!NOTATION isoccy-4217 SYSTEM
"http://www.iso.ch/cate/d23132.html">

<!ELEMENT currency (#PCDATA)>
<!ATTRLIST currency
e-dtype NMTOKEN #FIXED "string"
e-dsize NMTOKEN #FIXED "3"
e-dvalue NOTATION (isoccy-4217) #IMPLIED>
```

Further, an element that represents a put or call option indicator can be modelled as

```
<!ELEMENT putCall (#PCDATA)>
<!ATTRLIST putCall
e-dtype NMTOKEN #FIXED "string"
e-dsize NMTOKEN #FIXED "4"
e-dvalue NMTOKENS #FIXED "put call">
```

2.2.8 Use known ISO standards for data formats

FpML uses ISO standards for element contents, where applicable. ISO currency and ISO date standards are used to describe the contents of currency and date related fields.

2.2.9 Limit use of abbreviations

Do
<FXOptionData> <position>buy</position> <callAmount> <currency>USD</currency>

Do
<amount>5000000</amount> </callAmount> … </FXOptionData>

Avoid
<FXOD> <pos>buy</pos> <ca> <ccy>USD</currency> <amt>5000000</amount> </ca> … </FXOD>

2.2.10 Use business centered names

Use of element names that make business sense.

Do	**Avoid**
<CancellableSwap> <optionOwner>Morgan</optionOwner> <optionDate>1998-08-03</optionDate> … </CancellableSwap>	<CancellableSwap> <by>Morgan</by> <on>1998-08-03</on> … </CancellableSwap>

3 APPENDIX: MODULARIZATION

This section summarizes the modularization approach used in the design of FpML. The underlying goal is to develop FpML using off-the-shelf validating parsers and DTDs for the initial release. It is anticipated that DTDs are in the process of being phased out as being the standard means of XML structure definition and validation. This is expected to change to XML schema after a schema standard has been accepted by the W3C committee. The intent of this proposal is to minimize modifications to the XML (and hence the applications) when this happened.

3.1 Objectives

The objectives of this modularization proposal are two fold:

(i) To provide a solution that will support structural validations with DTDs during the evolution of the FpML standard and through its initial release.

(ii) To provide a smooth transition to schema and true namespace support once the standard have been determined by the W3C committee and appropriate tools are available in the open market.

3.2 The approach

It is proposed that the colon (:) be used to prefix each element name with an abbreviated namespace-like notation.

The options available were to use the colon character ':', or using the dot-notation. In both cases validation using DTDs is possible. The colon notation was selected because it is anticipated that FpML files will be upward compatible with namespaces & schemata approaches which will provide a smoother transition from DTDs to schema at a later point in time.

3.2.1 Examples

The following three files illustrate this idea. Note, the content of these files is just for illustrating purposes, and does not use FpML proper.

File fpml.dtd

```
<!ENTITY % urn-fpml-Trade PUBLIC "" "Trade.dtd">
%urn-fpml-Trade;

<!-- FpML -->
<!ELEMENT fpml:FpML (trade:Trade)*>
<!ATTLIST fpml:FpML
  xmlns:fpml CDATA #FIXED "urn:fpml"
  xmlns:trade CDATA #FIXED "urn:trade"
 >
```

File: trade.dtd

```
<!ELEMENT trade:Trade (trade:start, trade:end) >
<!ATTLIST trade:Trade
  xmlns:trade CDATA #FIXED "urn:trade">

<!ELEMENT trade:start (#PCDATA)>
<!ELEMENT trade:end (#PCDATA)>
```

File test.xml

```
<?xml version="1.0"?>
<!DOCTYPE fpml:FpML SYSTEM "fpml.dtd" >

<fpml:FpML xmlns:fpml="urn:fpml"
           xmlns:trade="urn:trade">

     <trade:Trade>
          <trade:start/>
          <trade:end/>
     </trade:Trade>

</fpml:FpML>
```

3.3 Proposed Namespaces

Namespace support is not provided by the validating parsers for DTDs currently available in the market. However, the conventions proposed below would provide a mechanism to redefine elements relevant to their context and also set the stage for a relatively smooth transition to using namespaces later.

Each element should be prefixed by an abbreviated namespace as described in the previous section.

Each DTD will include, as ENTITY references, the definitions of shared elements it depends on. (The actual syntax for external reference will be slightly different as discussed on page 45).

3.3.1 Example

File: shared-fx.dtd

```
<!ELEMENT shared-fx:direction  (buyer , seller )>

<!ELEMENT shared-fx:buyer  (#PCDATA )>
<!ATTLIST shared-fx:buyer ccy CDATA  #IMPLIED>

<!ELEMENT shared-fx:seller  (#PCDATA )>
<!ATTLIST shared-fx:seller ccy CDATA  #IMPLIED>

<!ELEMENT shared-fx:instrumentID  (#PCDATA )>
…
```

File: FXLeg.dtd

```
<!ENTITY % shared-fx.dtd SYSTEM "shared-fx.dtd">
%shared-fx.dtd;

<!ELEMENT FXLeg  (shared-fx:instrumentID , shared-fx:direction ,
                  shared-fx:fxCore , fxleg:instrumentRates )>
<!ELEMENT fxleg:instrumentRates  (fxRate+ )>
```

File: Trade.dtd

```
<!ENTITY % TradeIDs.dtd PUBLIC "" "TradeIDs.dtd">
<!ENTITY % AccountDetails.dtd PUBLIC ""     "AccountDetails.dtd">
<!ENTITY % FXLeg.dtd PUBLIC ""              "FXLeg.dtd">
<!ENTITY % VanillaFixedFloat.dtd PUBLIC ""  "VanillaFixedFloat.dtd">
<!ENTITY % CallableSwap.dtd PUBLIC "" "CallableSwap.dtd">
…

%TradeIDs.dtd;
%AccountDetails.dtd;
%FXLeg.dtd;
%VanillaFixeFloat.dtd;
…

<!ELEMENT Trade  (trade:TradeID , accountDetails:AccountDetails ,
                 ( fxleg:FXLeg |
                   vanillaFixedFloat:VanillaFixedFloat    |
                   callableSwap:CallableSwap  |
… ))>
```

3.4 Transition to Schemata

One of the objectives is to ensure a smooth transition to schemata as the technology evolves. DTDs are in the process of being phased out but are the available and the supported standard at this time. This was a major factor in the choice of the schema development tool by Extensibility Inc.

Extensibility Inc. has indicated its keenness to stay on top of the evolving schema standards and provide support for the one(s) that rise above the crowd. In this regard, they have also indicated an intention to provide support for the following:

- Expand their Export DTD to Schema functionality to support evolving standards
- Provide customized functionality to strip away prefixes from element names during the conversion process if required.
- Define all DTDs as a namespace
- Allow a customizable prefix name

This may still require some hand or custom conversion effort, but this is expected to be minimal.

3.4.1 Upside

The advantages with this approach are:

- The standard can be defined with full structural validation using currently available technology and have a charted plan for transitioning to the evolving standards.
- Have the potential of releasing DTDs as part of the standard and FpML files generated accordingly be upwardly compatible with the schema approach.
- Have the ability to outsource the conversion process to a third party.
- Use inheritance typing mechanisms as soon as schemata provide them.

3.4.2 Downside

The disadvantages of using this approach are:

- Element names may become really long.
- Namespace prefixes would need to be abbreviated which would cause loss of readability
- The namespace names are fixed for any DTD validation.

3.4.3 Issues with the approach

Known issues with this approach are:

- Currently, the XML-Data reduced implementation mixes up namespaces and schema designation. The namespace name is identical with the schema file name. *This should not be a major issue because the XML-Data reduced implementation is not a currently accepted standard. However, it has been implemented in Internet Explorer 5.0's XML parser and is one of few schema versions available for parallel development and testing.*
- Not all referenced DTDs will be included in a DTD because elements would be defined more than once at the top levels, so it would be hard to write out a schema which has the correct namespaces in it. The proposed solution is to set up a flagging mechanism which says that in stand-alone mode certain DTDs have to be included. The downside of this approach is that DTDs appear more cryptic and contain positional inclusion dependencies. (A more detailed explanation of the flagging approach is provided below)
- XML-Authority currently defaults a namespace name when a DTD is exported into XML-Data format. (minor issue)
- The FpML files parse in validating mode with IBM's and Sun's parsers, they do not open in IE5.0. This issue can be investigated in parallel with defining the FpML language.

3.5 Modularization and duplicate ENTITY declarations

XML DTDs allow including and referencing external ENTITY declarations. However, multiple inclusions are not permitted and flagged as errors. This will be an issue for FpML as represented by figure I. As illustrated, Trade.dtd and FpML.dtd will contain duplicate declarations of the elements in shared.dtd.

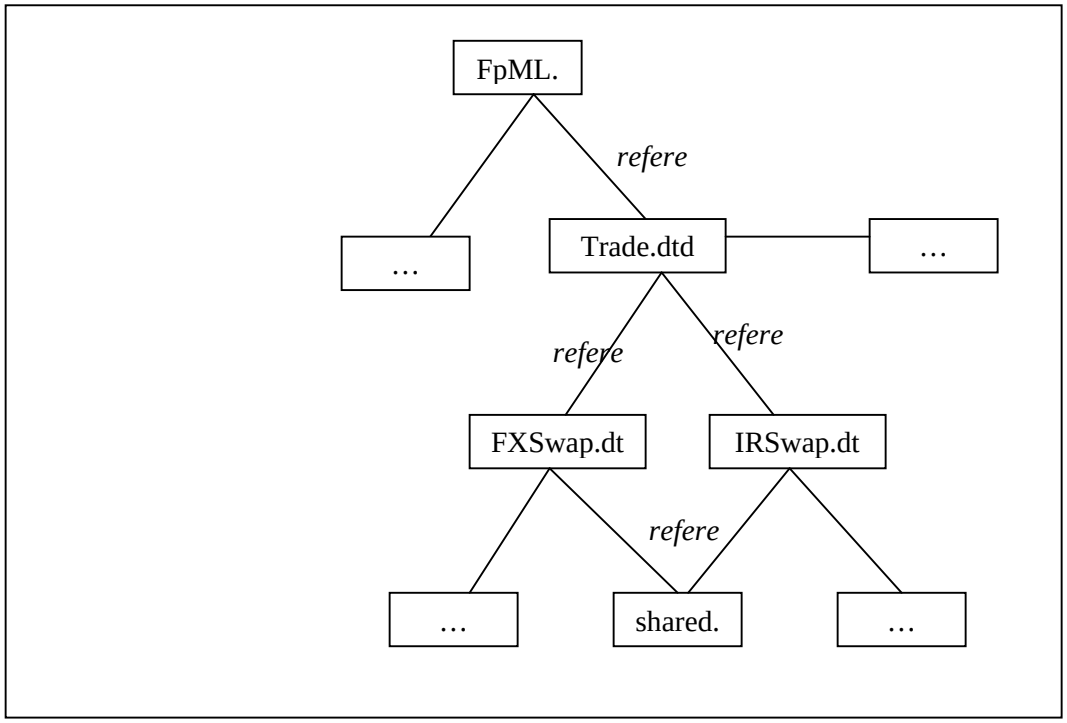


Figure 1

To resolve this issue, the FpML standard proposes to use the Conditional Sections as described in Section 3.4 of the XML Specification 1.0. This will be implemented as per the following mechanism:

If a file bar.dtd is included by foo.dtd. Another super dtd called foobar.dtd includes foo.dtd. In foo.dtd, one would define a reference parameter entity (called BAR) to include bar.dtd. In foobar.dtd, one would first include bar.dtd (the leaf level) and then define BAR to IGNORE and then include foo.dtd.

The trick is that the parser ignores/warns about duplication entity definition whereas it gives an error on duplicate element definition. The following example illustrates the above:

```
<!-- bar.dtd -->
<!ELEMENT bar (#PCDATA)>
```

```
<!-- foo.dtd -->
<!ENTITY % BAR 'INCLUDE'>

<![%BAR;[
<!ENTITY % bar.dtd PUBLIC "" "bar.dtd">
%bar.dtd;
]]>

<!ELEMENT foo (bar)>
```

```
<!-- foobar.dtd -->
<!ENTITY % bar.dtd PUBLIC "" "bar.dtd">
%bar.dtd;

<!ENTITY % BAR 'IGNORE'>
<!ENTITY % foo.dtd PUBLIC "" "foo.dtd">
%foo.dtd;

<!ELEMENT foobar (foo)>
```

```
<!-- foo.xml -->
<?xml version="1.0" standalone="no"?>

<!DOCTYPE foo SYSTEM "foo.dtd">
<foo>
      <bar>hello</bar>
</foo>
```

```
<!-- foobar.xml -->
<?xml version="1.0" standalone="no"?>

<!DOCTYPE foobar SYSTEM "foobar.dtd">
<foobar>
      <foo><bar>hello</bar></foo>
</foobar>
```

The advantage of this approach is that it provides a bridge over the issue until the FpML Standard is converted from DTDs to schema (the issue will be moot under schema).

The downside is that this approach includes order dependency and the inclusion sections of the FpML DTDs will seem less readable.

4 APPENDIX: MOZILLA PUBLIC LICENSE
VERSION 1.0

1. Definitions.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof**.**

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by **Exhibit A**.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by **Exhibit A** as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or a list of source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor’s choice. The Source Code can be

in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of fifty percent (50%) or more of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1. The Initial Developer Grant.

The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, or as part of a Larger Work; and

(b) under patents now or hereafter owned or controlled by Initial Developer, to make, have made, use and sell ("Utilize") the Original Code (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Original Code (or portions thereof) and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

2.2. Contributor Grant.

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code or as part of a Larger Work; and

(b) under patents now or hereafter owned or controlled by Contributor, to Utilize the Contributor Version (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Contributor Version (or portions thereof), and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

3. Distribution Obligations.

3.1. Application of License.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section **2.2**. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section **6.1**, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder.

However, You may include an additional document offering the additional rights described in Section **3.5**.

3.2. Availability of Source Code.
Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications.
You must cause all Covered Code to which you contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters

(a) Third Party Claims.
If You have knowledge that a party claims an intellectual property right in particular functionality or code (or its utilization under this License), you must include a text file with the source code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If you obtain such knowledge after You make Your Modification available as described in Section **3.2**, You shall promptly modify the LEGAL file in all copies You make available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.
If Your Modification is an application programming interface and You own or control patents which are reasonably necessary to implement that API, you must also include this information in the LEGAL file.

3.5. Required Notices.
You must duplicate the notice in **Exhibit A** in each file of the Source Code, and this License in any documentation for the Source Code, where You describe recipients' rights relating to Covered Code. If You created one or more Modification(s), You may add your name as a Contributor to the notice described in **Exhibit A**. If it is not possible to put such notice in a particular Source Code file due to its structure, then you must include

such notice in a location (such as a relevant directory file) where a user would be likely to look for such a notice. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions.

You may distribute Covered Code in Executable form only if the requirements of Section **3.1-3.5** have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section **3.2**. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works.

You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section **3.4** and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in **Exhibit A**, and to related Covered Code.

6. Versions of the License.

6.1. New Versions.
Netscape Communications Corporation ("Netscape") may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions.
Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Netscape. No one other than Netscape has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works.
If you create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), you must (a) rename Your license so that the phrases "Mozilla", "MOZILLAPL", "MOZPL", "Netscape", "NPL" or any confusingly similar phrase do not appear anywhere in your license and (b) otherwise make it clear that your version of the license contains terms which differ from the Mozilla Public License and Netscape Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in **Exhibit A** shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S

NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THAT EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in, the United States of America: (a) unless otherwise agreed in writing, all disputes relating to this License (excepting any dispute relating to intellectual property rights) shall be subject to final and binding arbitration, with the losing party paying all costs of arbitration; (b) any arbitration relating to this Agreement shall be held in Santa Clara County, California, under the auspices of JAMS/EndDispute; and (c) any litigation relating to this Agreement shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

Except in cases where another Contributor has failed to comply with Section **3.4**, You are responsible for damages arising, directly or indirectly, out of Your utilization of rights under this License, based on the number of copies of Covered Code you made available, the revenues you received from utilizing such rights, and other relevant factors. You agree to work with affected parties to distribute responsibility on an equitable basis.

EXHIBIT A.

"The contents of this file are subject to the Mozilla Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of the Original Code is PricewaterhouseCoopers in partnership with JP Morgan. All Rights Reserved."