Stylus Studio Case Study: FIXML– Working with Complex Message Sets Defined Using XML Schema

Introduction

The advanced XML Schema handling and presentation capabilities of Stylus Studio have valuable implications for users of industry-specific XML message sets. XML Schema is the W3C specification for building an abstract model of a message using meta-data (elements, attributes, and assemblies of these) that expresses the organization, structure, syntax and meaning of a message so that it is possible to generate an actual instance of a message directly from the schema model. Alternatively, if you have a message instance you can use its schema definition to validate and determine the processing requirements of the document. Once a message is defined using XML Schema it can be readily understood and used by people and machines alike. This can substantially reduce the time and effort involved in translating a message specification into a working application.

There are numerous industry-specific initiatives in place or under way to define and codify common, standardized vocabularies and message sets using the structures and datatypes defined by the W3C XML Schema specification. Many of these initiatives are “green field” endeavors that have no formal semantic precedents, while a number of them are also being created from existing, operative languages. An example of the latter is FIXML, the XML version of the Financial Information eXchange (FIX) protocol. The XML schemas for the FIXML message set were released in January 2004 by FIX Protocol Ltd., the organization that maintains FIX as an open standard. In March 2004 the Chicago Mercantile Exchange (CME) became the first adopter of FIXML by specifying its use for the real-time Position Management services it is starting to offer. The CME specification provided an opportunity to use Stylus Studio to work with the Position Maintenance Request schema, which defines the fundamental set of messages used by trading organizations to take and execute positions in futures or options through the CME.

XML Schema, like any robust language, provides numerous options for expressing the organization, structure, syntax and meaning of information. Schemas can be written in any number of ways using the various structural and datatype conventions defined in the XML Schema specification. While the different versions would ostensibly be correct in their conformance to the XML Schema specification; the usability, behaviour and performance of the variant schemas could be significantly different in the following respects:

- Transparency – How effectively the schema presents the overall message and message component definitions so that a person knowledgeable in XML Schema could readily understand its use and function.

- Organization – With nearly 100 messages and 1000 field definitions, the amount of information in the FIXML schemas is considerable. How well this information is organized affects the way the information can be accessed and navigated.

- Programmatic usability – The ultimate value of defining a message using XML Schema is to simplify and automate the programmatic generation or processing of a message.

Data Dictionary Style vs. Document Style Schemas

Comprehending FIXML’s overall organization, structure and functional intent poses immediate challenges. This is because the FIXML schemas are organized in a data dictionary style, as opposed to document style format. Document-style schemas contain only one root element (i.e. a top-level element, such as “PurchaseOrder”, that characterizes the type of message or function being described by the schema) and all the relevant metadata required to generate an instance of the document or to programmatically access the functions defined by the schema. Any referential definitions (pointers) are internal to the document as well. Document-style schemas are designed to represent discrete functions, such as a message type (e.g. a purchase order) or a set of operational instructions, such as a Web Services Description Language (WSDL) document. Alternatively, data dictionary style schemas are comprised of multiple schema files, with multiple schema documents (i.e. no single root element) within each file. The metadata is usually organized in a highly referential manner, with metadata objects in one schema being defined by objects in another schema.

FIXML Schema File Organization

The FIXML message schemas are found in multiple files organized by functional category – Allocation, Confirmation, Position, etc. The **fixml-positions-base** file contains the high-level schema definitions for the Position Maintenance Request, Position Maintenance Report, Request For Position, Request For Position Acknowledgement, Position Report and Assignment Report messages. This can be seen in **Illustration 1** which shows the tree structure of the **fixml-positions-base** as generated by Stylus Studio.

Illustration 1 – Tree structure of fixml-positions-base schema file as generated by Stylus Studio

By high-level we mean that the message schemas are abstract; they present "metadata outlines" that contain objects that reference other metadata objects in external files. **Illustration 2** shows the high-level schema model for the **PosMntReq** message. It identifies the element and attribute metadata tags that make up the message, and these reference various "types" (metadata objects) found in external files (**fixml-components-base**, **fixml-fields-impl** and **fixml-fields-base**). These object types specify the values that can be used with the message tags and they may also reference additional tags used in the message. **Illustration 3** shows a fragment of the **fixml-fields-impl** file in a tree structure view generated by Stylus Studio. There are approximately 1000 types defined in the **fixml-fields-impl** and **fixml-fields-base** files.

A developer's ultimate objective in working with the FIXML **PosMntReq** schema is to create an application that generates Position Maintenance Request messages that can be submitted to and used by the CME. A significant value of defining a message model in XML Schema is the ability to directly generate an actual message instance that conforms to the structure and content model of the schema. This in turn simplifies and reduces the programmatic effort of the developer. However, this efficiency is only possible if there is a direct correlation between the schema model of the message and the message itself. If the schema model becomes too abstracted, that is, if the actual metadata used to represent a field object in an instance document is more than one reference step removed from the high-level message definition (the pseudo root level schema), then it will not be possible to generate an instance document directly from the high-level message schema.

```
<xs:schema targetNamespace="http://www.fixprotocol.org/FIXML-4-4" xmlns="http://www.fixprotocol.org/FIXML-4-4"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="fixml-components-impl-4-4.xsd"/>
    <xs:group name="PositionMaintenanceRequestElements">
        <xs:sequence>
            <xs:element name="Pty" type="Parties_Block_t" maxOccurs="unbounded"/>
            <xs:element name="Instrmt" type="Instrument_Block_t"/>
            <xs:element name="Leg" type="InstrmtLegGrp_Block_t" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Undly" type="UndInstrmtGrp_Block_t" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="TrdSes" type="TrdgSesGrp_Block_t" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Qty" type="PositionQty_Block_t" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:group>
    <xs:attributeGroup name="PositionMaintenanceRequestAttributes">
        <xs:attribute name="TxnTyp" type="PosTransType_t" use="required"/>
        <xs:attribute name="Actn" type="PosMaintAction_t" use="required"/>
        <xs:attribute name="OrigReqRefID" type="OrigPosReqRefID_t" use="optional"/>
        <xs:attribute name="RptRefID" type="PosMaintRptRefID_t" use="optional"/>
        <xs:attribute name="BizDt" type="ClearingBusinessDate_t" use="required"/>
        <xs:attribute name="SetSesID" type="SettlSessID_t" use="optional"/>
        <xs:attribute name="SetSesSub" type="SettlSessSubID_t" use="optional"/>
        <xs:attribute name="Acct" type="Account_t" use="required"/>
        <xs:attribute name="AcctIDSrc" type="AcctIDSource_t" use="optional"/>
        <xs:attribute name="AcctTyp" type="AccountType_t" use="required"/>
        <xs:attribute name="Ccy" type="Currency_t" use="optional"/>
        <xs:attribute name="AdjTyp" type="AdjustmentType_t" use="optional"/>
        <xs:attribute name="CntraryInstrctnInd" type="ContraryInstructionIndicator_t" use="optional"/>
        <xs:attribute name="PriorSpreadInd" type="PriorSpreadIndicator_t" use="optional"/>
        <xs:attribute name="ThresholdAmt" type="ThresholdAmount_t" use="optional"/>
        <xs:attribute name="Txt" type="Text_t" use="optional"/>
        <xs:attribute name="EncTxtLen" type="EncodedTextLen_t" use="optional"/>
        <xs:attribute name="EncTxt" type="EncodedText_t" use="optional"/>
        <xs:attribute name="ReqID" type="PosReqID_t" use="optional"/>
        <xs:attribute name="TxnTm" type="TransactTime_t" use="optional"/>
    </xs:attributeGroup>
    <xs:complexType name="PositionMaintenanceRequest_message_t" final="#all">
        <xs:annotation>
            <xs:documentation xml:lang="en">PositionMaintenanceRequest can be found in Volume5 of the specification
        </xs:documentation>
            <xs:appinfo xmlns:x="http://www.fixprotocol.org/fixml/metadata.xsd">
                <xs:Xref Protocol="FIX" name="PositionMaintenanceRequest" ComponentType="Message"/>
                <xs:Xref Protocol="ISO_15022_XML"/>
            </xs:appinfo>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="Abstract_message_t">
                <xs:sequence>
                    <xs:group ref="PositionMaintenanceRequestElements"/>
                </xs:sequence>
                <xs:attributeGroup ref="PositionMaintenanceRequestAttributes"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="PosMntReq" type="PositionMaintenanceRequest_message_t" substitutionGroup="Message" final="#all"/>
```

Illustration 2 –High-level schema model for the PosMntReq message

Consequently, when confronted with a data-dictionary style schema it is usually necessary to synthesize and extract a document style schema of the message in order to use it in a functionally meaningful way in an application. The first step in doing this is to determine the actual field objects in the message instance. This information was obtained from a CME implementation guide that describes a message instance for a Position Maintenance Request message. The Party Block group contains a required metadata tag – the **Sub@ID** and **Sub@Typ** attributes. To find the definitions of these attributes and the enumeration values defined for use with them, we followed the schema references starting with the **PositionMaintenanceRequest_message_t** type reference that is shown in the **PosMntReq** schema in the bottom of **Illustration 2**. **Illustration 4** shows the sequence of steps and locations traversed in order to find this attribute tag and its values. These schema fragments were extracted from 11 different locations in 3 separate schema files. It took hours of work to find where the defining information was for just one attribute by navigating through the native XML schema file and working with any number of available schema editing tools. Obviously, with dozens of elements and attributes in a Position Maintenance Request schema working directly with the XML notation and manually navigating through the FIXML schema files is not an effective way to address complex XML schema information organized in data dictionary style schemas.

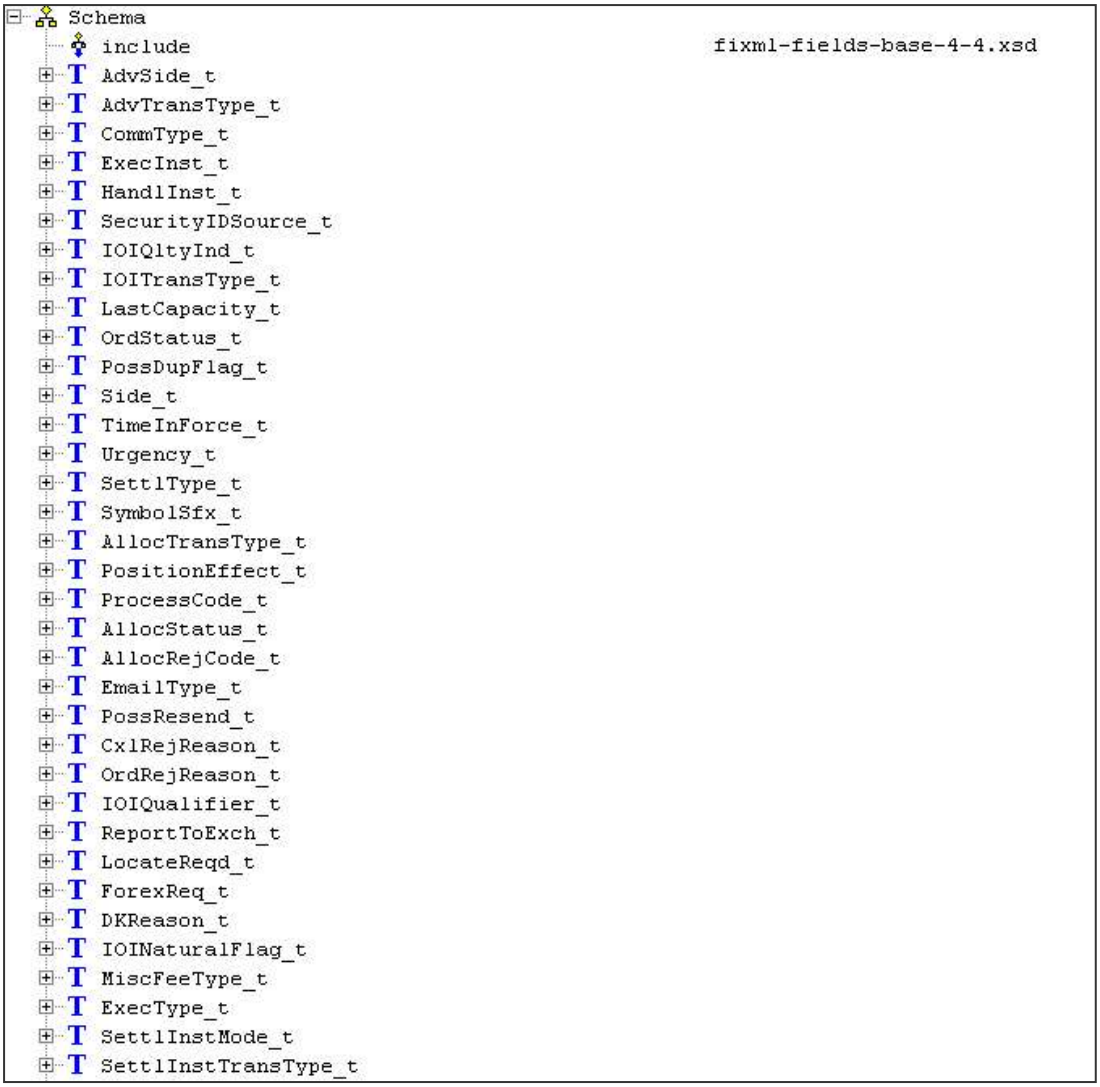


Illustration 3 – Tree structure of fixml-fields-impl schema file showing the random organization of simpleTypes

XML Editing Tool Requirements For Working with Data Dictionary Style Schemas

Most XML Schema editing and manipulation tools are designed to work with document-style schemas but do not have the advanced facilities to work with highly referential, data dictionary-style schemas. Examples of these advanced facilities include:

- The ability to load all referenced schemas (i.e. support for the XML Schema “include” declaration)
- A mechanism to automate the navigation of schema object references throughout multiple schemas.
- A way to logically re-order schema objects
- The ability to access information embedded in the documentation/annotation elements of a schema

This is where Stylus Studio stands out. Using the FIXML Position Maintenance Request message schema as an object lesson, we will demonstrate the unique, robust XML Schema handling and presentation facilities of Stylus Studio that make it an indispensable tool for working with highly complex XML Schema.

By employing Stylus Studio’s XML Schema Documentation facility we were able to quickly determine the inventory of elements and attributes that comprise a Position Maintenance Request message. The XML Schema Documentation view generates an HTML presentation of a schema in which each schema object is hyperlinked to the objects it references; multiple schemas for which there are nested **include** declarations are also presented in an intuitive graphical fashion.

From Position Base:

```
<xs:element name="PosMntReq" type="PositionMaintenanceRequest_message_t"
substitutionGroup="Message" final="#all"/>
```

From Position Base:

```
<xs:complexType name="PositionMaintenanceRequest_message_t" final="#all">
<xs:complexContent>
<xs:extension base="Abstract_message_t">
<xs:sequence>
<xs:group ref="PositionMaintenanceRequestElements"/>
</xs:sequence>
<xs:attributeGroup ref="PositionMaintenanceRequestAttributes"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

From Position Base:

```
<xs:group name="PositionMaintenanceRequestElements">
<xs:sequence>
<xs:element name="Pty" type="Parties_Block_t" maxOccurs="unbounded"/>
<xs:element name="Instrmt" type="Instrument_Block_t"/>
<xs:element name="Leg" type="InstrmtLegGrp_Block_t" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="Undly" type="UndInstrmtGrp_Block_t" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="TrdSes" type="TrdgSesGrp_Block_t" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="Qty" type="PositionQty_Block_t" maxOccurs="unbounded"/>
</xs:sequence>
</xs:group>
```

From Components Base:

```
<xs:complexType name="Parties_Block_t" final="#all">
<xs:sequence>
<xs:group ref="PartiesElements"/>
</xs:sequence>
<xs:attributeGroup ref="PartiesAttributes"/>
</xs:complexType>
```

From Components Base:

```
<xs:group name="PartiesElements">
<xs:sequence>
<xs:element name="Sub" type="PtysSubGrp_Block_t" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:group>
```

From Components Base:

```
<xs:attributeGroup name="PartiesAttributes">
<xs:attribute name="ID" type="PartyID_t" use="optional"/>
<xs:attribute name="Src" type="PartyIDSource_t" use="optional"/>
<xs:attribute name="R" type="PartyRole_t" use="optional"/>
</xs:attributeGroup
```

From Components Base:

```
<xs:complexType name="PtysSubGrp_Block_t" final="#all">
<xs:sequence>
<xs:group ref="PtysSubGrpElements"/>
</xs:sequence>
<xs:attributeGroup ref="PtysSubGrpAttributes"/>
</xs:complexType>
```

From Components Base:

```
<xs:group name="PtysSubGrpElements">
<xs:sequence/>
</xs:group>
```

From Components Base:

```
<xs:attributeGroup name="PtysSubGrpAttributes">
<xs:attribute name="ID" type="PartySubID_t" use="optional"/>
<xs:attribute name="Typ" type="PartySubIDType_t" use="optional"/>
</xs:attributeGroup>
```

From Fields Base:

```
<xs:simpleType name="PartySubID_t">
<xs:restriction base="xs:string"/>
</xs:simpleType>
```

From Fields Base:

```
<xs:simpleType name="PartySubIDType_enum_t">
<xs:restriction base="xs:string">
<xs:enumeration value="1"/>
<xs:enumeration value="2"/>
<xs:enumeration value="3"/>
<xs:enumeration value="4"/>
<xs:enumeration value="5"/>
<xs:enumeration value="6"/>
<xs:enumeration value="7"/>
<xs:enumeration value="8"/>
<xs:enumeration value="9"/>
<xs:enumeration value="10"/>
<xs:enumeration value="11"/>
```

Illustration 4 – The steps and files navigated to find the Sub@ID and Sub@Typ message values

Utilizing Stylus Studio's XML Schema Documentation View Facility

Illustration 5 shows the **fixml-positions-base** schema when displayed in the XML Schema Documentation view of Stylus Studio. Note that the five abstract root elements declared in the schema are hyperlinked. The result of clicking on the **PosMntReq** element, for example is shown in **Illustration 6**.

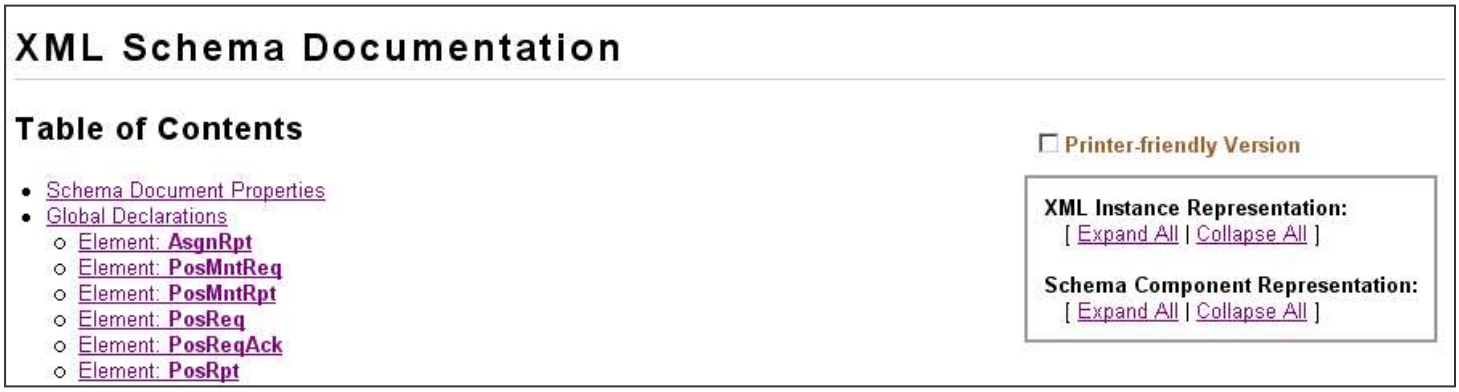
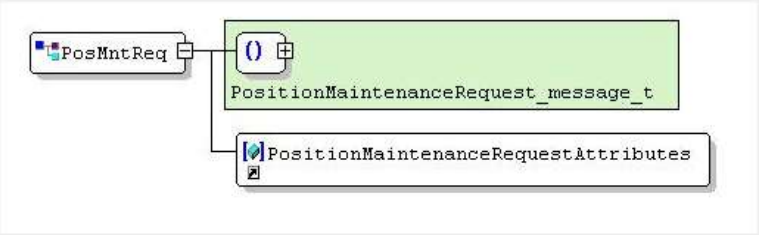


Illustration 5 –Table of Contents fragment generated for the fixml-positions-base schema file

Element: PosMntReq

- *This element can be used wherever the following element is referenced:*
  - Message

Name	PosMntReq
Type	PositionMaintenanceRequest_message_t
Nillable	no
Abstract	no
Substitution Group Exclusions	restriction, extension
Diagram	

XML Instance Representation

```
<PosMntReq
 TxnTyp="PosTransType_t [1]"
 Actn="PosMaintAction_t [1]"
 OrigReqRefID="OrigPosReqRefID_t [0..1]"
 RptRefID="PosMaintRptRefID_t [0..1]"
 BizDt="ClearingBusinessDate_t [1]"
 SetSesID="SettlSessID_t [0..1]"
 SetSesSub="SettlSessSubID_t [0..1]"
 Acct="Account_t [1]"
 AcctIDSrc="AcctIDSource_t [0..1]"
 AcctTyp="AccountType_t [1]"
 Ccy="Currency_t [0..1]"
 AdjTyp="AdjustmentType_t [0..1]"
 CntraryInstrctnInd="ContraryInstructionIndicator_t [0..1]"
 PriorSpreadInd="PriorSpreadIndicator_t [0..1]"
 ThresholdAmt="ThresholdAmount_t [0..1]"
 Txt="Text_t [0..1]"
 EncTxtLen="EncodedTextLen_t [0..1]"
 EncTxt="EncodedText_t [0..1]"
 ReqID="PosReqID_t [0..1]"
 TxnTm="TransactTime_t [0..1]">
   <Hdr> MessageHeader_t </Hdr> [0..1]
   <Pty> Parties_Block_t </Pty> [1..*]
   <Instrmt> Instrument_Block_t </Instrmt> [1]
   <Leg> InstrmtLegGrp_Block_t </Leg> [0..*]
   <Undly> UndInstrmtGrp_Block_t </Undly> [0..*]
   <TrdSes> TrdgSesGrp_Block_t </TrdSes> [0..*]
   <Qty> PositionQty_Block_t </Qty> [1..*]
</PosMntReq>
```

Schema Component Representation

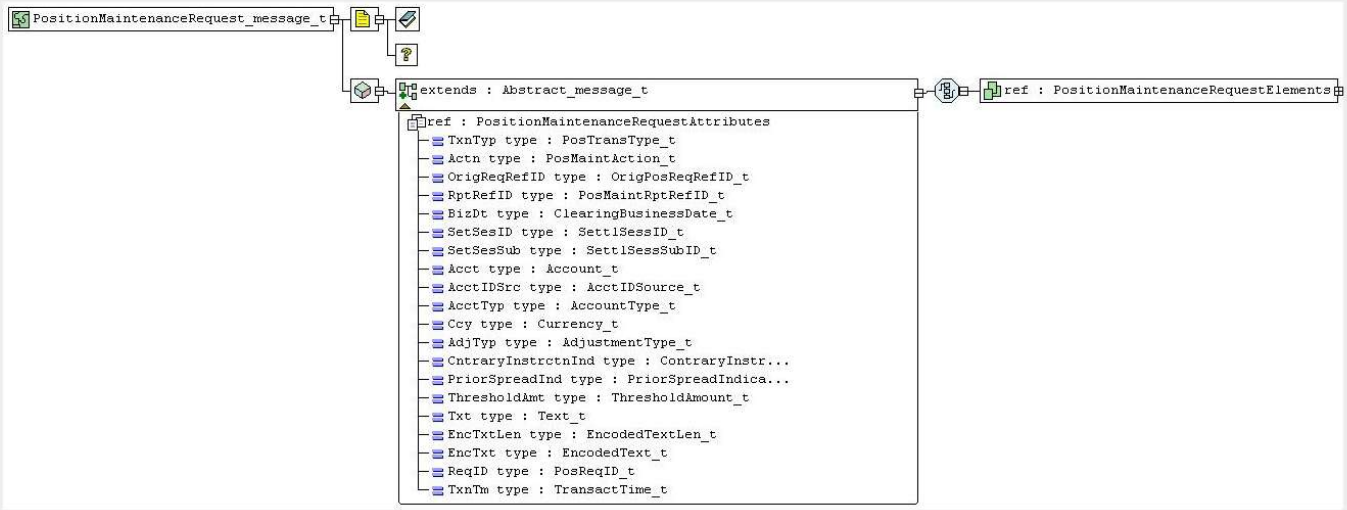
```
<xs:element name="PosMntReq" type="PositionMaintenanceRequest_message_t" substitutionGroup="Message" final="#all"/>
```

Illustration 6 – XML Schema Documentation view generated for the PosMntReq element by Stylus Studio

Illustration 6 shows the documentation generated for the **PosMntReq** element from the information found in the **fixml-positions-base** schema file. The element declarations are stated at the top, and the documentation indicates that the **PosMntReq** element is defined from the **PositionMaintenanceRequest_message_t** type. A diagram is also generated that illustrates the high-level structure of a **PosMntReq** element instance. Note that the diagram shows that the **PosMntReq** element contains a **PositionMaintenanceRequestAttributes** group, even though the Schema Component Representation does not indicate this. Because the **PositionMaintenanceRequest_message_t** definition is located in-line within the **fixml-positions-base** schema file, Stylus Studio directly referenced the **PositionMaintenanceRequest_message_t** definition to furnish this information.

Complex Type: PositionMaintenanceRequest_message_t

Super-types:	Abstract_message_t < **PositionMaintenanceRequest_message_t** (by extension)
Sub-types:	None

Name	PositionMaintenanceRequest_message_t
Abstract	no
Prohibited Derivations	restriction, extension
Documentation	PositionMaintenanceRequest can be found in Volume5 of the specification
Application Data	<xs:Xref Protocol="FIX" name="PositionMaintenanceRequest" ComponentType="Message"/> <xs:Xref Protocol="ISO_15022_XML"/>
Diagram	

+ XML Instance Representation

- Schema Component Representation

```
<xs:complexType name="PositionMaintenanceRequest_message_t" final="#all">
   <xs:complexContent>
      <xs:extension base="Abstract_message_t">
         <xs:sequence>
            <xs:group ref="PositionMaintenanceRequestElements"/>
         </xs:sequence>
         <xs:attributeGroup ref="PositionMaintenanceRequestAttributes"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Illustration 7 – XML Schema Documentation view for the PositionMaintenanceRequest_message_t complexType generated by Stylus Studio

The documentation generated for the **PositionMaintenanceRequest_message_t** complexType is shown in **Illustration 7**. The references to the **PostionMaintenanceRequestElements** and **PositionMaintenanceRequestAttributes** groups can be seen In the Schema Component Representation

The Value of the XML Instance Representation

An important feature of the XML Schema Documentation view is the XML Instance Representation, which presents a clear and comprehensible document style schema model for the message instance. The XML Instance Representation indicates the actual attribute and element metadata tags that will be generated in the instance document in the order in which they will appear. By being able to clearly visualize the overall structure of the message, it becomes significantly easier for the developer to navigate the referenced type definitions containing the enumerated values of the attributes of the **PosMntReq** root element as well as the attributes and elements of the **PosMntReq** child elements (HDR, Pty, Instmt, etc.). Furthermore, by hyperlinking the types in the Instance Representation to their definitions in the referenced schemas, the XML Schema Documentation view facilitates significant efficiencies in navigating widely dispersed schema objects.

By clicking on the **PosTransType_t** type object that defines the **TxnTyp** attribute in the XML Instance Representation, Stylus Studio automatically loads the **fixml-fields-impl** schema file and navigates to the location of the **PosTransType_t** simpleType object, as shown in **Illustration 8**.

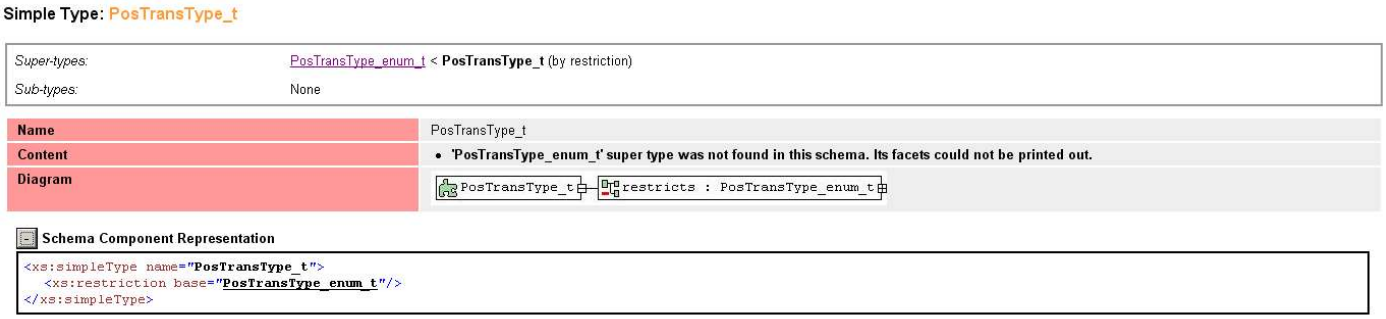


Illustration 8 – XML Documentation view for the PosTransType_t simpleType object

As the Schema Component Representation indicates, the **PosTransType_t** type object is further defined as a **PosTransType_enum_t** type object. By subsequently clicking on the **PosTransType_enum_t** link in the Schema Component Representation, Stylus Studio automatically loads the **fixml-fields-base** schema file and navigates to the **PosTransType_enum_t** definition shown in **Illustration 9**.

The Schema Component Representation of the **PosTransType_enum_t** indicates that this type object contains the enumerated values of 1, 2, 3, 4 and 5. These enumerated values are further qualified by information in the Application Data documentation section, which indicates that they represent the Position Maintenance Request transactions “Exercise”, “Do Not Exercise”, “Position Adjustment”, “Position Change Submission Margin Disposition”, and “Pledge”, respectively. One of these five code values must be specified for the **TxnTyp** attribute by the application that will generate the **PosMntReq** message.

Illustration 9 – XML Documentation view for the PosTransType_enum_t simpleType object

Using Stylus Studio to Make Sense of Random Organization

As indicated earlier, the **fixml-fields-impl** and **fixml-fields-base** schema files contain nearly 1000 type definitions. These type definitions are organized randomly in the **fixml-fields-impl** file, and sequentially by the original FIX tag number in the **fixml-fields-base** file. **Illustration 3** shows the Stylus Studio Tree view of a fragment of the **fixml-fields-impl** schema file and its random order. In either case, the random or tag sequence organization of such a large volume of objects contributes to the difficulty of working with the schemas, especially considering the way in which the referenced objects are dispersed. Fortunately, Stylus Studio remedies this problem by presenting the schema objects

in these files alphabetically in its XML Schema Documentation view. **Illustration 10** shows the Table of Contents for the **fixml-fields-impl** schema file generated by the XML Schema Documentation view. This feature alone facilitated a significant improvement in the ability to navigate and work with the FIXML schemas.

XML Schema Documentation

Table of Contents

Illustration 10 – Table of Contents generated by the XML Schema Documentation view for the fixml-fields-impl schema file

The same navigation exercise described for the **PosTransType_t** type object had to be executed for each of the attributes and elements of the PosMntReq document style schema in order to determine the actual values and attributes that are required for the construction of a Position Maintenance Request message. Without the schema handling and presentation capabilities of Stylus Studio, this documentation assembly task could easily have taken weeks to accomplish. Using Stylus Studio, the entire PosMntReq message was deciphered and documented within two days, providing an immediate return on investment in the very first project.

Conclusion

The complex organization and structure of the FIXML schemas is not atypical. Most industry initiatives to codify common vocabularies and message sets using XML Schema result in the creation of highly referential data dictionary style schemas. As this case study demonstrates, the XML presentation capabilities of Stylus Studio have a fundamental effect on a developer's ability to work with complex XML information in an efficient and meaningful way.

The developers of Stylus Studio correctly anticipated the complexity that XML Schema engenders when it is used to model substantial volumes of information and they specifically designed Stylus Studio to negotiate large quantities of abstract, referential XML Schema information and present it in a variety of ways that allow it to be clearly understood and managed. Anyone facing a project that involves working with complex schemas, such as FIXML, will find Stylus Studio to be an indispensable tool that will dramatically improve their ability to manage such a project while simultaneously improving their productivity and efficiency.