Software & Information Industry (SIIA) Financial Information Services Division (FISD)

FISD's XML Messaging Specification

"fisdMessage" Version 1.0-beta 16 December 2003

fisdMessage Reference Guide

http://www.mddl.org/ -> http://www.fisd.net/ -> http://www.siia.net/

SIIA/FISD Chief Technologist:
James E. Hartley, FISD/SIIA, Denver, U.S.A.

Editor:
James E. Hartley, FISD/SIIA, Denver, U.S.A.

Abstract

The FISD Message Technical Specification ("fisdMessage") defines the protocol and "on-wire" encoding of statically formed eXtensible Markup Language (XML) documents in a flexible and efficient manner. The fisdMessage specification represents a standard for the *transmission* and *reception* of certain types of XML documents and defines neither the content of the data distributed nor the use of that information.

It is intended that applications (via toolkits) that implement this protocol faithfully, consistent with XML processing guidelines, will enable senders to distribute high-volumes of content changes in the lowest possible bandwidth while maintaining the ability to add content without breaking processing applications.

This document uses the Market Data Definition Language (MDDL) to illustrate examples and demonstrate the functionality. MDDL is a specification based on the XML standard to enable interchange of data necessary to account for, to analyze, and to trade instruments of the world's financial markets (see http://www.mddl.org/).

Acknowledgements

This specification is a cooperative effort of representatives of the financial industry at the behest of the general membership of the Financial Information Services Division of the Software & Information Industry Association. Those organizations, and the individual contributors, are hereby thanked and commended for contributing staff and energy to this volunteer collective. Inherit competition has been cast aside and common goals have been pursued to bring this industry-wide goal to fruition.

Conventions Used in This Document

The following highlighting is used to present technical material in this document:

```
Complete examples or fragments
```

The following highlighting is used for commentary in this document:

Example
`<fisdMessage version="1.0-beta" > (Incomplete examples or fragments)`
And an explanation of the example.

NOTE: General comments directed to all readers.

All **`fisdMessage`** specific XML elements and attributes will be **`bolded in monospace`**. All words, other than elements or attributes, for which **`fisdMessage`** has a specific definition will be *colored and italicized*. The following example paragraph illustrates relevant highlighting:

The main XML element of an FISD Message instance document, **`fisdMessage`,** has one required attribute, **`version`**, containing the explicit version of FISD's XML Messaging Specification W3C Schema used to validate the document. The element **`fisdMessage`** must then contain a **`header`** element followed by any of the valid *constructs* for the referenced version of **`fisdMessage`**.

Status of this Document

This is the first formal release of this document and is available coincident with the public announcment of FISD Message Version 1.0-beta at the FISD Quarterly Meetings held 10 and 11 December 2003 in New York City.

This document is independent of the other MDDL-related specifications except **`mddlService`** which defines similar underlying concepts to **`fisdMessage`**. Revisions will be noted within the Change Log section of this document. All modifications to either specification will likely necessitate a re-release of this document.

NOTE: This is a "beta" document showing a plausible solution to the basic problem. Significant technical review and modification is likely as the specification is discussed among FISD/MDDL members. Please contact FISD if you have any questions or comments. See http://www.fisd.net/.

History of MDDL and fisdMessage

The original concept for the Market Data Definition Language was envisioned by Jeremy Sanders in 4th quarter of 2000. While there were some foundational activities by members of the Financial Information Services Division (FISD) of the Software & Information Industry Association (SIIA), it was not until Mr. Sanders urged the FISD membership to consider the concept of a standardized XML for market data that any coordinated industry-wide activity began.

The first structured meeting was held on 17 January 2001 at the offices of Merrill Lynch, in New York City, involving some thirty different participants from market data companies wanting to harmonize the vocabulary of market data via XML. Using separate offerings from Bridge Information Systems (now Reuters) and Dow Jones labeled “mdml” or “Market Data Markup Language” as a basis, the group formed the MDDL committees that would develop the vocabulary and the technical aspects of the new specification to become an industry-wide standard.

The first formal release of MDDL, “mddl 1.0-final”, was presented at the 5th World Financial Information Conference in London, on 02 November 2001, signaling the successful coordination of competitors to provide a unified foundation for forward development. Through the addition of various asset classes and other constructs, MDDL now stands as the prominent market data standard for a wide range of data content fulfilling various market data needs. Implementations of market data systems and data feeds with MDDL as a foundation are now evident throughout the industry. Through the work of FISD members doing real implementations, MDDL is maturing to provide the full capability necessary for a modern market data system.

In the 1st quarter of 2003, it became obvious that several additional technical capabilities were necessary to enable MDDL adoption at a broader scale. These shortcomings are identified when the following questions are proffered:

- MDDL provides a mechanism for formatting market data to be sent from provider to consumer. How does the consumer ask for the data of interest? The result - the definition of **`mddlQuery`** as the “query” to support MDDL’s “response” nature.
- How does one get an MDDL document from one point A to point B? The result - the definition of **`mddlService`** to identify query/response interfaces like web services.
- The value of MDDL for reference data is apparent but two different ways of handling market data - one for static data and one for realtime - does not seem reasonable. Is there no way to use MDDL for streaming of realtime market data? The result - the definition of **`fisdMessage`** to stream MDDL (and other) content.
- Accepting multiple data feeds comes with the burden of developing and maintaining separate data feeds from many sources or, similarly stated, focusing on distributing quality and new content is important but too many resources are devoted to researching, developing, and maintaining data feeds. Is there no way to leverage industry standards? The result - more support for **`fisdMessage`**.

The **`fisdMessage`** specification is the realization of a standards-based realtime streaming data feed. In effect, **`fisdMessage`** defines an “industry-standard” data feed protocol for transmitting statically formed content from producers to consumers. The proper implementation of an **`fisdMessage`** enabled data feed thus allows a consumer to accept encoded XML content from any **`fisdMessage`** enabled provider.

Overview of fisdMessage

The fisdMessage specification is intended as a standard protocol for the exchange of XML content. Specifically, fisdMessage facilitates an industry standard protocol for the financial industry to exchange market data content. Note that fisdMessage does not seek to dictate *what* data is exchanged over such a datafeed, only *how* that data is delivered. For the financial industry, Market Data Definition Language (MDDL) is the natural choice for market data feed applications but any relevant XML derived specification may be appropriate depending on the capabilities of the provider and the needs of the consumer.

The fisdMessage protocol is predicated on the fact that most data for a particular class of information, as distributed from a source or aggregator, has the same structure when represented in XML. For example, one can abstract that all equities reported from a particular exchange will have the same datafields for each of the instruments traded. This fact is evident in the existing datafeeds that are prevalent in the financial industry - although the exact fields available for specific instruments varies, the datafeeds are able to deliver the exact same fields for all instruments of a particular class. The fisdMessage protocol capitalizes on this normalization and seeks to formalize a datafeed based on this consistency of source content.

The initial test case for fisdMessage is market data (specifically MDDL) but the protocol is applicable to all XML derived specifications that result in documents that have the same "static" structure - the only real differences between one instance of the document and the next being the content of the elements within the document. The protocol provides a structure for delivering the maximum number of updates to the content with the mimimal use of bandwidth - a key factor in many realtime updating systems.

The fisdMessage protocol utilizes a "template" that outlines the static structure of an instance document. The template identifies all of the fields within the XML document that are of interest to the datafeed. Individual "instances" of that template are created to carry the values for a unique dataset of interest - for example, there would be an instance for each stock traded on a particular exchange that references the template of content for that exchange. The protocol uses messages to define a template (or multiple templates), create an instance of a template (also called an "image" or "refresh"), and update fields within the instance.

Table of Contents

Index of Figures

Index of Tables

Change Log

2003/12/05	JE Hartley	First formal release.
2003/12/16	JE Hartley	Added provider vs. framework directories

Pending Issues

- Look for a better way to change scheme values. Perhaps the list of legal schemes for a particular attribute can be placed in an enumeration. The existing syntax results in ill-formed XML documents AND requires a very large message if it needs to be changed.

- Develop an application, protocol, and flow graph. Primary to this is finding a point in the stack where this particular method of delivering content can be replaced with another while still maintaining a standard higher level intereface.

- Consider that a turnaround key should be prefixed with user identify of some sort. In other words, establish guidelines or format for the key.

- A mechanism for specifying an alternate default compression scheme for most of the Session Establishment messages and the AMsg Directory Response needs to be investigated and formalized. However, the list should be restricted and known.

- The convention for session establishment and negotiation of encryption needs to be established, formalized, and documented. The use of three-way handshake, trusted third party, and Diffie-Hellman key exchange in combination with AES/DES, RSA, MD5, and combinations of digital signatures needs to be documented.

- Clarify the way to add additional fields to an already distributed template. Some fields will be defined as “repeating sets” which, when specified, can dynamically increase the number of fields.

- Specify the timeout values for Session Establishment communications. Other message timeouts are a function of the prevailing AMsg Timestamp frequency.

- Processing rules for each message need to be emphasized.

- The schema needs to be completed and made available.

- Although the framework distribution order is enumerations, templates, directory this creates some problems if the compression schemes for integers and timestamps are different than the defaults. An abbreviated directory message needs to be created that contains only this information for use during these updates.

- Verify that 64kB-48-20-2 is a reasonable upper limit for the size of an fisdMessage protocol packet. Thus, the packet will fit within TCP over IPv4 or IPv6.

- The “CMsg Blocked Field Update” needs to be completed.

1. Introduction

The realities of distributing realtime streaming XML-encoded market data content (like MDDL - Market Data Definition Language) has led to the need for a compaction scheme and messaging protocol for efficiently conveying XML content from *sender* to *recipient*. The **fisdMessage** specification defines session establishment protocols as well as "on-wire" message structures that are suitable for high-volume high-throughput of statically formed XML documents. Proper adherence to the specification enables any **fisdMessage** capable receiver to accept all content from any **fisdMessage** conformant distributor via interactive or broadcast network media.

The **fisdMessage** protocol uses a "dynamic template" methodology whereby a *template* of the content is distributed and then specific *instances* of that *template* are referenced for each unique data object within the system. The overriding assumption is that (generally) all *instances* of a particular source within the system will require the same content. Thus, there is a unique *template* for each type of data collection, and many references to that *template* by each *instance* of that same type.

fisdMessage includes features for redefining content models so that new information can be added to the governing *template*, consistent with XML processing guidelines, without affecting existing applications. Thus, when new data needs to be made available via a datafeed, a provider may add the content and begin distributing new information immediately – via the **fisdMessage** protocol – without modification to existing **fisdMessage** compliant receiving applications. The receiving applications may then be upgraded to process the new content when convenient.

fisdMessage DOES NOT define the network media or the configuration necessary to exchange content between provider and consumer. **fisdMessage** defines the packets of data necessary to convey the source data, without loss, to the target application. The implementer must provide the binding to the network transport (and lower) layers. It is envisioned that a *sender* and *recipient* each would use toolkits that provide a viable implementation of **fisdMessage** for a common network infrastructure. **fisdMessage** is compatible with broadcast (multicast) and interactive networking modes.

The examples in this document use Market Data Definition Language (MDDL) as the content to be streamed with **fisdMessage**. Specifically, the examples discuss market data encoded in MDDL for realtime streaming of quotes and trade reports. The reader should note that, although the specification was originally designed with MDDL in mind, **fisdMessage** has applicability beyond market data and the financial industry.

MDDL is the XML derived specification to enable the interchange of information necessary to account for, to analyze, and to trade financial instruments of the world's markets. MDDL defines a common vocabulary and format for exchange to facilitate the interaction between exchanges, vendors, redistributors, and consumers. The common market data terminology embraced by MDDL allows providers to clearly state the nature and origin of market data elements thus removing ambiguity. The straightforward format of MDDL provides a convenient vehicle for exchanging this data between XML aware systems.

1.1. Purpose

This document describes the methodology and processing guidelines to send and/or receive the **`fisdMessage`** protocol. The messages are categorized as three major types with a section assigned to each:

Session Establishment: The communication necessary for a *recipient* to connect to a *sender*, authenticate, and configure to begin to receive administrative and content messages. This section does not apply for broadcast scenarios (where there is no backchannel for authentication) – in such cases, the *recipient* must be preconfigured manually for proper encoding with frequent update support by the broadcaster. Session Establishment messages also include “query” messages sent by the *recipient* to specify which content is of interest to the consumer for that session.

Administrative Messages: The messages, sent by the *sender*, which provide the content *templates* and other dynamic configuration parameters (collectively called the framework). Once configuration information is received, the *recipient* can begin decoding the content messages. In a broadcast scenario, where there is no backchannel for queries, these definition messages may only be distributed once or twice a day at the discretion of the *sender*, or provided via some other mechanism for initial configuration. Administrative messages also include reconfiguration messages used to optimize distribution.

Content Messages: The content messages with updates to particular *instances* of a type of data (as outlined in the Administrative Messages). There are two basic types of content messages distributed from the *sender* – the *image* or “refresh” containing the current values for all of the fields within an instance; and the *update* containing changes to specific fields within the *image*. It is up to the receiving application to decide how to store this information, or act upon it, but **`fisdMessage`** does specify how the *image* should be modified. The *sender* may also send query messages – consistent with an applicable query specification like **`mddlQuery`** – to specify which content is of interest or to make a request for historical or other non-realtime streaming information.

1.2. Terminology

The documentation uses various terms that have a specific meaning within the confines of this specification. Those terms related to **`fisdMessage`** are defined here as a way to clarify the intended meaning in reference to the reference system.

Presenting the details of an XML-based specification, like **`fisdMessage`**, requires the use of various terms unique to that genre. These terms are described in relation to the specification but also to XML in general.

1.2.1. Sender

A *sender* is any system that generates and provides content documents regardless of the industry role. Note that processing within a system may include subsystems that are *senders* as well as *recipients*. **`fisdMessage`** defines a protocol between a specific *sender* and *recipient* which is extendible to a large system with many applications. A *sender* may also be called a “provider”.

1.2.2. Recipient

A *recipient* is any system that, regardless of industry role, receives or processes content documents. *Recipients* of **`fisdMessage`** encoded documents will validate and parse the *instances* updating local copies of the *image*. *Recipient* systems may process documents and then provide that data to subordinate *recipients* in the same format thus fulfilling the role of *sender* as well. A *recipient* may also be called a “consumer”.

1.2.3. Directory

A *directory* is the listing of all *templates* and *enumerations* used in the *sender’s* data framework. The *directory* provides a checklist, including descriptions and other parameters, of those supporting documents that are necessary for the *recipient* to process all of the data available from the *sender*.

1.2.4. Enumeration

An *enumeration* is a list of terms that are mapped to integral values (0, 1, 2, etc.). Each *enumeration* has a unique URI (Universal Resource Identifier) which is provided as an entry in the *directory*. In MDDL, these are Controlled Vocabularies that contain abbreviations, with definitions, that are valid representations of the content for the corresponding data field. As an example, one controlled vocabulary for country code is identified by the URI http://www.mddl.org/ext/scheme/iso3166-alpha-2.xml which contains the ISO 3166 codes and country names. In the list at the above URL, the code “AQ” is defined as “ANTARCTICA” and is the 3rd item in the default list; Instead of transmitting the string “AQ” to represent the country Antarctica, the integer value 2 could be transmitted - generally at a significant bandwidth savings.

1.2.5. Template

A *template* defines the format of an XML instance document. Specifically, the *template* defines which portions of an instance document are static (the markup and non-changing data) with relationship to those pieces that change with each unique instantiation of the *template*, or over time. The *template* defines parameters about the dynamic elements that can be used to efficiently compress and distribute the *instance*.

The field definitions are the byproduct of the *template* and are used to deliver the content using **`fisdMessage`**. Each field identifies a unique piece of data that can be communicated independently of the framework. In many cases, a compound field will be defined which is comprised of many actual fields concatenated together. The compound field is a convenience notation for a collection of items that are updated together. A compound field may also specify some calculations that may be applied to the data provided before populating an actual field.

1.2.6. Instance

An *instance* is a specific instantiation of a *template* with a unique value for the key field. Generally, this term is used to refer to the “XML instance document” that is generated by applying the *image* data to the *template*. However, there is a subtle distinction in that a **`fisdMessage`** *instance* is not necessarily stored as an XML document.

1.2.7. Image

An *image* contains all of the content for a particular instrument as defined by the *template*. The *image* is the bits-and-bytes of data while the *template* outlines how this data should be represented in XML - the two combined generate an XML instance document.

1.2.8. Update

An *update* is a modification to one (or more) of the values in the *image*. The *update* contains the identifier of the field to be updated as cross-referenced in the *template*, and the implied processing rules on how to update the *image* (storage) of that field.

1.2.9. XML Specific Terms

XML is becoming a more prevalent standard within the computer and data industries and, as such, it has terminology that is unique to that standard.

1.2.9.1. Standard vs. Specification

XML is a metalanguage standard, developed and supported by the World Wide Web Consortium, intended to provide a flexible and adaptable scheme for information identification. MDDL is an XML specification – MDDL uses the XML standard to define the syntax for creating market data related documents.

1.2.9.2. Schema

A *schema*, when used to describe XML concepts, defines the collections of terms that can be used in a specific XML instance document. The *schema* (**`fisdMessage`** specifically uses W3C Schema) limits which elements and attributes can be placed within a document, what content they may contain, and how they are positioned within the document. Please read more at http://w3.org/TR/xmlschema-0/.

1.2.9.3. Instance Document

While *schemas* define the XML specification of interest (i.e. **`mddl`** and **`fisdMessage`**), the actual data interchanged between *sender* and *recipient* is formatted in an instance document based on the rules defined in the specification. An instance document is itself XML and the textual representation is dictated by the XML standard. An instance document is said to be *well-formed* XML if the elements and angle-brackets follow the rules of the XML standard and are correctly started, nested, and terminated *independent* of the governing specification.

An MDDL-based instance document is only valid if it is well-formed XML, validates against the correct version of the appropriate MDDL-based W3C schema, *and* is consistent with the intent of the written specification. In other words, there are implications of the content and format of an instance document that is above and beyond the capabilities of the schema to restrict.

1.3. Relationship to Other Work

MDDL and its derived specifications use several World Wide Web Consortium (W3C) recommendations including, but not limited to, XML 1.0, XML Namespaces, XLink, with specific references to XSLT. MDDL relies extensively on W3C XML Schema. Please see the W3C XML Website for references to XML related work.

Other XML specifications in the financial space have direct relationship to MDDL and integration is being pursued ala ISO TC68/SC4/WG11 (and other avenues). Targeted specifications for ongoing interoperability discussions include FIX (Financial Instrument eXchange Protocol), XBRL (eXtensible Business Reporting Language), FpML (Financial Products Markup Language, RIXML (Research Information Exchange Markup Language), and specifications of the IPTC (International Press Telecommunications Council) including NewsML and NITF (News Industry Text Format).

2. General Overview

The **`fisdMessage`** protocol is an open-standards based protocol for the exchange of rapidly changing, statically formed, XML documents in simple and complex systems. While specifically focused on the communication between a single provider and single consumer of market data, the specification is applicable to any large system (for example, the Generic Market Data System as described below). It is important to understand the reference framework for **`fisdMessage`** and the specific efficiencies that are possible within the closed system defined.

2.1. Protocol Applicability

The **`fisdMessage`** protocol is appropriate for realtime streaming of XML documents distributed using an "*image* with *updates*" paradigm. In this model, the *image* contains a complete snapshot of the data at a specific moment in time while the *updates* convey specific changes or replacement values to specific fields, or collections of fields, contained within the *image*.

The ability to define an *image* and apply *updates* is feasible when a relatively large number of instance documents can be said to have the same (ideally identical) structure but vary only in the data content of specific elements within the documents (which may included "undefined" values for some fields). By abstracting the commonality of those instance documents, a *template* can be defined which specifies the *format* of the document independent from the content.

Consider a Generic Market Data System which collects quote and trade market data from various exchanges and then merges the realtime information with static reference data from a database. In this system, it is likely that a particular exchange distributes approximately the same data for all like instruments traded on that exchange (for example, the Antarctica Exchange - AQE - reports the same basic data for all Antarctica common equities). Therefore, a *template* can be defined for "Antarctica Exchange Common Equities" data encoded in MDDL and thus distributed using the **`fisdMessage`** protocol. (At this point the reader is wondering whether there is an Antarctica Exchange - there isn't but the exchange processes every class of instrument known to MDDL and so it makes a good reference for the documentation.)

2.2. Reference Model

A reference system is appropriate to place the functionality of **`fisdMessage`** within a real world application. In this document, a Generic Market Data System will be described that illustrates the foundational capabilities of **`fisdMessage`** as well as some of the more complex interactions that are necessary when considering a large system.

The Generic Market Data System is a complex interconnection of nodes that combine to perform the primary function of the system – combining market data from multiple sources and presenting it to the end-user in a coherent and timely fashion. As diagrammed in Figure 2 below (discussed as the "Complex Multi-Node Model"), each node has a specific purpose within the system and communicates with its peer nodes on a simpler *sender* and *recipient* arrangement as diagrammed in Figure 1 below (discussed as the "Simple Two-Node Model").

2.2.1. Simple Two-Node Model

The Generic Market Data System can be decomposed into multiple interfaces where two separate components communicate, one as *sender* and one as *recipient*. While the interface may permit bidirectional interchange of information, the *sender* is defined as dominant provider of information and the other receives that data and processes it. In a bidirectional connection the recipient may send specific requests to the provider thus indicating the type of information requested. While it is the individual application that decides what data is required and how it is processed, the implied *sender*-*recipient* relationship governed by the **`fisdMessage`** protocol places specific requirements on each of the participants to guarantee complete and accurate exchange of data.

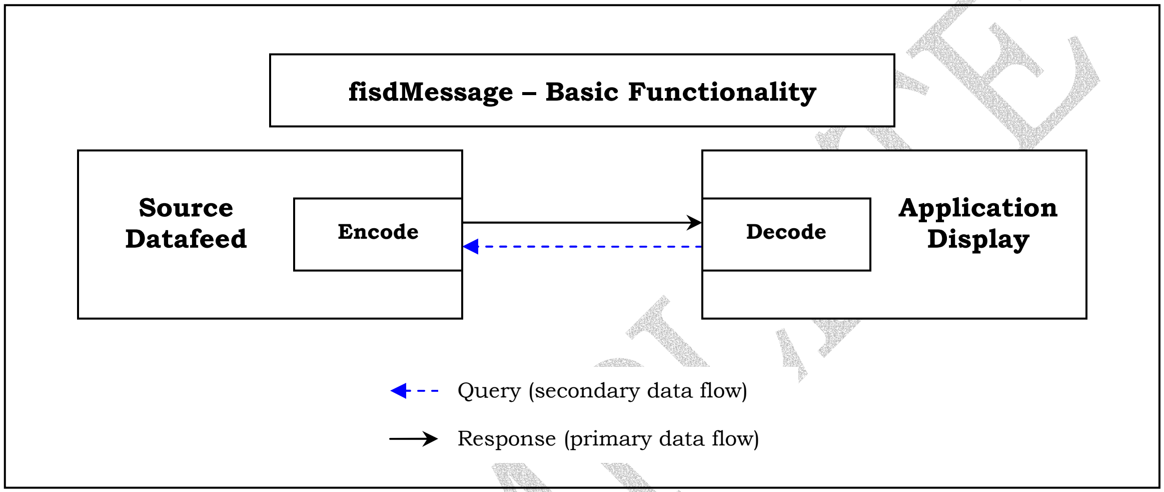
fisdMessage – Basic Functionality
Source Datafeed
Encode
Decode
Application Display
Query (secondary data flow)
Response (primary data flow)

Figure 1 - Simple Two-Node Reference System

2.2.2. Complex Multi-Node Model

The Generic Market Data System, diagrammed in Figure 2 below, includes data sources (Datafeeds and Databases), one or more Processing Engines, a System Headend that serves as the top of the distribution network, various Relay Nodes that act as cache and fanout units for the distribution network, and end applications (either Display Applications or Output Datafeeds to another system). **`fisdMessage`** is appropriate for use between each of the components of the system, as discussed in the section “Simple Two-Node Model” above, to convey the content of statically formed XML documents in their entirety and for communicating changes to the content efficiently.

Market data in the Generic Market Data System nominally flows from left to right through the nodes diagrammed. The Processing Nodes receive all data distributed from the System Headend and perform value added functions contributing the results back into the distribution network. While the individual end applications may enable users to contribute content, this is realized as a separate communication from the end-user application whereby it is viewed as a data source to the system.

The Datafeeds bring raw “realtime” data into the system. Each datafeed acquires the data from the specific source, transforms it into the Generic Market Data System representation of that content (ie. maps it into the appropriate *template* for that source), and distributes the resultant *images* and *updates* to the System Headend.

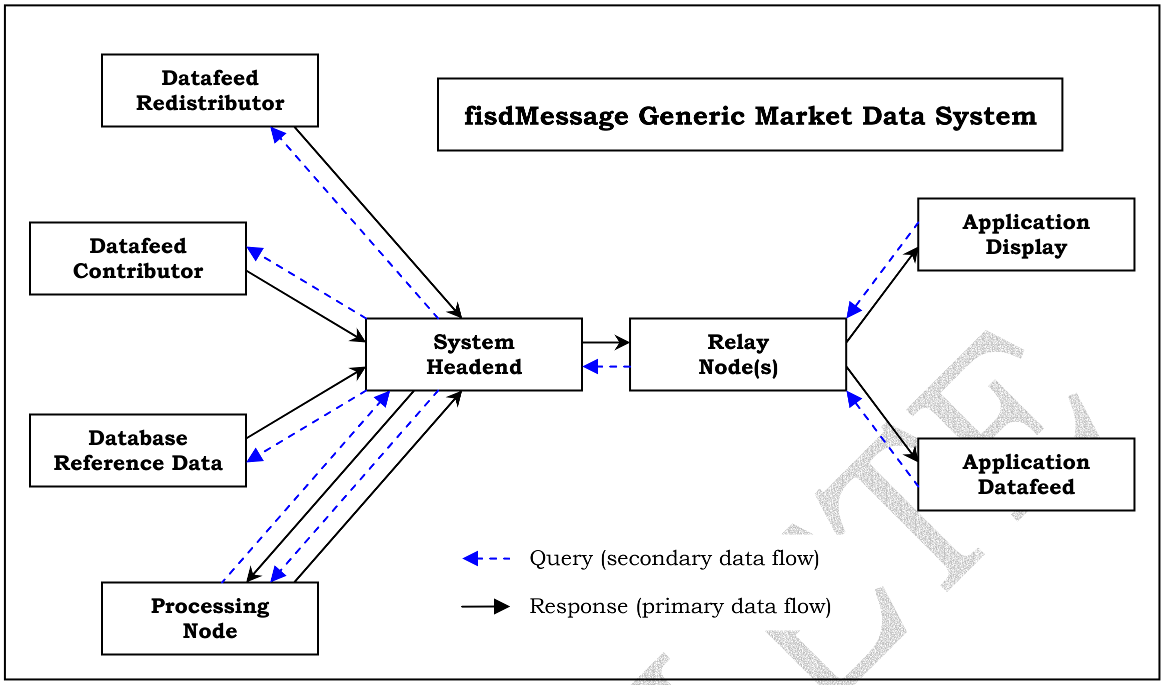

Figure 2 - The Generic Market Data System

The Databases contain relatively static data (static meaning the data doesn’t change often during a day or comes from an overnight download) that needs to be merged with the realtime content of the datafeeds. While the datafeeds may bring in the latest quote and trade information for particular instruments, the databases contain the historical analysis and reference data necessary - for example the country of registry, the exchanges on which the instrument trades, the dividend history, or even the previous day’s trading analytics.

The Processing Engine provides the bulk of the content value-add produced by the system. In an exchange, this would be the main trading system while in a vendor this might calculate analytic values or create historic data stores. Similarly, a market data user firm may perform its own analysis and perform portfolio calculations and risk analysis. The Processing Engine may take in any and all market data available and massage it, create new values (a “Calculation Engine”), or generate alerts based on user defined thresholds (an “Alert Engine”). A Processing Engine is unique in that it accepts content messages from a System Headend and may send *different* content messages back to the System Headend over the same connection.

The System Headend is the main focal point of the market data system through which all content must pass before it is distributed to the end users. While this may be a collection of computers performing multiple tasks, the purpose of the System Headend is to hold the definitive list of *templates* and *enumerations* (as well as the latest *image* of each *instance*) that are valid within this closed system. The System Headend acts as a concentrator of content from the various sources when a *template* requires data from more than one source - the System Headend merges partial *image updates* from each of the contributing Datafeeds, Databases, and Processing Engines to create a complete view of a particular *instance*.

The Relay Node is a “cache and fan-out” node for the distribution system. The Relay Node connects “upstream” to another Relay Node or the System Headend and accepts all updates from the upstream node. The Relay Node accepts "downstream" connections from other Relay Nodes or various applications and forwards (ie. copies) all content received from upstream to the downstream connections (perhaps on a selective by-interest basis). If the Relay Node performs caching functions, it may be able to service requests from downstream nodes from its stores.

The Display application is responsible for accepting the market data and displaying it in a form that the human user can understand. In most interactive scenarios, the Display application will accept input from the user and indicate to the upstream distribution system which information is of interest to the user.

The Output Datafeed is the output from the system that is analogous to the Datafeed source described above. The Output Datafeed translates the market data of this closed system into the format of the next - whether it is MDDL based or some legacy format. Note that the "Output Datafeed" of one system may, in fact, be the "Redistributor Datafeed" source of the next system.

2.2.3. Additional Extensions to the Reference Model

A production market data system, when compared to the Generic Market Data System, will have two significant additional features – 1) the number of nodes involved and 2) redundant sources and connectivity. A third consideration involves the monitoring, configuration, and control of the system.

2.2.3.1. Large Number of Nodes in System

A full production system comprises many Datafeeds, Databases, Processing Engines, and a significant distribution network serving many end applications. The total system may include anywhere from a few nodes to thousands of interconnecting computers distributed across the globe. The **fisdMessage** protocol is independent of the size of the system – it is the engineering applied to the various applications that ensure that the desired scalability can be achieved. The functionality implicit in the correct processing of the **fisdMessage** protocol is consistent with large scale implementations.

2.2.3.2. Redundant Processing and Delivery

A production system will contain redundant processing and distribution nodes to ensure delivery of information to the end-user in the event of telecommunications or hardware failure. The **fisdMessage** Version 1.0-beta protocol does not include any explicit support for redundant processing or delivery as it is envisioned that the principal applications (the System Headend, a specific Processing Node, a Relay Node, and end-user applications) would be designed to provide this required functionality. Should this assumption prove incorrect, the specification will be enhanced to include the necessary support.

2.2.3.3. Monitoring, Configuration, and Control

The realities of operational environments require (potentially remote) monitoring, configuration, and control. Monitoring involves two components – the analysis of the bandwidth and throughput of the data, and status of the health of the system. Configuration includes the parameters of the individual node and the interconnection pattern of the nodes within the system and is a function of the individual implementation. Control is exercised to bring nodes in and out of the system, and to initiate processing or reconfiguration.

Analysis of the operational system is an application with respect to **fisdMessage** Version 1.0-beta. In this model, individual nodes are responsible for collecting the necessary statistics and reporting it on request. Monitoring the health of the computer systems, individual processes, and network infrastructure is outside the scope of **fisdMessage** and should be incorporated into the system and application design.

While **fisdMessage** does contain some configuration and control features, it is recognized that Version 1.0-beta of the specification may not include all necessary functionality. Careful design of the *templates* and corresponding applications can accommodate all required parameters and processing but the protocol specification may require extension should standardization be necessary.

2.3. Connection Scenarios

The **fisdMessage** protocol is expressly for the purpose of exchanging XML documents efficiently between two nodes in a binary system. The protocol specifies the packets that are exchanged between the two nodes but not the underlying connectivity model. **fisdMessage** is consistent with a full interactive communication circuit as well as broadcast modes of operation. Note that the protocol does not specify the network layer connectivity but only the message payload transmitted and/or exchanged.

2.3.1. Selective Interactive Connectivity

In a fully interactive mode, all nodes will initiate connection to the System Headend, or "upstream" Relay Node within the distribution network. The requesting nodes must authenticate and fulfill the session validation requirements specified by the System Headend. Downstream nodes may then request content from the System Headend via the distribution network (which may involve simply asking for all content available). The requested content will then be returned back to the downstream node. In this model, the only content delivered to a downstream node was originally requested by that node (in addition to compulsory session and administrative messages) and for which the node is entitled to receive the data.

2.3.2. Broadcast Mode

In many scenarios it is not possible for downstream nodes to connect upstream such that the upstream node broadcasts all data and updates (for example, using multicast or satellite transmission). In this scenario, the recipient nodes must be preconfigured with appropriate authentication credentials and encryption keys. The recipient application must still process all administrative and content messages but need not generate, nor process, session establishment messages. The recipient application is responsible, however, for validating access to the delivered data based on entitlements.

2.3.3. Broadcast Mode (with Slow Backchannel)

The backchannel can be used for session establishment and/or marking interest or requesting content (other than primary distribution content). This scenario behaves like the broadcast mode configuration in that all peer nodes connected to the same distribution node will receive all content broadcast as a result of the requests from individual downstream nodes. However, it behaves as an interactive node for the purposes of establishing authentication and determining the current encryption as well as permitting downstream users to request content to be broadcast.

2.4. *Image* and *Update* Methodology

The **fisdMessage** protocol is predicated on the fact that the data may be modeled as a snapshot (or *image*) of current content with messages containing each change (or *update*) to be applied to the *image*. The structure of the *image* is static, as defined in the *template*, and each fielded value has a specific location (or locations) within the document created when the values are merged with the *template* to create the XML representation of the specific *instance*.

Although the protocol defines how *updates* are to be applied to an *image*, this is for the purposes of keeping an up-to-date copy of the *instance* for caching or display purposes. Receiving applications may choose to keep the initial *image* independent of the *updates* (or in addition to the *updates*) thus providing a "historical" list of changes to the *image*. The clever crafting of the *template* facilitates this application requirement.

2.5. Document Template and Instance

The **fisdMessage** protocol makes use of a *template* to describe the content distributed within the system – for example, two different instruments from a particular stock exchange are considered to be *instances* of the single *template* representing content from that stock exchange. Each instance maps to the template but may contain different values for each fielded value within the *template* – and *must* contain different values for the field identified as the key for that *template*.

2.5.1. Specifying a *Template*

An **fisdMessage** *template* is an XML document that defines the structure of the content document which represents the *image* for a particular source. The required **fisdMessage** elements defining the parameters of each field are substituted for the real content of the document thus the *template* will not validate against the governing content document schema. However, the additional markup will validate against the governing **fisdMessage** schema.

There are predefined *templates* that are foundational to the **fisdMessage** protocol which represent the structure of the individual protocol messages. The same rules that relate *images* to *templates* and thus specific fielded content, also apply to messages that are transmitted as part of the protocol. The structure and content of the protocol messages are not updated in the same way as the primary content *instances*. The protocol messages are defined in *templates* that explicitly must not be modified.

The *dictionary* lists all *templates* that are valid within the system. Adding a new *template* is as simple as modifying the *dictionary* and having it distributed throughout the system. Note: it is advised that the *template* be broadcast to all recipients before redistributing the *directory* because sending a *directory* with a reference to an unknown *template* (or *enumeration*) will force all recipients to send a query for the template.

2.5.2. Specifying a Field

A *template* is defined to isolate the changing content from the static content in an XML document. The individual pieces of changing content are defined as fields that may be referenced as tokenized values in update communications. Each field is delimited in the *template* and will contain the following amplifying properties:

- Field Moniker - used by applications
- Field Name - formal field name for display
- Field Number - as encoded in the protocol

- Key Field - Boolean indicating if this field is the instance unique identifier
- Index Bits - number of bits used to encode the Key Field
- Source - data source for use by System Headend for merging content
- Update Frequency - clue to relative frequence of changes to field's values
- Atomic Type - integer, string, float, Boolean, enumeration, etc.
- Range - range of values for the field (numerical types)
- Enumeration Scheme - scheme governing legal values (if applicable)
- Formula - to be applied to value placed in field when updating
- Entitlement Mnemonic Fomula - expression governing access to the field
- Entitlement Value - required to access this field
- Compressions - for this field used in processing
- Subordinate Fields - children (and formulas) of this field if it is a compound field
- Coordinating Fields - other fields that must be transmitted simultaneously

Field specifics are defined in more detail in the appendix.

2.5.3. Applicability to an *Instance*

An *instance* is a specific *image*, identified by a unique key field, where the fields that are the content for that *image* are governed by the corresponding *template*. Every field identified within the *template* is contained within the *instance* - although some fields may not have values and thus are "undefined". The *recipient* is responsible for keeping the current value of each field, as published by the distribution system, so that it may be used to map the *instance* to the *template* thus yielding an XML instance document representing the content – at the moment of creation – of the *image*.

2.5.4. Extracting a Complete XML Document

The process of merging an *image* with its governing *template* creates an instance XML document. The resulting document will validate against the appropriate schema – for example, an *instance* of a protocol message *template* will validate against the **`fisdMessage`** schema while an *instance* of a *template* for MDDL content will validate against the appropriate MDDL schema.

Each field defined within the *template* indicates the atomic type of the field and the rules for adapting the fielded value to the XML representation. This process is identical for every *template* and the rules are consistent for all fields thus the process is easily automated and should require very little processing overhead. The algorithm required for this conversion is part of the **`fisdMessage`** protocol specification and is discussed within the appendix.

Note that within the Reference System, an actual XML document derived from an *instance* is expected to only be necessary at interface points out of the system – within the system most processing can be performed using the tokenized content passed via **`fisdMessage`**.

2.5.5. Use of an *Enumeration*

Many XML-based specifications make use of *enumerations* – lists of string values mapped to integral numbers. The integral value of the appropriate item from the *enumeration*, not the string itself, is passed within the system.

The *enumeration* is distributed, and is required to be stored by the *recipient*, so that it may be referenced whenever the string value mapping to the integral value is required.

Implementers should be aware that the *enumeration* used within one system is specific to that use because the provider may have reordered the *enumeration* to facilitate compression or improve other performance factors.

The value of the scheme attribute can be a field itself as the *enumeration* may be changed (for any number of reasons). Note that changing the value of a scheme attribute field is a relatively costly operation because of the length of the string (it is a URI that typically looks like a URL). Care should be taken to define *templates* so values of scheme attributes are changed infrequently (or not at all).

Changing the mapping of an *enumeration* should be avoided as well. When mappings are changed, every *instance* derived from every *template* using the *enumeration* must be scanned and values substituted (which is potentially a very expensive operation). By definition, an *enumeration* change invalidates every *instance* derived from every affected *template*. It is recommended that *enumeration* changes that require remapping should be postponed until system load is low or until affected *instances* can be republished. The **fisdMessage** specification does provide facilities to dynamically *extend* an *enumeration* (thus adding values to it) efficiently. While these additions may not provide optimally ordered schemes, the dynamic extension will allow delivery of new content until a complete remapping can be executed (perhaps in an "overnight" update).

2.6. How a *sender* Behaves

A *sender* is responsible for specifying the framework under which *updates* to *instances* are delivered. By defining the *enumerations* and *templates* that define the XML documents, the *sender* (effectively) automatically configures the delivery system. Once the framework is delivered (generally via Administrative messages), the *sender* must deliver AMsg Timestamp messages (heartbeats) and compressed or uncompressed *updates* consistent with the configuration.

Assuming a valid network channel is available (either through an interactive session or over some broadcast medium), the *sender* should distribute the following Administrative messages to configure the framework *in the following recommended order*:

- AMsg Directory Response - the digest of the framework
- AMsg Enumeration Response - for all *enumerations*
- AMsg Template Response - for all *templates*
- AMsg Directory Response - the digest of *provider* content
- AMsg Timestamp - per frequency established in the *directory*
- Content messages as appropriate…

It is not required to send all *enumerations* before all *templates* but it is mandatory that all *enumerations* required for a *template* be delivered before the *template* is transmitted. This is necessary because *recipients* must determine the number of entries in an *enumeration* to know how to interpret field references to them.

The *directory* should contain all protocol message *templates* and *enumerations* as well. It should not be assumed that all *recipients* have the exact same release of software and framework as the *sender*. The **fisdMessage** protocol is intended to be flexible enough that all relevant information can be sent from the *sender* allowing the *recipient* to automatically configure to receive content (given a proper receiving application).

A *sender* must be prepared to send all *enumerations* - those used within the **fisdMessage** framework as well as those in the content defined by the *templates*. In an interactive feed, the *recipient* will check the entries in the *directory* and ask for those that it does not have or have been determined to be out of date or different. In a broadcast scenario, while it is convenient to assume that all *recipients* are configured from an initial installation, it is prudent to broadcast the *directory* and all *enumerations* and *templates* as frequently as feasible to guarantee accurate configuration on an ongoing basis.

The AMsg Timestamp message is not required for proper operation of the protocol but content that uses time sensitive information relies on a "heartbeat" to verify upstream connectivity (particularly in broadcast mode). Further, content that makes reference to the current time can be more efficiently compressed the more often a heartbeat is transmitted. The *sender* should weigh the ability to generate reliable heartbeat messages against the compression benefit derived (if necessary) as well as the ability for a *recipient* to notice a circuit is down. As the beginning point of a discussion - consider sending a timestamp once every 500 milliseconds (with potential delays stretching the delta up to 1000 milliseconds) in the system design. If time resolution is required to the hundredths of seconds (10^{-2}), then a timestamp delta can be encoded in 7 bits - a significant savings over the 64 bit absolute time encoding. Remember that the heartbeat is not so much about dictating the accurate current time (although it is useful for that function) but in giving a known reference on which to base deltas within messages as well as assurance the system is operational.

2.7. How a *recipient* Behaves

A *recipient* should be designed to configure itself automatically around the framework and content distributed by the *sender*. The *recipient* should make as few assumptions about *directory*, *enumeration*, and *template* content as possible - instead, the configuration information as dictated by the *sender* should be used. While the receiving application will most likely configure itself to process particular data (for example, a particular field within a particular template), it should not hardcode its expectations of the framework - instead abstract the references to particular data into a configuration file or other mapping facility using the field mnemonic for references.

The discussion on "How a *sender* Behaves" above has significance to the *recipient* as well. The guidelines applicable to the *sender* also outline what a *recipient* can expect - but nothing should be assumed. Given a sufficiently flexible receiving application, there are two concerns for *recipients*: 1) guidelines for making requests upstream (if an upstream connection exists), and 2) what to do in the case of errors or inconsistencies.

The initial AMsg Directory Response (or the same content embedded in the SMsg Authentication Response) provides crucial configuration information and outlines which framework pieces are required to operate. The receiving application should accept all Administrative messages and store the *directory*, *enumerations*, and *templates* for comparison even if content defined by some *templates* will be ignored by the application. If the *directory* specifies framework items that the *recipient* does not have then they should be requested and stored. Keeping the framework consistent assures the *recipient* can properly decode all messages and ignore the undesired content. If the *sender* adds new content to *templates* of interest that reference framework elements that are not available then the *recipient* may lose the ability to interpet the data.

Error processing is always a concern for data and time sensitve applications. The problems can usually be categorized as a) failure in delivery of messages and b) improper content. The former is a reality of even the most reliable communication media and must be accepted. A receiving application may re-request data from upstream nodes (if connectivity is available) but such requests should be made with discretion or else the *recipient* node may inadvertently cause system overload.

Messages containing content that is improperly encoded must be addressed with the provider. At the time of this writing, there is no certification program in place for the **`fisdMessage`** protocol but one is being devised to provide some third-party verification of capabilities. If bad messages are suspected, document specifically the most recent AMsg Timestamp content as well as the *enumerations*, *templates*, *images*, and all intervening *updates* so that the provider may correctly diagnose the problem.

Although it is encouraged that the *recipient* should not assume anything about the messaging protocol messages, *enumerations*, or *templates*, the following assumptions must be made (as of version 1.0-beta):

- The enumeration “requestStatus” values. For fisdMessage 1.0-beta, this enumeration is fixed at 256 entries and requires exactly 1 byte.
- The enumeration “encryptionMethod” values. For fisdMessage 1.0-beta, this enumeration is fixed at 256 entries and requires exactly 1 byte.
- The enumeration “entryType” values. For fisdMessage 1.0-beta, this enumeration is fixed at 256 entries and requires exactly 1 byte.
- The enumeration “enumerationType” values. For fisdMessage 1.0-beta, this enumeration is fixed at 256 entries and requires exactly 1 byte.
- The compression “zlib” for whole message compression is the default.
- The default compression for integer fields is “i8.0”.
- A timestamp is exactly 8 bytes (5 bytes for seconds since the Epoch followed by 3 bytes for the number of microseconds in the current second).
- The enumeration “messageType” is encoded with 6 bits.
- Every message has the same 5 fields at the beginning (see later sections).

3. The fisdMessage Schema

The **fisdMessage** schema defines the XML markup used to specify field and *template* parameters as well as the XML format of the **fisdMessage** framework messages. Note that the specification uses the schema to govern the *templates* for the protocol messages in the same way that *templates* are defined for the system data content.

There are four major functions to the schema: 1) governing the protocol messages, 2) outlining the directory of content, 3) portraying an enumeration, and 4) defining a template for content of interest.

3.1. Applied to Messages

Each of the messages defined in this protocol is described by a special *template* just as the content is defined. While it is more of an intellectual exercise, the mapping of protocol message *templates* to the on-wire protocol gives insight into how a *template* can be defined for content of interest. Please note that currently the specification does not permit any flexibility in altering protocol messages (in other words, a *recipient* could "hard-code" the processing of protocol messages for a given version of the protocol). Future releases of the specification will rely heavily on the use of *templates* for the protocol messages so that custom messages (especially compressions) can be defined.

3.2. Applied to the *Directory*

The *directory* is the core of the framework of **fisdMessage** and conveys needed configuration information to the downstream applications. The specification provides the core (minimum necessary) *directory* to which the *sender* may add *enumerations* and *templates*. The *sender* must not modify the core elements defined in the *directory*.

3.3. Applied to an *Enumeration*

The representation of an *enumeration* is fairly straightforward with **fisdMessage** and there are few variables to the specification. The protocol message *enumerations* may be used as examples of this encoding, as can the examples provided in the appendix.

While there are a number of *enumerations* that are specified by the protocol, a *sender* may create many additional *enumerations* to convey the data of interest or facilitate compression (for example, some of the large *enumerations* like the country codes may be broken into smaller *enumerations* based on regions or a subset of the currency codes may be extracted because the *sender* only deals in certain currencies).

3.4. Applied to a *Template*

The protocol message *templates* may be used as an example for some of the functionality available. In addition, a complete set of examples is available in the appendix (based on MDDL).

The creation of *templates* is the most complicated aspect of configuring **fisdMessage** and care must be exercised to define *templates* that are meaningful and functional. Each field defined within a *template* has many parameters and all should be specified explicitly. Further, the creation of virtual fields, particularly permutations, provides for the greatest flexibility, and thus efficiency, of bandwidth usage.

4. Packet Characteristics

Each message is governed by a *template* defined by the **`fisdMessage`** schema. Sections 5-7 describe the detailed contents of packets on-the-wire as derived from the *template* provided with this release of the specification. Section 8 presents examples of templates and their corresponding "on-wire" formats. It may be adisable to read Section 8 both before *and* after reviewing sections 5, 6, and 7.

Changing the order of contents in any of the *templates* may affect some field lengths or values – for example, adding new **`templates`** may affect the size of the **`templateIdentifier`** field. Fields that are sensitive to these changes are indicated by an *italicized* "Name" and "Size" in the message description sections. The *recipient* should programmatically determine these values and field sizes from configuration information as provided by the *sender*.

The **`fisdMessage`** protocol uses network standard big-endian convention – all integer and floating values requiring more than one byte are delivered with the most significant bits of the most significant byte first. All character values are sent as null-terminated UTF-8 strings, with the first byte first, unless a specific byte count is otherwise defined. The *recipient* application must handle conversion of all values to local byte ordering and form, if necessary.

4.1. Message "Header"

All **`fisdMessage`** based messages are derived directly from the governing XML document (the *template*) that is conformant to the applicable **`fisdMessage`** W3C schema. All *templates* are defined, in this version of the specification, with the same fields at the beginning of the message to facilitate processing by the *recipient*. This section illustrates how the messages are documented and highlights the fields that are common to all documents which comprise the **`header`** as described in the table below.

Column definitions

Field Mnemonic: Denotes the abbreviation of the field used in the *template* defining the message and has a unique meaning across all *templates* defined by the provider. In all cases, the field mnemonic should be prefaced with "/fisdMessage/" to construct the full field mnemonic. e.g, "messageType" is really "/fisdMessage/messageType".

Name: The full textual name of the field. An italicized name is indicative of a field with a variable size dependent on content within the message or configuration framework.

Size: The number of bits or bytes required to represent the UNCOMPRESSED field. The exact length of each field is defined by the *template* for that message and the applicable controlled vocabulary (for *enumerations*), the length of a string, or applicable compressions. The size of the field will be in italicized print if it is variable (string variables are implicitly variable length). Note that this version of the specification may define constant sized fields that may be variable in future versions.

Type: The basic type of the field (listed here for convenience) as defined by the *template* and governing schema. The type provides hints about interpreting the value of the field and memory requirements for storing the value.

Value/Notes: Specific values are provided for those fields providing identifying features (like the `messageType`). Comments may require additional discussion which will be included following the table.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed

Table 1 - fisdMessage 1.0-beta header

Special field processing considerations

messageType: The unique identifier for this type of message as defined by the messageType *enumeration*. The value of the messageType may change between versions of `fisdMessage` but will always be mapped to the governing *enumeration*.

encrypted: Indicates if the message content, after the (2 or more bytes EXCLUDING the fullLength) header is encrypted as negotiated (or specified) for the communications channel. Initial Session Establishment messages MUST NOT be encrypted until the encryption protocol has been specified. All other messages will be encrypted in identical fashion as dictated by the provider. Encryption keys and protocols are negotiated during Session Establishment or prearranged via separate communications as necessary. The unencrypted packet must be null padded to the appropriate length for the encryption protocol in effect. Note that encryption is applied to a compressed message packet immediately before message delivery. Therefore, the recipient must perform decryption before message-level decompression can take place (if applicable).

compressed: Indicates if the message content, after the header, is compressed using the message-level compression algorithm defined in the *directory* (the default is zlib). If compression is used, the fullLength field defines the uncompressed message payload EXCLUDING the header.

messageLength: The total length of the message EXCLUDING the first 2 bytes of the header. The messageLength field is used by the recipient to preallocate buffers necessary to read the content from the network. The messageLength field is an integer that is compressed using the “i8.0” compression scheme – there will be one byte containing 7 bits of the message length - if the upper bit is set (for any of the bytes) then there are additional bytes following, containing 7 bits each, that when concatenated provide the message length. The last byte of the length will have the upper bit clear. NOTE: The messageLength SHOULD NOT EXCEED 64kB-48-20-2 bytes. Thus, an fisdMessage packet can be accommodated by TCP with IPv4 or IPv6 (allowing for network specific fragmentation and reassembly). To be reviewed…

fullLength: The total length o the pre-compression message payload EXCLUDING the full header (which may be 3 or more bytes). The fullLength field is used by the recipient to preallocate buffers necessary to perform zlib decompression. This field will be absent if zlib compression of the message payload has not been applied. The fullLength field is an integer that is compressed using “i8.0” compression (unless redefined in the *directory*).

4.2. Query Requests and Responses

The nature of an interactive feed requires that the downstream application (or “user”) be permitted to make requests upstream for specific types of content. The **`fisdMessage`** protocol defines four query messages and a single response message to allow the user to identify the data of interest. Three of the query messages are specific to the “image with updates” model that **`fisdMessage`** is based on. The fourth (“SMsg Query for Other Content”) is a general vehicle for ad-hoc queries. The single response message (“CMsg Query Response”) conveys the status for the first three types of queries and the actual response document for the fourth.

SMsg Query for Instance: Specifies a *template* and makes a request for a specific *instance* of that *template*. The request signals the *sender* that the *recipient* is interested in all content for that *template* as applied to the requested *instance*, including the *image* and all *updates*.

SMsg Query to Unmark: Indicates that the *recipient* is no longer interested in receiving the indicated content - send no more *updates* for that *instance*.

SMsg Query for All Data: Allows the *receipient* to request “All Data that is Available” or “All Data that All Others have Requested”. The former indicates that the *sender* should transmit all *instances* that are available in the *sender’s* databases. The latter informs the *sender* that the *recipient* would like a copy of all streaming content sent to other *recipients*.

SMsg Query for Other Content: A general ad-hoc query mechanism that results in a specific XML instance document being returned with the requested content.

CMsg Query Response: Generic response mechanism with status information for each request - may include actual content message in the case of “SMsg Query for Other Content”.

5. Session Establishment Messages

Session Establishment messages are used on fully or partially interactive feeds to authenticate and configure a new recipient of the data feed into the system. The Session Establishment messages allow negotiation of communication encryption parameters and permit the recipient to request configuration information that it may not already have. In addition, Session Establishment messages allow a downstream application to request certain framework or content that it does not have.

NOTE: Session negotiation has not been completed. Most notably missing are requisite three-way handshakes and specifics on type of authentication and encryption to be used. It is intended that a subscriber may interactively negotiate AES/DES, RSA, and/or MD5 based criteria for session authentication and establishment. The provider will dictate the form of the encryption used for administrative and content messages. Subsequent encryption may employ any combination of **`fisdMessage`** defined encryptions deemed appropriate by the provider that can be communicated via the session establishment mechanism.

5.1. SMsg Service Notification

URI: http://www.fisd.net/fisdMessage/SMsgServiceNotification/

Direction: From sender
Tag line: "I am ready to serve"
Description: The Service Notification message is sent from the provider to announce that the service is ready. This message is sent when the system has started, has been reset, or at periodic intervals. An initial timestamp is included.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	1 (0x01)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	"x.y-z"
servicesLength	Number of Services	*n Bytes*	integer	"i8.0" compression
…Repeating Entries… 1 set per "servicesLength"				
serviceRequested	Requested Service	*n Bytes*	string	"fisdMessage x.y-z"
…End Repeating Entries… for "servicesLength"				

Table 2 - SMsg Service Notification

Special field processing considerations

encyrpted: This message must not be encrypted (as the encryption method has not been negotiated).

servicesLength: Compressed with the default "i8.0" compression.

Discussion

After initial network connection, recipients should not attempt to interact with the service until this message or a heartbeat (AMsg Timestamp) message has been received.

5.2. SMsg Service Request

URI: http://www.fisd.net/fisdMessage/SMsgServiceRequest/

Direction: From recipient
Tag line: “I want to connect and here is my user name”
Description: The Service Request message is sent from the consumer (the recipient) to the provider (sender) requesting connectivity to the distribution network or system specifying the type of encryption to use.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	2 (0x02)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	“x.y-z”
requestKey	Request Key	*n Bytes*	string	
requestStatus	Request Status	*n bits*	enum.	per enumeration
encryptionMethod	Encryption Method	*n bits*	enum.	per enumeration
encryptionKey	Encryption Key	*n Bytes*	string	
encryptionValue	Encryption Value	*n Bytes*	integer	“i8.0” ocmpression
serviceRequested	Requested Service	*n Bytes*	string	“fisdMessage x.y-z”
username	User Name	*n Bytes*	string	

Table 3 - SMsg Service Request

Special field processing considerations

encyrpted: This message must not be encrypted (as the encryption method has not been negotiated).

encyrptionMethod: Specifies the encryption method for subsequent interactions. The implications of this selection are covered in the appendix. The encyrptionMethod enumeration must be preconfigured at the recipient.

requestKey: Each request upstream should be given a unique requestKey. The requestKey, with an appropriate status, will be echoed in subsequent responses.

Discussion

In conjunction with the SMsg Service Response message, this message is used to coordinate encryption for subsequent messages.

5.3. SMsg Service Response

URI: http://www.fisd.net/fisdMessage/SMsgServiceResponse/

Direction: From sender
Tag line: "Okay, now authenticate in this way over specified channel"
Description: The Service Response message is sent from the sender acknowledging the Service Request message. The sender will provide the encryption key to be used on all future requests and may specify an alternate channel to make subsequent requests.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	3 (0x03)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	"x.y-z"
requestKey	Request Key	*n Bytes*	string	from ServiceRequest
requestStatus	Request Status	*n bits*	enum.	per enumeration
encryptionMethod	Encryption Method	*n bits*	enum.	per enumeration
encryptionKey	Encryption Key	*n Bytes*	string	
encryptionValue	Encryption Value	*n Bytes*	integer	"i8.0" compression
serviceRequested	Requested Service	*n Bytes*	string	"fisdMessage x.y-z"
channelIdentifier	Channel Identifier	*n Bytes*	string	

Table 4 - SMsg Service Response

Special field processing considerations

requestStatus: The requestStatus enumeration must be preconfigured at the recipient.

encyrptionMethod: Specifies the encryption method for subsequent interactions. The implications of this selection are covered in the appendix. The encyrptionMethod enumeration must be preconfigured at the recipient.

channelIdentifier: In some instances, the sender may require that the recipient continue the communication on another channel. The recipient should disconnect from the current channel and resume the conversation on the specified channel.

Discussion

In conjunction with the SMsg Service Request message, this message is used to coordinate encryption for subsequent messages.

5.4. SMsg Authentication Request

URI: http://www.fisd.net/fisdMessage/SMsgAuthenticationRequest/

Direction: From recipient
Tag line: "Please authenticate me"
Description: The Authentication Request is sent by the recipient requesting authentication and access.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	4 (0x04)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	"x.y-z"
requestKey	Request Key	*n Bytes*	string	new value…
username	User Name	*n Bytes*	string	
password	Password	*n Bytes*	string	

Table 5 - SMsg Authentication Request

Special field processing considerations

Discussion

This must be encrypted using the encryption scheme the encryption key from the Service Response.

5.5. SMsg Authentication Response (with Directory)

URI: http://www.fisd.net/fisdMessage/SMsgAuthenticationResponse/

Direction: From sender
Tag line: “You are authenticated, and this is what you need”
Description: The Authentication Response is sent from the sender acknowledging the previous Authentication Request and specifying the messages, templates, and enumerations that are required to fully process the framework of the datafeed.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	5 (0x05)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	“x.y-z”
requestKey	Request Key	*n Bytes*	string	new value…
requestStatus	Request Status	*n bits*	enum.	per enumeration
providerDirectory	Provider Directory	1 Byte	bool	should be false
compression	Compression Scheme	*n Bytes*	string	for subsequent msgs
compressionInt	Integer Compression	*n Bytes*	string	for integer counts
compressionTSD	Timestamp Delta Cmp.	*n Bytes*	string	for timestamp deltas
directoryLength	Directory Length	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per “directoryLength”				
entryType	Entry Type	*n bits*	enum.	per enumeration
entryNumber	Entry Number	*n bits*	integer	see compressionInt
entrySize	Entry Size	*n bits*	integer	see compressionInt
entryExpression	Entitlement Expression	*n Bytes*	string	
entryEntitlement	Entry Entitlement	4 Bytes	integer	
entryChecksum	Entry Checksum	2 Bytes	integer	
entryDate	Entry Date/Time	8 Bytes	time	of modification
entryURI	Entry URI	*n Bytes*	string	
entryName	Entry Name	*n Bytes*	string	
…End Repeating Entries… for “directoryLength”				

Table 6 - SMsg Authentication Response

Special field processing considerations

entryType: The entryType enumeration must be preconfigured at the recipient.

Discussion

This message defines the default compression scheme for integers and message-level compression. Subsequent messages may therefore use compressions other than zlib.

The provider will also send an “AMsg Directory Response” for provider content.

5.6. SMsg Entry Request

URI: http://www.fisd.net/fisdMessage/SMsgEntryRequest/

Direction: From recipient
Tag line: "I need this entry"
Description: A request for a required entry (as defined in the directory) that the recipient may not have (or may have an outdated version).

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	6 (0x06)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
entryURI	Entry URI	*n Bytes*	string	
entryType	Entry Type	*n bits*	enum.	per enumeration

Table 7 - SMsg Entry Request

Special field processing considerations

entryType: The entryType enumeration must be preconfigured at the recipient.

Discussion

The sender should deliver an AMsg Entry Response with the requested entry.

5.7. SMsg Session Notification

URI: http://www.fisd.net/fisdMessage/SMsgSessionNotification/

Direction: From recipient
Tag line: "I'm synched and ready to go"
Description: An indication that the recipient has received all framework items and is ready to accept content updates. Generally, this message is followed by some sort of Query message.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	7 (0x07)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…

Table 8 - SMsg Session Notification

<u>*Special field processing considerations*</u>

<u>*Discussion*</u>

5.8. SMsg Session Notification Response

URI: http://www.fisd.net/fisdMessage/SMsgSessionNotificationResponse/

Direction: From sender
Tag line: "Okay, here comes the data"
Description: An acknowledgement by the sender that content messages, including AMsg Timestamp "heartbeats" will begin to flow on this channel.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	8 (0x08)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
requestStatus	Request Status	*n bits*	enum.	per enumeration

Table 9 - SMsg Session Notification Response

<u>*Special field processing considerations*</u>

<u>*Discussion*</u>

This message may also be sent in response to the SMsg Session Termination Request indicating a successful termination of the requestor's session.

5.9. SMsg Session Session Termination Request

URI: http://www.fisd.net/fisdMessage/SMsgSessionSessionTerminationRequest/

Direction: From recipient
Tag line: “Please terminate my session”
Description: A request by the recipient to terminate the active session.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	9 (0x09)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
username	User Name	*n Bytes*	string	

Table 10 - SMsg Session Termination Request

Special field processing considerations

Discussion

The sender will return with the SMsg Session Notification Response but the recipient is free to abandon the connection once this message is delivered.

5.10. SMsg Query for Instance (Mark)

URI: http://www.fisd.net/fisdMessage/SMsgQueryInstance/

Direction: From recipient
Tag line: "I would like this instance of this template"
Description: A request for an instance of a specific template. The recipient is asking that the image for the instance, and all subsequent updates, be delivered. This effectively "marks" interest in particular content for streaming purposes.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	10 (0x0A)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
templateURI	Template URI	*n Bytes*	string	
queryLength	Query Length	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per "queryLength"				
keyFieldValue	Key Field Value	*n Bytes*	string	
…End Repeating Entries… for "queryLength"				

Table 11 - SMsg Query for Instance

Special field processing considerations

Discussion

The sender will respond with a CMsg Query Response message with the success status. If the query is successful, the sender will distribute a CMsg Image Update (or CMsg Fielded Image Update).

5.11. SMsg Query to Unmark

URI: http://www.fisd.net/fisdMessage/SMsgQueryUnmark/

Direction: From recipient
Tag line: "I am no longer interested in this instance of this template"
Description: This "unmarks" interest in particular content. Depending on the nature of the feed, the sender may be able to stop distributing updates of this instance thus conserving bandwidth.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	11 (0x0B)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
queryLength	Query Length	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per "queryLength"				
keyField	*Key Field*	*n bits*	enum.	per directory
…End Repeating Entries… for "queryLength"				

Table 12 - SMsg Query to Unmark

Special field processing considerations

Discussion

The sender will respond with a CMsg Query Response message with the success status. Note that the response is merely an acknowledgement of receipt of the message. In a fully selective feed, the sender will stop sending updates to the recipient.

5.12. SMsg Query for All Data

URI: http://www.fisd.net/fisdMessage/SMsgQueryAllData/

Direction: From recipient
Tag line: “I would like all content requested by anybody”
Description: A request for all instances of all templates - in effect all images and subsequent updates. The request may be qualified by a specific template of interest. The request may be further qualified with an implied “mark” of existing data.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	12 (0x0C)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
templateURI	Template URI	*n Bytes*	string	for qualification
markAll	Mark All Content	1 bit	bool	

Table 13 - SMsg Query for All Data

Special field processing considerations

templateURI: A null field indicates that all templates are of interest. Otherwise, the sender specifies which template is of interest thus implying all instances of that specific template should be delivered.

markAll: A Boolean differentiating the request for “All Data Available” and “All Data of Interest to other subscribers”. If true, the recipient is effectively marking all data that is available from the sender. If false, the recipient is requesting a duplicate of all information that is sent to other subscribers and does not wish to mark any content.

Discussion

The sender will respond with a CMsg Query Response indicating the status of the request.

The SMsg Query for All Data may be used by nodes that wish to receive all content that is of interest to other nodes (for monitoring or analysis) or all information that is available (for complete duplication of content). In either event, the request may put excessive demands on the communications circuit and should only be used when confident that sufficient bandwidth exists to send all of the data requested.

5.13. SMsg Query for Other Content

URI: http://www.fisd.net/fisdMessage/SMsgQueryOther/

Direction: From recipient
Tag line: "I would like this content retrieved and returned to me"
Description: A request for content that may not be governed by a template or the image and update paradigm of fisdMessage. The form of the request is governed by a template which the sender will route to the appropriate node for processing.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	13 (0x0D)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
templateURI	Template URI	*n Bytes*	string	for qualification
queryLength	Query Length	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per "queryLength"				
fullQuery	Full Query	*n Bytes*	string	
…End Repeating Entries… for "queryLength"				

Table 14 - SMsg Query for Other Content

Special field processing considerations

templateURI: The template that governs the form of the query. Only query templates specified by the provider can be accepted.

fullQuery: The query as constructed by the consumer. The form of this query is governed by the template and is serviced at the discretion of the provider. For example, in an MDDL based system, this query will be structured as mddlQuery.

Discussion

The sender will respond with a CMsg Query Response which include the status of the request and, if successful, will contain the XML document with the requested content.

6. Administrative Messages

The Administrative messages deliver the content framework of the *sender* to all downstream *recipients*. There are three basic types of messages:

- Status - specifically the AMsg Timestamp "heartbeat" message.
- Framework - the *directory*, *enumerations*, and *templates*
- Adjustment - to the framework to improve efficiency

In a fully interactive feed, the framework messages are returned as responses to specific requests (see Session Establishment messages). However, once the session is enabled, the interactive feed receives the same types of Adminstrative messages as would a broadcast connection.

The "heartbeat" message is transmitted by the *sender* on a regularly scheduled basis (perhaps twice a second) so that the *recipient* knows that the circuit is active and the *sender* can use the heartbeat as a reference for compression.

The framework messages are sent whenever it is necessary to synchronize the framework between the *sender* and *recipient* - perhaps twice a day (especially important on pure broadcast feeds). The framework messages define the scope of content that will be delivered by the *sender* and provide the *recipient* with vital sizing and scaling information.

The adjustment messages are delivered when the *sender* wishes to reconfigure or adjust the existing framework to remove unneeded information or to add new content to the system unobtrusively. These messages can be as simple as adding a new value to an *enumeration* or as complex as removing all content derived from a *template*.

6.1. AMsg Timestamp (Heartbeat)

Direction: From sender
Tag line: “Please note the time”
URI: http://www.fisd.net/fisdMessage/AMsgTimestamp/
Description: The Timestamp message is sent by the sender and indicates the current time as well as channel sequencing information.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	15 (0x0F)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	“x.y-z”
sequenceNumber	Timestamp Sequence	*n bits*	integer	see compressionInt
previousTimeDelta	Delta Since Previous	*n bits*	integer	see compressionInt

Table 15 - AMsg Timestamp

Special field processing considerations

compressed: This message will never be compressed. Thus the “Compressed” bit must always be zero (0) and there will not be a “fullLength” field.

timestamp: The timestamp includes two fields – a 5 byte field containing the number of seconds since midnight of 01 January 1970, and a 3 byte field containing the number of microseconds (10^{-6}). This time is consistent with the features of most prevailing operating systems at the time of writing of this document.

sequenceNumber: The sequenceNumber is an integer that is incremented with each timestamp message delivered. The recipient may use this field to identify missing messages or to query for recovery purposes. The sequenceNumber should be unique over AT LEAST a 96-hour period.

previousTimeDelta: This is the number of microseconds (10^{-6}) since the previous timestamp was delivered.

Discussion

In broadcast mode, the Timestamp will be present as soon as network connectivity is established. In interactive mode on a fully selective feed, the sender will not transmit the timestamp on the recipient’s channel until after the Session Establishment Authentication Response has been sent (which also includes a Timestamp).

6.2. AMsg Directory Response

Direction: From sender
Tag line: “This is what you need to participate”
URI: http://www.fisd.net/fisdMessage/AMsgDirectoryResponse/
Description: The Directory Response is sent from the sender specifying the messages, templates, and enumerations that are required to fully process the datafeed. Enumerations for all protocol messages are compulsory entries in the directory.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	16 (0x10)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	“x.y-z”
requestKey	Request Key	*n Bytes*	string	from request…
requestStatus	Request Status	*n bits*	enum.	per enumeration
providerDirectory	Provider Directory	1 Byte	bool	vs. framework
compression	Compression Scheme	*n Bytes*	string	for subsequent msgs
compressionInt	Integer Compression.	*n Bytes*	string	for integer counts
compressionTSD	Timestamp Delta Cmp.	*n Bytes*	string	for timestamp deltas
directoryLength	Directory Length	*n bits*	integer	
…Repeating Entries… 1 set per “directoryLength”				
entryType	Entry Type	*n bits*	enum.	
entryNumber	Entry Number	*n bits*	integer	
entrySize	Entry Size	*n bits*	integer	
entryExpression	Entitlement Expression	*n Bytes*	string	
entryEntitlement	Entry Entitlement	4 Bytes	integer	
entryChecksum	Entry Checksum	2 Bytes	integer	
entryDate	Entry Date/Time	8 Bytes	time	
entryURI	Entry URI	*n Bytes*	string	
entryName	Entry Name	*n Bytes*	string	
…End Repeating Entries… for “directoryLength”				

Table 16 - AMsg Directory Response

<u>Special field processing considerations</u>

<u>Discussion</u>

6.3. AMsg Enumeration Response

Direction: From sender
Tag line: "This is an enumeration"
URI: http://www.fisd.net/fisdMessage/AMsgEnumerationResponse/
Description: The Enumeration Response is sent from the sender specifying an enumeration including all of its values and mappings.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	17 (0x11)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	"x.y-z"
requestKey	Request Key	*n Bytes*	string	new value…
requestStatus	Request Status	*n bits*	enum.	per enumeration
enumerationType	Enumeration Type	*n bits*	enum.	form of enumeration
enumerationURI	Enumeration URI	*n Bytes*	string	
enumLength	Enum Length	*n bits*	integer	
…Repeating Entries… 1 set per "enumLength"				
enumOrder	Enum Order	*n bits*	integer	Order in file…
enumValue	Enum Value	*n bits*	integer	Value transmitted
enumShort	Enum Short Desc.	*n Bytes*	string	
enumLong	Enum Long Desc.	*n Bytes*	string	
…End Repeating Entries… for "enumLength"				
enumeration	Enumeration Text	*n Bytes*	string	Full XML Text

Table 17 - AMsg Enumeration Response

Special field processing considerations

Discussion

enumOrder: Specifies the order of the value within the file. Normally, this should be an increasing integer but may not be if there have been changes since the upstream system performed a restart.

enumValue: The equivalent integer value for the enumShort description.

6.4. AMsg Enumeration Renumber

URI: http://www.fisd.net/fisdMessage/AMsgEnumerationRenumber/

Direction: From sender
Tag line: “Renumber this enumeration as defined”
Description: Instructs the recipient to renumber the enumeration referenced. Note that this may be a very expensive command as it requires processing all instances of all templates using the enumeration.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	18 (0x12)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
enumerationURI	Enumeration URI	*n Bytes*	string	
enumLength	Enum Length	*n bits*	integer	
…Repeating Entries… 1 set per “enumLength”				
enumOrder	Enum Order	*n bits*	integer	
enumValue	Enum Value	*n bits*	integer	
enumValueNew	Enum Value New	*n bits*	integer	
…End Repeating Entries… for “enumLength”				

Table 18 - AMsg Enumeration Renumber

Special field processing considerations

Discussion

enumOrder: Specifies the order of the value within the file. Normally, this should be an increasing integer but may not be if there have been changes since the upstream system performed a restart.

enumValue: The equivalent integer value for the enumShort description.

enumValueNew: The new equivalent integer value fo the enumShort description. Thus, entry enumOrder in the enumeration that previously had a value of enumValue is now changed. Note that there should generally always be at least two entries (two values swap locations).

6.5. AMsg Enumeration Extend

To be completed… From provider – "Add some new values to this enumeration". Generally used to add enumerations without the expense of the re-number option.

URI: http://www.fisd.net/fisdMessage/AMsgEnumerationExtend/

Direction: From sender
Tag line: "Add new entries to this enumeration."
Description: Instructs the recipient to add new entries to the end of an existing enumeration. This expansion may cause the number of bits necessary to encode a field containing a value from this enumeration to change.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	19 (0x13)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
enumerationURI	Enumeration URI	*n Bytes*	string	
enumLength	Enum Length	*n bits*	integer	
…Repeating Entries… 1 set per "enumLength"				
enumOrder	Enum Order	*n bits*	integer	
enumValue	Enum Value	*n bits*	integer	
enumShort	Enum Short Desc.	*n Bytes*	string	
enumLong	Enum Long Desc.	*n Bytes*	string	
…End Repeating Entries… for "enumLength"				

Table 19 - AMsg Enumeration Extend

<u>*Special field processing considerations*</u>

<u>*Discussion*</u>

enumOrder: Specifies the order of the value within the file. At least one of the entries should specify a number one greater than the number of entries already within the file. However, if another value is provided (thus implying a swap),

6.6. AMsg Enumeration Remove

URI: http://www.fisd.net/fisdMessage/AMsgEnumerationRemove/

Direction: From sender
Tag line: "Remove this enumeration and invalidate references to it"
Description: Instructs the recipient to delete the enumeration from its processing and set all fields that refer to this enumeration to "undefined". This is potentially an expensive operation.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	20 (0x14)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
enumerationURI	Enumeration URI	*n Bytes*	string	

Table 20 - AMsg Enumeration Remove

<u>*Special field processing considerations*</u>

<u>*Discussion*</u>

6.7. AMsg Template Response

Direction: From sender
Tag line: “This is a template”
URI: http://www.fisd.net/fisdMessage/AMsgTemplateResponse/
Description: The Template Response is sent from the sender specifying a template including all of its actual and virtual fields. This message may be broadcast periodically or transmitted when a template is redefined.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	21 (0x15)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	“x.y-z”
requestKey	Request Key	*n Bytes*	string	from request…
requestStatus	Request Status	*n bits*	enum.	
templateURI	Template URI	*n Bytes*	string	
keyFieldNumber	Key Field Number	*n bits*	integer	
keyFieldSize	Key Field Size	*n bits*	integer	#bits in key allowed
fieldLength	Field Length	*n bits*	integer	
…Repeating Entries… 1 set per “fieldLength”				
fieldNumber	Field Number	*n bits*	integer	
fieldMnemonic	Field Mnemonic	*n Bytes*	string	
fieldName	Field Name	*n Bytes*	string	
fieldType	*Field Type*	*n Bits*	enum.	
fieldUnits	*Field Units*	*n bits*	enum.	
fieldDisplayUnits	*Field Display Units*	*n bits*	enum.	
fieldSize	Field Size	*n bits*	integer	
fieldMinValue	Field Minimum Value	*n bits*	integer	
fieldMaxValue	Field Maximum Value	*n bits*	integer	
fieldEntitlement	Field Entitlement	*n bits*	integer	
fieldEntExpression	Field Ent. Expression	*n Bytes*	string	
fieldEnumeration	Field Enumeration	*n Bytes*	string	
fieldDescription	Field Description	*n Bytes*	string	
fieldDefaultComp	*Field Def. Compression*	*n bits*	enum.	Default from list
fieldCompressions	Field Compressions	*n bits*	integer	Excludes Default
…Repeating Entries… 1 set per “fieldCompressions”				
fieldCompression	*Field Compression*	*n bits*	enum.	
…End Repeating Entries… for “fieldCompressions”				
fieldDependencies	Field Dependencies	*n bits*	integer	Includes Self
…Repeating Entries… 1 set per “fieldDependencies”				
fieldDependency	Field Dependency	*n bits*	integer	
…End Repeating Entries… for “fieldDependencies”				
fieldSubordinates	Field Subordinates	*n bits*	integer	
…Repeating Entries… 1 set per “fieldSubordinates”				
fieldSubordinate	Field Subordinate	*n bits*	integer	
fieldSubFormula	Field Sub. Formula	*n bytes*	string	Interpretive Logic
…End Repeating Entries… for “fieldSubordinates”				
fieldSource	Field Source	*n Bytes*	string	Defined by Headend

fieldFormula	Field Formula	*n Bytes*	string	Interpretive Logic
…End Repeating Entries… for “fieldLength”				
template	Template	*n Bytes*	string	Full XML Text

Table 21 - AMsg Template Response

Special field processing considerations

Discussion

It may seem that the AMsg Template Response will invalidate all instances that refer to the template. However, the recipient should invalidate only those fields within those instances that have changed.

6.8. AMsg Template Remap

URI: http://www.fisd.net/fisdMessage/AMsgTemplateRemap/

Direction: From sender
Tag line: “Remap this template for all current instances”
Description: Instructs the recipient to renumber the fields within a template and thus change all instances of this template. This may be done to resequence fields for more efficient distribution of updates.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	22 (0x16)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
templateURI	template URI	*n Bytes*	string	

Table 22 - AMsg Template Remap

Special field processing considerations

Discussion

TO BE COMPLETED…

6.9. AMsg Template Expansion

URI: http://www.fisd.net/fisdMessage/AMsgTemplateExpansion/

Direction: From sender
Tag line: "Add these new fields to the template"
Description: Instructs the recipient to add new fields to a template in preparation of new content to be delivered. There is an implied change to the template which will be performed by the sender and realized when next the directory is synchronized.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	23 (0x17)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
templateURI	template URI	*n Bytes*	string	

Table 23 - AMsg Template Expansion

<u>*Special field processing considerations*</u>

<u>*Discussion*</u>

TO BE COMPLETED…

6.10. AMsg Template Remove

URI: http://www.fisd.net/fisdMessage/AMsgTemplateRemove/

Direction: From sender
Tag line: “Delete this template and all instances”
Description: Instructs the recipient to delete the template from its processing system and remove all instances that refer to the template. The content of interest is not “unmarked” but all content is invalidated.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	24 (0x18)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
templateURI	template URI	*n Bytes*	string	

Table 24 - AMsg Template Remove

Special field processing considerations

Discussion

6.11. AMsg Instance Reindex

URI: http://www.fisd.net/fisdMessage/AMsgInstanceReindex/

Direction: From sender
Tag line: “Reindex the instances using the mapping provided”
Description: Instructs the recipient to renumber the instances to a different order. This is done to bring the most frequently updated instances to lower numbers to enable more effective compression

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	25 (0x19)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed

Table 25 - AMsg Instance Reindex

<u>*Special field processing considerations*</u>

<u>*Discussion*</u>

TO BE COMPLETED...

6.12. AMsg Instance Remove

URI: http://www.fisd.net/fisdMessage/AMsgInstanceRemove/

Direction: From sender
Tag line: “Delete this instance from your storage”
Description: Instructs the recipient to delete the specified instance from its stores and perform any cleanup processing that is appropriate. This may be done in preparation of sending a fresh image.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	26 (0x1A)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
templateURI	Template URI	*n Bytes*	string	
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer

Table 26 - AMsg Instance Remove

Special field processing considerations

Discussion

7. Content Messages

The Content messages are the most frequently used messages within the system and deliver the data, in a tokenized form, to the consumer. The messages can be categorized in three ways:

- Image - complete content for an *instance*
- Field - updates to specific field(s) within an instance
- Query - responses to previously issued queries (on an interactive feed)

The Content messages are selected, by the *sender*, to deliver the content as efficiently as possible. While the selection of Session Establishment and Administrative messages is straightforward and predictable, the proper selection of Content messages for the optimum bandwidth vs. processing performance is complex. It is encumbent on the *sender* to always send accurate data but the *sender* may choose to expend more processing resources to select a better combination of messages to deliver the data more efficiently or compromise bandwidth for easier implementations and quicker delivery.

7.1. CMsg Image Update

URI: http://www.fisd.net/fisdMessage/CMsgImageUpdate/

Direction: From sender
Tag line: "This is an image – a new or changed instance of a template"
Description: Contains new values for all fields of an instance of a template. Only standard atomic fields are sent – neither compound nor derived fields are transmitted. No compression is applied to the individual fields.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	28 (0x1C)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	"x.y-z"
templateURI	Template URI	*n Bytes*	string	
keyField	*Key Field*	*n bits*	enum.	
keyFieldValue	Key Field Value	*n Bytes*	string	
fieldLength	fieldLength	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per "fieldLength"				
fieldUndefined	Field Value Undefined	1 byte	bool	No value present
fieldValue	Field Value	*n Bytes*	…	field dependent
…End Repeating Entries… for "fieldLength"				

Table 27 - CMsg Image Update

Special field processing considerations

keyField: Identifies an integral index value used in future messages to represent the "keyFieldValue". The number of bits allocated is defined in the template "keyFieldSize".

keyFieldValue: The unique identifier for this instance. The template may define this field as a fixed number of bytes rather than a null-terminated string.

fieldLength: Included only as a convenience and sanity check – this number can be derived from the template definition that is referenced.

fieldUndefined: Indicates no value is present and field should be set to "undefined".

version: Provided as a verification of expected messaging structure.

Discussion

All fields are sent in numerically increasing order (although due to template sorting subsequent fields may not be sequentially adjacent). All included fields are not field-level compressed (the raw values are sent) and all non-transmitted fields should be set to "undefined". After initializing all fields with the provided (or "undefined") values, all compound and virtual fields should be processed.

The image defines the initial content on which all subsequent update messages are applied. Processing update packets without having a complete image may produce undesired results and is therefore discouraged.

7.2. CMsg Fielded Image Update

URI: http://www.fisd.net/fisdMessage/CMsgFieldedImageUpdate/

Direction: From sender
Tag line: "This is an image with fields explicitly identified"
Description: Identical to the Image Update message except that each field number is included with its corresponding value. All fields not transmitted should be set to "undefined". As with Image Update, no field level compression is performed.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	29 (0x1D)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestamp	Timestamp	8 Bytes	time	
version	Messaging Version	*n Bytes*	string	"x.y-z"
templateURI	Template URI	*n Bytes*	string	
keyField	*Key Field*	*n bits*	enum.	
keyFieldValue	Key Field Value	*n Bytes*	string	
fieldLength	fieldLength	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per "fieldLength"				
fieldNumber	*Field Number*	*n bits*	integer	
fieldValue	Field Value	*n bits*	…	field dependent
…End Repeating Entries… for "fieldLength"				

Table 28 - CMsg Fielded Image Update

Special field processing considerations

See "CMsg Image Update".

fieldNumber: Number of bits set by number of fields defined in template "fieldLength".

Discussion

See "CMsg Image Update". All comments apply EXCEPT that the sender may not distribute all fields. Fields not sent should be set to "undefined".

The Fielded Image Update is generally used when a large number of basic fields in the image should be set to "undefined" or when the sender wishes to limit processing delays and use of computational resources.

7.3. CMsg Uncompressed Field Update

URI: http://www.fisd.net/fisdMessage/CMsgUncompressedFieldUpdate/

Direction: From sender
Tag line: "Update this field where key and value are not compressed"
Description: Contains new values for a field within a particular instance. No compression is used on the field – the entire raw value is passed for the field. However, enumerations will be sent as integral values. May contain compound updates.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	30 (0x1E)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestampDelta	Timestamp Delta	*n bits*	time	delta from heartbeat
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer
fieldNumber	*Field Number*	*n bits*	integer	
fieldValue	Field Value(s)	*n bits*	…	field dependent

Table 29 - CMsg Uncompressed Field Update

Special field processing considerations

keyField: May be an integer, a fixed-length string, or a null-terminated string (see CMsg Image Update). If "keyFieldRaw" is clear then this field is an integer, otherwise the type is controlled by the prevailing template (a string or fixed number of bytes).

fieldNumber: Number of bits set by number of fields defined in template "fieldLength".

Discussion

The Uncompressed Field Update is most applicable for values of a field that are near their maximums or, for whatever reason, may not compress efficiently using the default compression scheme. Uncompressed Field Update may also be used by the sender to limit processing delays and use of computational resources.

The fieldNumber may point to a single or compound field. In the event of a compound field, the uncompressed values of all subordinate fields will be concatenated into fieldValue. The number of bits assigned to each fieldValue is a function of the maximum number of bits defined with the template for the field.

7.4. CMsg Compressed Field Update

URI: http://www.fisd.net/fisdMessage/CMsgCompressedFieldUpdate/

Direction: From sender
Tag line: "Update this field with value compressed normally"
Description: Contains new values for a field where the standard compression defined for this field has been applied. This message is identical to the Uncompressed Field Update in every other respect.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	31 (0x1F)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestampDelta	Timestamp Delta	*n bits*	time	delta from heartbeat
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer
fieldNumber	*Field Number*	*n bits*	integer	
fieldValue	Field Value(s)	*n bits*	…	field dependent

Table 30 - CMsg Compressed Field Update

Special field processing considerations

See "CMsg Uncompressed Field Update".

Discussion

The Compressed Field Update is most applicable when the value for a field can be compressed most efficiently using the default compression defined for that field.

The fieldNumber may point to a single or compound field. In the event of a compound field, the compressed values of all subordinate fields, using the default compression for each field, will be concatenated into fieldValue. The number of bits assigned to each fieldValue is a function of the compression schemed selected for the field.

7.5. CMsg Alternate Field Update

URI: http://www.fisd.net/fisdMessage/CMsgAlternateFieldUpdate/

Direction: From sender
Tag line: "Update this field with specified alternate compression"
Description: Contains new values for a field within a particular instance when the standard field compression is not appropriate. The compression applied, which must be from the list of legal compressions for this field, is included.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	32 (0x20)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestampDelta	TimestampDelta	*n bits*	time	delta from heartbeat
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer
fieldNumber	*Field Number*	*n bits*	integer	
fieldCompression	*Field Compression*	*n bits*	enum.	field dependent
fieldValue	Field Value	*n bits*	…	field dependent

Table 31 - CMsg Alternate Field Update

Special field processing considerations

See "CMsg Uncompressed Field Update".

fieldCompression: The compression selected from the list of compressions that are legal for the field (independent of the base type). This list may include two values that, if selected, would be better distributed using another message: 1) "none" should be delivered as an Uncompressed Field Update, and 2) the default compression for the field would be better sent as a Compressed Field Update.

fieldNumber: May only point to a single field - compound fields are not allowed.

Discussion

The Alternate Field Update is most applicable when the value for a field can be compressed most efficiently using a compression other than the default compression defined for that field. Certain compressions, namely those that use a "continuation bit" to indicate follow-on bits or bytes, may create compressed values that are inordinately longer than the original value. However, an alternate compression may prove more efficient. The overhead of indicating the fieldCompression should be weighed into the decision to use this message.

7.6. CMsg Compressed Range Update

URI: http://www.fisd.net/fisdMessage/CMsgCompressedRangeUpdate/

Direction: From sender
Tag line: "Update this range of fields"
Description: Contains new values for a contiguous range of fields within a particular instance. This message is similar to the Compressed Field Update message except that multiple sequential fields are concatenated.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	33 (0x21)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestampDelta	Timestamp Delta	*n bits*	time	delta from heartbeat
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer
fieldNumberBegin	*Field Number Begin*	*n bits*	integer	
fieldNumberDelta	*Field Number Delta*	*n bits*	integer	
fieldValue	Field Value(s)	*n bits*	..	field dependent

Table 32 - CMsg Compressed Range Update

Special field processing considerations

See "CMsg Uncompressed Field Update".

fieldNumberBegin: The first field included in the update.

fieldNumberDelta: The number of fields included in the update.

Discussion

The Compressed Range Update is most applicable when sequential fields need to be updated that are not part of a predefined compound field.

The fieldValue contains fieldNumberDelta fields whose compressed values are concatenated together. The field numbers are identified sequentially beginning at fieldNumberBegin.

7.7. CMsg Undefine Field Update

URI: http://www.fisd.net/fisdMessage/CMsgUndefineFieldUpdate/

Direction: From sender
Tag line: "Update this field by setting its value to undefined"
Description: Indicates that the specified field should be cleared or set to "undefined" because it is no longer valid or is contradictory to other field values. The identified field may be a compound field so all applicable processing should be applied.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	34 (0x22)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestampDelta	Timestamp Delta	*n bits*	time	delta from heartbeat
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer
fieldNumber	*Field Number*	*n bits*	integer	

Table 33 - CMsg Undefine Field Update

Special field processing considerations

See "CMsg Uncompressed Field Update".

fieldNumber: The field that should be set to "undefined".

Discussion

The fieldNumber may point to a single or compound field. In the event that a compound field is referenced, all subordinate fields should also be set to "undefined".

7.8. CMsg Instance Expansion Update

URI: http://www.fisd.net/fisdMessage/CMsgInstanceExpansionUpdate/

Direction: From sender
Tag line: "Add a new collection of fields to the end of the instance"
Description: Contains values for a new occurrence of all the fields in a compound field within a particular instance. The subordinate fields of the compound field are copied. Normal compression is used for each of the fields within the compound field.

Field Mnemonic	Name	Size	Type	Value/Notes
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	35 (0x23)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
timestampDelta	Timestamp Delta	*n bits*	time	delta from heartbeat
keyFieldRaw	Key Field Provided Raw	1 bit	bool	keyField is string
keyField	*Key Field*	*n bits*	…	string or integer
fieldNumber	*Field Number*	*n bits*	integer	
newFieldNumber	*New Field Number*	3 Bytes	integer	first of field list
fieldValue	Field Value(s)	*n bits*	…	field dependent

Table 34 - CMsg Instance Expansion Update

Special field processing considerations

See "CMsg Uncompressed Field Update".

newFieldNumber: An uncompressed field number. If the number of bits necessary to define the newFieldNumber (plus the number of fields necessary for all of the subordinate fields of fieldNumber) then the number of bits for fieldNumber fields in all updates must be adjusted.

Discussion

The fieldNumber is most normally a compound field of a repeating series. All of the fields that are part of the compound field are copied to sequentially subsequent fields beginning with newFieldNumber.

The newFieldNumber value identifies a field number previously not defined for the instance. As noted above, if the magnitude of newFieldNumber, plus the number of subordinate fields, exceeds the number of bits necessary to represent the previous maximum field number for the instance, then the number of bits must be adjusted. All updates from this point forward MUST use the new bit count when identifying fields.

7.9. CMsg Blocked Field Update

URI: http://www.fisd.net/fisdMessage/CMsgBlockedFieldUpdate/

Direction: From sender
Tag line: “Update these fields on these instances”
Description: Multiple updates applicable to the same timestamp or same instance have been lumped together in a single message. Process each of the updates against the appropriate instance(s).

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	36 (0x24)
messageLength	Message Length	*n Bytes*	integer	“i8.0” compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed

Table 35 - CMsg Blocked Field Update

<u>Special field processing considerations</u>

<u>Discussion</u>

TO BE COMPLETED…

The intent with this message is to provide a container for multiple field updates to limit the duplication of the header for each message, to potentially eliminate multiple references to the key field, and promote transport layer (TCP and/or IP) economies that can be accommodated when multiple fisdMessage packets are put into a single communications message (for the lower levels).

The structure (when finalized) will accommodate:
- An implied “AMsg Timestamp” for facilitating compression
- Multiple “CMsg Compressed Field Update” messages for the same instrument
- Multiple “CMsg Compressed Field Update” messages for different instruments
- Support for “CMsg Uncompressed Field Update” messages
- Support for “CMsg Alternate Field Update” messages
- Support for “CMsg Compressed Range Update” messages
- Support for “CMsg Undefine Field Update” messages
- In short, support for all content field update messages in a shared wrapper

7.10. CMsg Query Response

URI: http://www.fisd.net/fisdMessage/CMsgQueryResponse/

Direction: From sender
Tag line: "Here is the response to your query"
Description: Contains the status to a previous query. If the query was for other, non-markable content, it may contain the response document as well.

Field Mnemonic	**Name**	**Size**	**Type**	**Value/Notes**
encrypted	Encrypted	1 bit	bool	for payload
compressed	Compressed	1 bit	bool	for payload
messageType	Message Type	6 bits	enum.	37 (0x25)
messageLength	Message Length	*n Bytes*	integer	"i8.0" compression
fullLength	Uncompressed Length	*n Bytes*	integer	Only if compressed
requestKey	Request Key	*n Bytes*	string	new value…
queryLength	Query Length	*n bits*	integer	see compressionInt
…Repeating Entries… 1 set per "queryLength"				
requestStatus	Request Status	*n bits*	enum.	per enumeration
queryResponse	Query Response	*n Bytes*	string	
…End Repeating Entries… for "queryLength"				

Table 36 - CMsg Query Response

<u>Special field processing considerations</u>

<u>Discussion</u>

8. Example Templates

The following examples show how templates are used to construct messages. While the details of each field have been omitted for brevity, the reader should be able to garner an understanding of the methodology and considerations when designing a system with this protocol. More detailed examples are provided in the appendix.

On example is provided for the protocol messages and a second example is provided showing a potential use with a content XML like MDDL.

8.1. For Messages

The complete format for each message is included in a separate distribution. This example illustrates how the “AMsg Timestamp” is encoded for delivery “over-the-wire” and decoded on reception.

Template Outline for AMsg Timestamp (actual field specifiers removed for brevity):

```
<fisdMessage version=”1.0-beta”>
  <header>
    <messageType><!-- messageType --></messageType>
    <encrypted><!-- encrypted --></encrypted>
    <compressed><!-- compressed --></compressed>
    <messageLength><!-- messageLength --></messageLength>
    <fullLength><!-- fullLength --></fullLength>
  </header>
  <AMsgTimestamp>
    <timestamp><!-- timestamp --></timestamp>
    <version><!-- version --> </version>
    <sequenceNumber><!-- sequenceNumber --></sequenceNumber>
    <previousTimeDelta><!-- previousTimeDelta --></previousTimeDelta>
  </AMsgTimestamp>
</fisdMessage>
```

With the following information relative to the framework:

- “compressionInt” is the default value of “i8.0”
- There are no additional compressions defined for any of the fields
- There are no entitlements required for the fields
- There are no formulas (or other complications)
- All fields are dependent on one another (and so are always sent together)

And the following information about the template (XML of field definitions not provided)

- Field #1- “encrypted” with no compression fixed at 1 bit
- Field #2- “compressed” with no compression fixed at 1 bit
- Field #3- “messageType” with no compression fixed at 6 bits
- Field #4- “messageLength” with compression “i8.0”
- Field #5- “fullLength” with compression “i8.0” dependent on field #2=1
- Field #6- “timestamp” with no compression fixed at 64 bits
- Field #7- “version” and is a string
- Field #8- “sequenceNumber” with compression “i8.0”
- Field #9- “previousTimeDelta” with compression “i8.0”

Example Instance for AMsg Timestamp (which is 473 bytes):

```
<fisdMessage version="1.0-beta">
  <header>
    <messageType>AMsgTimestamp</messageType>
    <encrypted>false</encrypted>
    <compressed>false</compressed>
    <messageLength></messageLength>
    <!-- no fullLength -->
  </header>
  <AMsgTimestamp>
    <timestamp>2003-12-06T03:30:23.12Z</timestamp>
    <version>1.0-beta</version>
    <sequenceNumber>53367</sequenceNumber>
    <previousTimeDelta>5089</previousTimeDelta>
  </AMsgTimestamp>
</fisdMessage>
```

The byte stream transmitted for the above message looks like this (24 bytes total):

```
Field             Value      In hex (encoded)
encrypted         0          0
compressed        0          0
messageType       15         0F
messageLength     21         15 (15)
fullLength        -----      --
timestamp         (above)    00 3F D1 4D 4F 01 D4 C0
version           1.0-beta   31 2E 30 2D 62 65 74 61 00
sequenceNumber    53367      D0 77 (83 A0 77)
previousTimeDelta 5089       13 E1 (A7 61)

Concatenated "over-the-wire":
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
0F 15 00 3F D1 4D 4F 01 D4 C0 31 2E 30 2D 62 65 74 61 00 83 A0 77 A7 61
```

This is how the byte stream might be decoded (depending on implementation):

- 2 bytes are read from the network and compared against the header configuration
- Field #1- high bit is tested for "encrypted" -> it is not
- Field #2- second bit is tested for "compressed" -> it is not
- Field #3- remaining 6 bits are compared -> "messageIdentifier" is "AMsgTimestamp"
- Field #4- second byte is examined and decoded -> 21 more bytes to be read
- 21 more bytes are read from the network
- Field #5- is "zero length" and is skipped as no decompression is needed
- Processing of an "AMsgTimestamp" starts at byte 3
- Field #5- 8 bytes are decoded -> timestamp is "2003-12-06T03:30:23.120000Z"
- Field #6- String is read starting at byte 11 -> "version" is "1.0-beta" with null
- Field #7- Byte 20 is examined and decoded -> need another byte
- Byte 21 is examined and decoded -> need another byte
- Byte 22 is examined and decoded with bytes 20-21 -> "sequenceNumber" is 53367
- Field #8 - Byte 23 is examined and decoded -> need another byte
- Byte 24 is examined and decoded with byte 23 -> "previousTimeDelta" is 5089.

8.2. For Content

Example template of trade (field specifics removed for brevity):

```
<mddl version=”2.3-beta”>
<header>
    <dateTime><!-- myMessageTime --><dateTime>
    <source>XTC Demonstration</source>
</header>
<snap><equityDomain><commonClass>
    <instrumentIdentifier>
        <code scheme=”http://www.mddl.org/ext/scheme/symbol?SRC=XTKS”
          ><!-- myTicker --></code>
        <name><!-- myCompanyName --></name>
    </instrumentIdentifier>
    <sequence><!-- mySequence --></sequence>
    <session><!-- mySession --></session>
    <trade>
        <last><!-- myTradePrice --></last>
        <dateTime><!-- myTradeTime --></dateTime>
        <marketCenter>
            <code scheme=”http://www.mddl.org/xtc/scheme/iso10383.xml”
              ><!-- myTradeExchange --></code>
        </marketCenter>
        <size><!-- myTradeSize --></size>
        <currency><!-- myTradeCurrency --></currency>
        <status scheme=”http://www.mddl.org/xtc/scheme/tradeStatus.xml”
          ><!-- myTradeStatus --></status>
    </trade>
</commonClass></equityDomain></snap>
</mddl>
```

Example instance document of trade (which is 912 bytes):

```
<mddl version=”2.3-beta”>
<header>
    <dateTime>2003-12-05T22:30:23.380+05:00<dateTime>
    <source>XTC Demonstration</source>
</header>
<snap><equityDomain><commonClass>
    <instrumentIdentifier>
        <code scheme=”http://www.mddl.org/ext/scheme/symbol?SRC=XTKS”
          >6501</code>
        <name>A Company in Your Neighborhood</name>
    </instrumentIdentifier>
    <sequence>1306</sequence>
    <session>1</session>
    <trade>
        <last>12375</last>
        <dateTime>2003-12-05T22:30:23.360+05:00</dateTime>
        <marketCenter>
            <code scheme=”http://www.mddl.org/xtc/scheme/iso10383.xml”
              >XTKS</code>
        </marketCenter>
        <size>200</size>
        <currency>JPY</currency>
        <status scheme=”http://www.mddl.org/xtc/scheme/tradeStatus.xml”
          >normal</status>
    </trade>
</commonClass></equityDomain></snap>
</mddl>
```

With the following information relative to the framework:

- “compressionInt” is the default value of “i8.0”
- There are no entitlements required for the fields
- There are no formulas (or other complications)
- System time resolution is defined to be 10 milliseconds (10^{-2}) or 0.01.
- A heartbeat message is sent once a second.
- Key fields are given 12 bits with compression “i8.0”.
- This template has 12 fields so 4 bits are assigned. However, the most frequent field is #1 so compression “i3.0” is selected.

And the following information about the template (XML of template fields not provided)

- Field #1- a compound for fields 2-7
- Field #2- “mySequence” with compression “i8.1”
- Field #3- “myTradePrice” with compression “i8.1”
- Field #4- “myTradeTime” with compression “tHs8.2”
- Field #5- “myTradeSize” with compression “i8.0”
- Field #6- “myTradeStatus” as enumeration with 32 values “i4.0” comp.
- Field #7- “myTradeExchange” as enumeration with 16 values “i4.0” comp.
- Field #8- “mySession” with compression “i4.0”
- Field #9- “myTradeCurrency” as enumeration with 16 values “i4.0” comp.
- Field #10- “myCompanyName” is a string
- Field #11- “myMessageTime” derived from message timestamp
- Field #12- “myTicker” derived from message index

Now, compose an update for compound field #1 with the following values:

- Field #2- mySequence is 1306
- Field #3- myTradePrice is 12375 (yen)
- Field #4- myTradeTime is “2003-12-05T22:30:23.360+05:00”
- Field #5- myTradeSize is 200
- Field #6- myTradeStatus is “normal” (value 2 of 16)
- Field #7- myTradeExchange is “XTKS” (value 7 of 16)
- Field #12- myTicker is “6501” (assigned index 0xEF4 in the system)
- Last timestamp is 2003-12-06T03:30:23.12Z

So, why not send the other fields with every trade?

- Field #8- the session only changes twice a day
- Field #9- all trades are done in Japanese yen
- Field #10- the company name is loaded once and never changes
- Field #11- myMessageTime is linked to the field update timestamp
- Field #12- myTicker is linked to the message key field

The System Headend decides to use a “CMsg Compressed Field Update” (see Section 7.4). Each of the fields from the message must be compressed:

Field	Value	In hex (encoded)
mySequence	1306	05 1A (05 1A)
myTradePrice	12375	30 57 (30 57)
myTradeTime	24	18 (18)

```
myTradeSize        200       C8 (C8)
myTradeStatus      2         02 (02) 4 bits
myTradeExchange    7         07 (07) 4 bits

Concatenated for "over-the-wire" delivery:
 1  2  3  4  5  6  7
05 1A 30 57 18 C8 27
```

The byte stream transmitted for the above message looks like this (13 bytes total):

```
Field              Value     In hex (encoded)
encrypted          0         0
compressed         0         0
messageType        31        1F
messageLength      12        0C (0C)
fullLength         -----     --
timestampDelta     26        1A (1A)
keyFieldRaw        0         00 (00) 1 bit
keyField           3828      0E F4 (9D 74)
fieldNumber        1         01 (01) 3 bits
fieldValue         (above)   05 1A 30 57 18 C8 27
waste              00        00 (00) 4 bits

Concatenated "over-the-wire":
 1  2  3  4  5  6  7  8  9 10 11 12 13
1F 0C 1A 4E BA 10 51 A3 05 71 8C 82 70
```

Possible reductions in the size of the message:

- myTradeSize is encoded as a full byte. In many cases, such trades can be reported as n*10 or n*100. Using compression "i4.0" with a formula of "*100", myTradeSize could be reported in 4 bits. Similarly, other formulas for a factor of 5 or other multiples could be used. Note that this would cause an increase in the number of fields such that 3 bits would not be sufficient.
- myTicker (the keyField) required 16 bits to encode. Judicious selection of the integral value for an instrument (i.e. by putting the most frequently used instruments in the lower numbers) could bring this size down to 8 bits for the most active instruments.
- mySequence is an integral counter for updates to the instrument. Many systems do not report this kind of number with the message thus the message size could be reduced by 16 bits by eliminating this field.
- myExchange allows for a stock to trade on 16 exchanges and requires 4 bits to encode. Many stocks only trade on a couple of exchanges so this field could be sized at 2 bits (for Tokyo listed instruments). In fact, additional virtual fields with formulas can be added which set the value of this field explicitly thus this field can be removed from the update entirely.
- myTradeStatus can be handled via formulas (as with myExchange) thus removing this field from the update (but increasing the number of fields).
- The timestamp deltas were encoded with 8 bits each because the heartbeat message is sent once per second and the time resolution is 0.01 (so we must be able to count +/- 100 units since the last timestamp). If we increase the timestamp to once every quarter second, then time deltas can be calculated to +/- 25 units and 5 bits each would be sufficient (thus saving 6 bits).
- Basically, this 13 byte message could be sent in 9 bytes or fewer if similar adjustments are made. Understanding of the most commonly distributed data will help the system designer pick the right compressions, proper sequencing of

values within enumerations, efficient field ordering, and effective compound fields. Consider these assignments for 8 bytes 2 bits of packet structure:

- mySequence - average of 12 bits (optional!)
- myTradePrice - 4 bits with formula support
- myTradeTime - 6 bits
- myTradeSize - 4 bits with formula support
- myTradeStatus - 0 bits with formula support
- myTradeExchange - 0 bits with formula support
- timestampDelta - 6 bits
- keyField (w/Raw) - average of 10 bits
- fieldNumber - average of 8 bits
- messageType and messageLength - 16 bits