ETP Specification

|  |  |
| --- | --- |
| **Standards Document Title** | **ETP Specification** |
| **Version:** | **0.0.2.1** |
| Abstract | This the abstract of the document. |
| Prepared by: | WITSML SIG |
| Date published: | August 14, 2014 |
|  |  |
| Document type: | Specification |
|  |  |
| Keywords: | Standards, data, information, process |



This document was produced by  
Energistics and the ETP Workgroup of the WITSML SIG.

Energistics™, POSC®, Epicentre®, WITSML™, PRODML™, Upstream Standards. Bottom Line Results.™, The Energy Standards Resource Centre™ and their logos are trademarks or registered trademarks of Energistics. Access, receipt, and/or use of these documents and all Energistics materials are generally available to the public and are specifically governed by the Energistics Product Licensing Agreement at <http://www.energistics.org>

Section: Document\_versions

End

Table of Contents

[Table of Contents 5](#_Toc395767743)

[1. Introduction 6](#_Toc395767744)

[1.1 Document Conventions 6](#_Toc395767745)

[**Stability Index** 7](#_Toc395767746)

[1.2 Overview 7](#_Toc395767747)

[1.3 Concepts 9](#_Toc395767748)

[1.3.1 WebSocket Transport 10](#_Toc395767749)

[1.3.2 Avro Serialization 10](#_Toc395767750)

[1.3.3 Client, Servers and Roles 11](#_Toc395767751)

[1.3.4 Protocol 0 – Session Management 11](#_Toc395767752)

[1.3.5 Protocol 1 – Channel Data 12](#_Toc395767753)

[2. Protocol 13](#_Toc395767754)

[2.1 Core 13](#_Toc395767755)

[2.1.1 Requirements 14](#_Toc395767756)

[2.1.2 Interface : IClient 16](#_Toc395767757)

[2.1.3 Interface : IServer 17](#_Toc395767758)

[2.2 ChannelStreaming 18](#_Toc395767759)

[2.2.1 Requirements 19](#_Toc395767760)

[2.2.2 Interface : IChannelConsumer 22](#_Toc395767761)

[2.2.3 Interface : IChannelProducer 23](#_Toc395767762)

[3. Schemas 25](#_Toc395767763)

[3.1 Energistics 25](#_Toc395767764)

[3.1.1 Datatypes 26](#_Toc395767765)

[3.1.2 Protocol 40](#_Toc395767766)

Section: Introduction

# Introduction

**Stability:** : n/a

## Document Conventions

| **Example** | **Description** |
| --- | --- |
| <productVolumes> | This font and color are used for code samples, examples, display text, error messages, parameter values or the value of XML elements and attributes, and XML examples. |
| **File + Open** | In syntax, words you are instructed to type appear in bold. In text, bold is used to identify the names of data types, menus, menu options, dialog boxes, and other named items in the user interface. |
| ENTER | Capital letters are used for the names of keys and key sequences, such as ENTER and CTRL+R. |
| ALT + F1 | A plus sign (+) between key names indicates a combination of keys or menu item selections. For example, ALT + F1 means to hold down the ALT key while pressing the F1 key. |
| *dTimCurrent* | Italic font is used when referring to file names, class names and methods, parameter names, variables, or the names of XML elements and attributes. |

**Stability Index**

Throughout the documentation, you will see indications of a section's stability. This concept was borrowed from the Node.js API and is intended to allow evolutionary development of the specification, while allowing implementers to use with confidence the portions that are stable. ETP is still changing, and as it matures, certain parts are more reliable than others. Some are so proven, and so relied upon, that they are unlikely to ever change at all. Others are brand new and experimental, or known to be hazardous and in the process of being redesigned.

The stability indices are as follows:

Stability: 0 - Deprecated

This feature is known to be problematic, and changes are planned. Do not rely on it. Use of the feature may cause warnings. Backwards compatibility should not be expected.

Stability: 1 - Experimental

This feature was introduced recently, and may change or be removed in future versions. Please try it out and provide feedback. If it addresses a use-case that is important to you, tell the node core team.

Stability: 2 - Unstable

The API is in the process of settling, but has not yet had sufficient real-world testing to be considered stable. Backwards-compatibility will be maintained if reasonable.

Stability: 3 - Stable

The API has proven satisfactory, but cleanup in the underlying code may cause minor changes. Backwards-compatibility is guaranteed.

Stability: 4 - API Frozen

This API has been tested extensively in production and is unlikely to ever have to change.

Stability: 5 - Locked

Unless serious bugs are found, this code will not ever change. Please do not suggest changes in this area; they will be refused.

## Overview

Energistics Transport Protocol (ETP) is a new data exchange specification that enables the efficient transfer of real- time data between applications. It will initially be implemented as part of the WITSML specifications but is expected to be used in other areas. The original working name for ETP was RLF or RaLF.

One of the aims of ETP is to replace TCP/IP WITS level 0 data transfers with a more efficient and but simple- to- implement alternative.

The three main initial use cases for ETP are to move real-time data between applications, including:

* Transfer from a wellsite provider to a WITSML store (server)
* Transfer of data from WITSML store to WITSML store (replication)
* Transfer of data from WITSML store to client applications

ETP defines a publish/subscribe mechanism so that data receivers do not have to poll for data and can receive new data as soon as they are available from a data provider.

It is anticipated that ETP will be expanded beyond real-time data transfer to include functionality for data discovery and historical data queries. ETP has the potential to be the underlying protocol for the next major revision of WITSML.

End

## Concepts

This section is non-normative.

ETP is a **communication protocol**. A communication protocol, generically, can be thought of as a precise set of rules for the exchange of data between agents, usually in the form of some kind of message. ETP has been specifically envisioned and designed to meet the unique needs of the upstream oil and gas industry and more specifically to facilitate the exchange of data in the EnergyML family of data standards (WITSML, PRODML, RESQML).

Like almost all modern communication protocols, ETP employs a layered approach, and indeed sits right on top of the existing TCP layered model. Thus, the concept of protocol is used in many contexts through this document and the notion of a **subprotocol** is used to discuss protocols which sit (somewhat un-intuitively) just *above* another protocol in the stack.

Section: dProtocolStack



Figure : ETP Protocol Stack

End

We can see that ETP is itself a subprotocol of the Web Socket protocol and that ETP also has its own layers and subprotocols. The Core protocol has a direct connection to WS and is agnostic to the various kinds of messages which are carried in each of its own subprotocols. So it is important to understand that the terms protocol and subprotocol are used interchangeably in this document, about any layer, depending on context. This layered approach allows for separation of concerns between the various parts of the stack and supports the adoption of future standards that develop lower in the stack. Specifically, ETP is itself a subprotocol of WebSocket, and ETP defines its own set of subprotocols, each designed to carry a specific data that follows a specific pattern (in terms of size, frequency, and variability). The elements of ETP shown in gray outline are for future releases and are not part of Release 1 of ETP.

All of the ETP subprotocols are assigned a numeric identifier as part of this specification, and all messages will be identified this way in the message header. The Core protocol is protocol 0 and the initial channel data protocol is protocol 1. This pattern of assigning numeric identifiers to key enumerations in the protocol will be used regularly, in order to allow for the smallest most efficient binary transfers.

ETP has been design to support multiple patterns, or styles of message transfer, especially with respect to message size, frequency and complexity. Separate subprotocols of ETP will generally related to specific kinds of messages. This is somewhat in contrast with previous Energistics data service specifications, where each SIG had service contracts that were unique to the domain objects of that SIG. With ETP, we are trying to use a single set of protocols across multiple domain areas, with the differences in protocol focused more on the communication requirements. The initial release of ETP will focus on high-speed streaming of channel data as described below.

### WebSocket Transport

ETP is initially designed to use the WebSocket protocol for transport. A full description of WebSocket is beyond the scope of this document, but there are many references available [TODO]. In brief, WebSocket is a protocol, standardized by the IETF as [RFC 6455](http://tools.ietf.org/html/rfc6455) which allows for high-speed, full-duplex, binary communication between agents (primarily web servers and browsers) utilizing TCP and the standard HTTP(s) ports 80/443. This allows communications to easily cross many corporate firewalls safely. Like WebSocket itself, ETP communication is strictly between two parties, with no allowance for multi-cast messages.

ETP is bound to WebSocket in two main ways:

1. ETP is considered a subprotocol of WebSocket as defined in in sections 1.9 and 11.5 of RFC6455.
2. ETP messages map directly to the “payload data” section of WebSocket frames and messages. In most cases these details will be invisible to developers, as they will use a vendor supplied library to interface with WebSocket.

All ETP communication is carried out through the **asynchronous** exchange of **messages**. This is distinct from the Request/Response pattern normally associated with HTTP, and the “RPC Style” associated with many SOAP implementations, including WITSML 1.0. Of course there are still many use cases that require a request on the part of one agent and expect a reponse of some sort from the other, however implementors should always consider these things to be happening asynchronously and handle processing via state machines that model the various timings of message exchange that could occur. A number of mechanisms are used within the protocol to ensure message are correlated correctly.

1. All messages within a session are numbered. Message numbers are integers and must be unique within a session.
2. A correlationId is included in each message, which may designate a given message as being a part of a ‘response’ to a previous request.
3. Large messages may be sent in parts, with a mechanism in the message header for determining when the last part has arrived.
4. Various ETP subprotocols may impose specific ordering and numbering of certain messages.
5. The WebSocket protocol itself guarantees the delivery of messages in the same order that they were dispatched.

These are described in more detail below.

### Avro Serialization

The **serialization** of messages in ETP follows the [Apache Avro specification](http://avro.apache.org/docs/current/spec.html). **Avro** is a system for defining schemas and serializing data objects according to those schemas. It was developed as a part of the Hadoop® project to provide a flexible, high-speed serialization mechanism for processing big-data. Avro was selected by the ETP design team after a review of a number of similar serialization systems. It is important to note that ETP uses only a subset of the functionality defined in Avro. In particular:

1. ETP **does** define all messages using the schema file format of Avro. The formal definitions of these schemas are defined in UML class models using Enterprise Architect, and the specifications in the document and the avro schema files are generated from these models.
2. ETP **does** serialize all messages on the wire in accordance with the Avro serialization rules.
3. ETP **does not** make use of the Avro RPC facility.
4. ETP **does not** make use of the Avro container file facility.
5. ETP makes use of the additional schema attributes (permissible in Avro) to define message and protocol metadata.

The Avro specification supports the use of both **binary** and **json** encoding of data. ETP also supports the use of both, with the following caveats:

1. All messages within a given ETP session must use the same encoding (binary or json). The encoding that is used is negotiated as per the discussion below of protocol 0.
2. Agents are not required to support both encodings. This is primarily to allow smaller, resource-constrained implementations to use only one encoding.

Unlike XML, Avro has no concept of a well-formed vs. valid document or a generic document node model, thus it is not possible to de-serialize an Avro document without knowledge of the schema of that document. For this first release of ETP, there will be an assumption that all parties have prior knowledge of the schemas involved. In future releases there will be the capability to exchange version-specific schemas at the time of negotiating the session, allowing an agent to consume any ETP message, even if not able to use all of the information in the message.

### Client, Servers and Roles

Connsistent with all TCP communication, ETP includes the fundamental roles of **client** and **server**. That is to say, in all ETP communication, one agent must be a server that is listening on a TCP port, and one agent is a client that begins by connecting on that address and port. Beyond this, however, the general direction of information flow is independent of this client/server relationship. This is a business requirement for WITSML. To accommodate this, each subprotocol in ETP will define a set of **roles** that are appropriate for that protocol. The assignment of these roles to agents will happen as a part of the Core (protocol 0). In general, the client will begin by telling the server which subprotocols it wishes to use, and the named roles it intends to fulfill in this session. The server will respond to indicate it is able to fulfill its role, closing the session if it cannot. This ‘client goes first’ logic is predicated on the basis that a client generally knows why it is connecting to server, where a server capable of support two different roles has no way of knowing which one the client wishes to use.

For a more concrete example: The channel data protocol (protocol 1) defines two roles: **producer** and **consumer**. Each agent is either a producer of consumer of streaming data. Simplistically, we could say that a device is usually a producer, an aggregator and a company store can be a producer or a consumer, and a web-browser is usually only a consumer. The connection from aggregator to store is of most interest. Either agent could be the client, but on connecting, the store client would ask to be a consumer of data where the aggregator as client would ask to be a producer.

### Protocol 0 – Session Management

ETP includes the notion of a **session**. This session is separate and unrelated to any HTTP session that exists as part of the WebSocket connection and upgrade. The creation and management of session happens through the first ETP subprotocol, known as the **Core** protocol, and identified as **protocol 0**. Sessions are assigned a unique identifier, which is a GUID, when they are created. The major purpose of the session is to allow for a context within which various domain objects can be referenced in messages by identifiers, as opposed to their character-based names. This allows for much smaller messages on the wire and more efficient processing of messages in code.

WebSocket Header

ETP Message Header

ETP Message

Figure 2 - ETP Message Layout

Each message in ETP begins with a common **message header**. The main purposes of the header are:

1. To identify the message subprotocol within ETP.
2. To identify the type of message being sent (messageType) - which implicitly defines the schema for the message data.
3. To identify the id of the message itself - each message in an ETP session is uniquely numbered.
4. The message header contains an option correlationId attribute. It’s purpose somewhat is-context sensitive, depending on the message in question, but in general it is used to correlate messages that are in response to some previous request. As an example, when a consumer makes ChannelRangeRequest, all of the resulting ChannelData messages will have a correlationId which is the value of the original ChannelRangeRequest message.
5. Finally, there is an optional messageFlags attribute for each message. This acts as a bit-field and allows multiple boolean flags to be set on the message. There are two flags currently defined: one that says this is a multi-part response (i.e. “more data messages are coming related to this request.”) and one that identifies a message as the final part of a multi-part message.

Note that unlike SOAP and XML, there are no begin/end message tags, and so there is no concept of an ‘envelope’ with ETP messages.

### Protocol 1 – Channel Data

Channel

Data Point

DataObject

Discovery

Large binary

Section: Protocols

# Protocol

**Stability:** : n/a

This section is normative.

This section provides a dynamic view of the protocols in ETP, using the following UML conventions.

1. Interfaces are used to model the implementation requirements for clients, server, producers and consumers, etc. There will thus generally be 2 interfaces per protocol, one for each role. This is not guaranteed in the future, as there may be purely peer-to-peer protocols which are identical to all agents. The arguments to each interface method are always called 'eventData' and are typed according to the message names in the Schemas section of this document.
2. There is an assumption the each message also carries a MessageHeader. **TODO should this be modeled using inheritance in the message classes? Would not affect avro schemas.**
3. Interface methods that send messages are not stereotyped, methods that receive or handle incoming messages are stereotyped as <<event>> methods. In some cases (such as CloseSession) either party may send or receive the message, and this will be reflected by having both call and event method.
4. The general behavioral aspects of each protocol are modeled as **Requirements** and documented immediately following the protocol description.
5. **Sequence** diagrams are used to model the normal flow of messages in each protocol.
6. Separate **Protocol State Diagrams** are used to model the possible states of each role in a sub protocol.



Figure : Sample Interfaces

This diagram demonstrates how the opposite roles of a protocol will implement similar but opposite interfaces. The calls for one become the events for the other.

## Core

**Stability:** : 3 - Stable

The Energistics.Core package (also referred to as 'Protocol 0') has the following responsibilities:

1. Establish the Web Socket Connection
2. Process Web Socket or HTTP Headers to determine a) whether Binary or JSON encoding will be used and b) the version of Protocol-0 itself that will be used.
3. Manage Session life cycle.
4. Negotiate the sub-protocols to be used in the session.

The core protocol also defines common messages which may be used in any protocol (such as the Exception message). These messages have a MessageTypeId >= 1000.

### Requirements

| Requirement | Type | Description |
| --- | --- | --- |
| Message IDs | Behavior | Message IDs MUST be:   1. Unique within a session, and for a given agent (i.e. client/server). 2. Generated increasingly from 1 for a session 3. Increasing within a multi-part response. |
| Protocol Negotiation | Behavior | 1. In the RequestSession message, client will specify the protocols, versions, and roles which it expects the **server** to support for this session. 2. The server MUST respond to this message using the OpenSession message, indicating which of the requested protocols and roles it can support. 3. The server MAY offer to support only some of the requested protocols. It MUST not offer to support any additional protocols. 4. The server MUST NOT change the requested role in its response. 5. The server MAY change the supported version of a protocol. 6. If the server is able to support the version the client requests, it MUST do so. 7. Client MAY CloseSession immediately if the server response does provide adequate functionality. |
| Session Survivability | Behavior | The following applies to all servers that are capable of maintaining history.   1. Server MUST maintain Session state if the websocket connection is dropped without a CloseSession message being receive. The specific information about session state will be define at the sub-protocol level. 2. Server MUST maintain this state for **1hr**, +/- 1 minute after disconnect. 3. On reconnection, Client MAY include the **etp-sessionid** HTTP header in the upgrade request. This effectively is a request to re-activate that session. 4. If the session is valid, Server MUST include the identical etp-sessionid HTTP header in upgrade response. 5. If the session is not valid, Server MUST deny the upgrade request with HTTP error code 404 (not found). 6. It is the server’s responsibility, using HTTP/S authentication to ensure that a request to re-activate a session is coming from an authorized agent. |



Figure : Protocol 0

The following interaction diagram describes the normal sequence of message for an ETP session:

1. Client begins with the standard Websocket handshake and the Server responds by doing the upgrade from HTTP to WebSockets. Refer to [RFC 6455](http://tools.ietf.org/html/rfc6455) for more information on the details of the handshake. Specific to the Energistics Transfer Protocol, the client must specify the Sec-Websocket-Protocol header value of **energistics-tp**, and the server must reply likewise. The client may also supply the custom header of **etp-encoding** with a value of **binary** or **json**. This specifies the Avro encoding style to be used for the life of the connection. If this header is not present, the encoding is assumed to be binary. All protocol header names and values are case-insensitive. Html5 web browser clients do not currently have the ability to add custom headers to a web socket request, and thus may include the etp-encoding header as a query request parameter. Servers must accept and process this value. In the example below, instead of GET / HTTP/1.1 the first line of the request would read GET /?exp-encoding=binary HTTP/1.1. If the server does not support the requested encoding, it SHOULD reject the upgrade request with HTTP status code 412 (Precondition Failed) and the client can try again (if it wishes) with the alternate etp-encoding value.
2. Client then send an RequestSession message. This consists of a list of sub-protocols which the client intends to use on this connection, along with a key/value map of configuration parameters for each sub-protocol. The names of allowable keys are strictly controlled by this specification , and defined on a per-protocol basis.
3. The requested protocols are started and message passing begins. The sequence and interaction diagrams for each are in the following sub-protocol sections.
4. If the server is not able to support all of the requested protocols, it may 1) immediately terminate the socket connection, or 2) respond with an list of protocols that it does support, leaving it up to the client to decide whether to continue with the session.

### Interface : IClient

IClient represents the interface that must be implemented from the client side of protocol 0.



Figure : IClient

#### Methods

| Method | Description | Type | Parameter Summary |
| --- | --- | --- | --- |
| requestSession | Request a new ETP Session with a Server |  | (in) eventData: RequestSession |
| onOpenSession | Handle OpenSession event from Server | event | (in) eventData: OpenSession |
| closeSession | Send CloseSession to server. |  | (in) eventData: CloseSession |
| onCloseSession | Handle CloseSession event from the server. Note that IClient can both send and receive CloseSession. | event | (in) eventData: CloseSession |

#### State Machine

The following diagram illustrates the definition of protocol 0 as a state machine, from the stand point of a client. This diagram incorporates both the connection/disconnection with websockets, as well as protocol 0.

From the standpoint of protocol 0, all of the ETP subprotocols can be thought of as concurrent substates. From an application point of view, there may be more complex state machines created that combine the subprotocols. For instance, an application may use the Discovery protocol to interactively determine what channels are available for streaming, and the use the ChannelData protocol to stream data on those channels. These higher level constructs are not specified as part of the protocol.



Figure 6 : ClientStateMachine

### Interface : IServer

IServer represents the server end of the interface that must be implemented for the core protocol.



Figure : IServer

#### Methods

| Method | Description | Type | Parameter Summary |
| --- | --- | --- | --- |
| onReqestSession |  | event | (in) eventData: RequestSession |
| openSession |  |  | (in) eventData: OpenSession |
| closeSession |  |  | (in) eventData: CloseSession |
| onCloseSession |  | event | (in) eventData: CloseSession |

#### State Machine



Figure : State Machine

## ChannelStreaming

**Stability:** : 3 - Stable

The Channel Streaming Protocol (Protocol 1) defines a set of messages for exchanging channel-oriented data, where a Channel is a Time or Depth series of individual data points.

### Requirements

| Requirement | Type | Description |
| --- | --- | --- |
| Message Order | Behavior | 1. Streaming data points MUST be sent in index order (as described in ChannelMetadta for the channel). 2. ChannelRangeRequest responses MUST be sent in index order. 3. The from and to indexes in a ChannelRangeRequest must be in index order. |
| Range Requests | Behavior | 1. All channels in a ChannelRangeRequest SHOULD have the same index (i.e. reference the same IndexMetadata from their ChannelMetadata). 2. Channels in a ChannelRangeRequest MAY have different indexes so long as the indexes a) use the same units and b) have the same direction (i.e. increasing or decreasing). |
| Session Survivability | Behavior | For the ChannelData Protocol:   1. Channel Metadata (including channelId) will remain valid, provided that all ChannelMetadata records have been received (remembering ChannelMetadata can be a multipart response). THIS ACTUALLY SEEMS PROBLEMATIC. Server has sent all ChannelMetadata and assumes channelId values are good, but client may not have received them. Worst case client re-sends ChannelDescribe message? 2. Multi-part responses to ChannelRangeRequest messsages will not be completed on session re-connect. If the consumer has not received all parts to a response, it must re-issue the ChannelRangeRequest. 3. On re-connect, producer MUST re-send all ChannelDataChange, ChannelDataDelete and ChannelDelete messages from the change log. There is no guarantee that the consumer will not receive one of these messages twice. Need discussion on exact meaning of change log, and the starting time for re-sent messages. Cannot be only from session disconnection time, since we have no idea what message got to the consumer. But, do we always want to send 24hrs worth of changes? Should we always send full 1 hour worth of changes? |



Figure : Protocol 1 - Simple Streaming

This diagram illustrates the minimal session interaction for a simple (i.e. protocol 1) producer of data. This the scenario that describes how a simple device can perform the functions of something like WITS-0 over this protocol.

Note that the notions of client and server are independent of this sequence. The roles of producer/consumer will have been requested and agreed in the session initiation sequence of protocol 0.

For simple streaming, the producer must provide a name value pair in the protocolCapabilities field of the SupportedProtocol record that indicates it will not accept requests to stream individual channels but will always send all of its channels. The name of the variable is **SimpleStreamer** and it mustd have a boolean value of **true**.

The producer must not send any data until the [Start](#BK_1114427721) message is received. The start message indicates that the consumer is ready to receive data and establishes any rate control or throttling parameters.

The producer will send at least one [ChannelMetadata](#BK_417954746) message, indicating the channels it will stream. For many producers this may be the only such message to be sent. However, if additional channels appear over time, it may send additional such messages.

The consumer will then send a ChannelStreamingStart message, with a zero-length array of channelIds.

After this, the producer can begin streaming [ChannelData](#BK_891737371).



Figure : Protocol 1 - Channel Streaming

This interaction diagram illustrates the full range of messages that might be exchanged over Protocol 1. Before streaming channels, or requesting specific ranges, the consumer will start by sending a ChannelDescribe message for at least one uri. The resulting ChannelMetadata messages provide a unique channelId (valid on for the life of the session) which is then used in subsequent ChannelStreamStart or ChangeRangeRequest messages.

### Interface : IChannelConsumer



Figure : RaLF Consumer as Client State

#### Methods

| Method | Description | Type | Parameter Summary |
| --- | --- | --- | --- |
| start |  |  | (in) eventData: Start |
| channelDescribe |  |  | (in) eventData: ChannelDescribe |
| channelStreamingStart |  |  | (in) eventData: ChannelStreamingStart |
| channelStreamingStop |  |  | (in) eventData: ChannelStreamingStop |
| channelRangeRequest |  |  | (in) eventData: ChannelRangeRequest |
| onChannelMetadata |  | event | (in) eventData: ChannelMetadata |
| onChannelData |  | event | (in) eventData: ChannelData |
| onChannelDataDelete |  | event | (in) eventData: ChannelDataDelete |
| onChannelDataChange |  | event | (in) eventData: ChannelDataChange |
| onChannelDelete |  | event | (in) eventData: ChannelDelete |

#### Session

### Interface : IChannelProducer



Figure : Channel Producer State

#### Methods

| Method | Description | Type | Parameter Summary |
| --- | --- | --- | --- |
| channelMetadata |  |  | (in) eventData: ChannelMetadata |
| onChannelRangeRequest |  | event | (in) eventData: ChannelRangeRequest |
| onStart |  | event | (in) eventData: Start |
| onChannelStreamingStart |  | event | (in) eventData: ChannelStreamingStart |
| onChannelStreamingStop |  | event | (in) eventData: ChannelStreamingStop |
| onChannelDescribe |  | event | (in) eventData: ChannelDescribe |
| channelDataChange |  |  | (in) eventData: ChannelDataChange |
| channelDataDelete |  |  | (in) eventData: ChannelDataDelete |
| channelDelete |  |  | (in) eventData: ChannelDelete |
| channelData |  |  | (in) eventData: ChannelData |

#### Channel Producer State

End

Section: Message\_Schemas

# Schemas

**Stability:** : n/a

This section is normative.

This section contains the formal UML and Avro Schema definitions of the messages that are exchanged as part of the Energistics Transfer Protocol. These are defined using the following UML stereotypes and conventions:

1. Enumeration - Enumerated values are defined in the schemas as a list of literal names, and serialized on the wire as an integer value. Avro schemas do not allow a bespoke integer to be associated with a given enumeration, and so they are order dependent. This implies a couple of things for design. First, the UML tool used for modeling must be capable of preserving this ordering, and schema authors must be careful to keep the ordering consistent between versions, to provide maximum interoperability.
2. Record - An Avro Record is more or less the same as a C or C++ struct. The Record stereotype is used to designate low level data types that are composed to create messages. For instance, the DateTime Record is used to define how a date is transferred in all messages.
3. Message - Represents a top-level message that can be sent between client and server. Messages are identical to Records in all ways, except that they are designated as being transferable as a top level element. There will be a one to one mapping between the messages used in a Protocol and the MessageTypes enumeration for that Protocol.
4. Union - Used to represent a type of a record which can be any one of a selected list. Each type is reflected in the UML as an attribute of the union class itself. Union more or less map to the xsd:choice element in XML Schemas.
5. Map - UML does not support maps very well natively. In our case, however, we can simplify because in Avro all maps have string keys. So in UML, a Map type is simply defined as a collection of type X, where X is the value types of the map, and the keys are assumed to be strings. This will be reflected in the Avro schema generation rules.

The Avro schemas, in JSON form, are produced automatically by the code generation process in Enterprise Architect. The built-in code generation process creates one .avsc file per class, in a folder structure that matches the package hierarchy. There is a second script that can be used to generate all of the schemas in a single .avpr (Avro Protocol) file. Note that while the .avpr format is a convenient way to place all of the schemas in a single file, we DO NOT use the Avro RPC protocol as part of the exchange specification.

The primitives used for attribute types in the UML class definitions are exactly those used in Avro. The set of primitive type names is:

* null: no value
* boolean: a binary value
* int: 32-bit signed integer
* long: 64-bit signed integer
* float: single precision (32-bit) IEEE 754 floating-point number
* double: double precision (64-bit) IEEE 754 floating-point number
* bytes: sequence of 8-bit unsigned bytes
* string: unicode character sequence

Conversion from the Avro schema to language-specific proxy classes is described later in this document.

## Energistics

**Stability:** : n/a

This is the root package, or namespace for all messages and data types in Energistics Transport Protocol. It is not expected that this namespace will contain any types or classes directly, it is just container for other namespaces. All schemas and protocol defined by any of the Energisitics SIGs or working groups MUST exist in this namespace, and extensions or customizations MUST exist in another namespace.

### Datatypes

**Stability:** : n/a

The datatypes package is intended to hold only low-level types that will be broadly re-used in various protocols.



Figure 13 : Datatypes

These are the lower-level datatypes defined for the protocol. They are only used as fields of messages, not messages in their own right.

#### Web Socket Headers

The majority of information exchange in ETP occurs through the Avro messages. However, a few things are transferred via HTTP headers. The initial web socket connection request makes use of the following headers:

1. The **Sec-WebSocket-Protocol** header is defined in RFC-6455 and SHALL be used to establish the use of the Energistics Transfer Protocol for this web socket connection. The value MUST be equal to the string "**energistics-tp**" (TODO - this needs to be registered with IANA).
2. In addition to the main protocol identifier, the connection request MAY supply a header indicating the encoding to be used for the life of the connection. The name of the header is the string "**etp-encoding**" and the value may be either the string "**binary**", or the string "**json**". If the header is not present, the server **MUST** assume a binary encoding.
3. The Client **MAY** supply a header named **etp-session**. The value of this header **MUST** be the guid of a previously established session with this server. If the session stills exists on the server, the server will reconnect the session according to the behavior describe in section ???.

There is currently no support in the web browser version of the websockets API for passing any custom headers (the Sec-WebSocket-Protocol header is supported by the API, but nothing else). For this reason, in order to support Html5 (i.e. browser-based) clients, servers **MUST** also examine the HTTP request variables (i.e. query string) for any of the **etp-xxxxx** values and use them, if present. If both are present, the server SHOULD ignore the HTTP headers and use the query string.

#### Record : Version

The Version record is used to identify a unique version of an ETP schema or protocol. The semantics of the individual fields of the record follow those that are generally defined for all Energistics data standards.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes",       "name": "Version",       "fields":       [           { "name": "major", "type": "int" },           { "name": "minor", "type": "int" },           { "name": "revision", "type": "int" },           { "name": "patch", "type": "int" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| major | A major version is one which may involve a significant change to all schemas, protocols, and business rules of a specification. | int | 1 | 1 |
| minor | A minor version will include significant changes to schemas, most probably with breaking changes. The overall protocols and approach should not change significantly. | int | 1 | 1 |
| revision | A revision may contain additions to existing schemas, but will not remove any schema elements. There may also be changes to enumerated types. | int | 1 | 1 |
| patch | A patch release can only involve minor changes, usually bug fixes to a release, and should not create breaking changes for other clients and servers on the same revision. | int | 1 | 1 |

#### Record : ArrayOfDouble

Convenience type representing an array of double precision floating numbers.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes",       "name": "ArrayOfDouble",       "fields":       [           { "name": "values", "type": { "type": "array", "items": "double" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| values | An array of double precision numbers. | double | 0 | n |

#### Record : DataAttribute

Structure for passing attributes associated with individual data points, such as quality, confidence, audit information, etc. Not used in Release 1.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes",       "name": "DataAttribute",       "fields":       [           { "name": "attributeId", "type": "int" },           { "name": "attributeValue", "type": "Energistics.Datatypes.DataValue" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| attributeId | Release 1 of ETP does specify any value metadata, and this field is reserved for future releases. | int | 1 | 1 |
| attributeValue | Release 1 of ETP does specify any value metadata, and this field is reserved for future releases. | DataValue | 1 | 1 |

#### Record : DateTime

The DateTime value type represents dates and times, with an optional timezone offset.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes",       "name": "DateTime",       "fields":       [           { "name": "time", "type": "long" },           { "name": "offset", "type": "float" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| time | The number of microseconds from the epoch - Jan 1, 1970. | long | 1 | 1 |
| offset | The time zone offset from UTC. | float | 1 | 1 |

#### Record : MessageHeader

The message header is the protocol control block sent at the beginning of every message. On the wire, every Message sent will contain this block first. From an Avro perspective, this can be thought of as the first member of every message, however it will normally be processed independently. This is to allow agents to inspect the protocol and message type fields to determine the appropriate serializer for the rest of the message.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes",       "name": "MessageHeader",       "fields":       [           { "name": "protocol", "type": "int" },           { "name": "messageType", "type": "int" },           { "name": "correlationId", "type": "long" },           { "name": "messageId", "type": "long" },           { "name": "messageFlags", "type": "int" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| protocol | This identifies the protocol for which the message is intended. The values must come from the [ProtocolID](#BK_421402570) enumeration. | int | 1 | 1 |
| messageType | The messageType contains the enumerated value (a protocol-specific value) for the accompanying message. Thus a client or server, can read the Message Type from the message header and then know which schema proxy must be used to decode the rest of the message. | int | 1 | 1 |
| correlationId | ETP generally does not follow a request/response pattern, thus the correlationId is used to allow servers and clients to match, asynchronously, related messages. For instance, an Exception message must have as its correlationId the number of the message which caused the exception to be raised. | long | 1 | 1 |
| messageId | The unique identifier for this message with the ETP session. | long | 1 | 1 |
| messageFlags | This is a bit map of flags that apply to a message. It has the following bits defined:   * 0x01 - This message is a part of a multi-message response. * 0x02 - This message is the final part of a multi-message response. | int | 1 | 1 |

#### Record : SupportedProtocol

Record to describe a protocol that is supported in a particular role by a given actor. Includes the protocol id and role. Used primarily in initial session negotiation to determine how a client and server will interact for a given session.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes",       "name": "SupportedProtocol",       "fields":       [           { "name": "protocol", "type": "int" },           { "name": "protocolVersion", "type": "Energistics.Datatypes.Version" },           { "name": "role", "type": "string" },           { "name": "protocolCapabilities", "type": { "type": "map", "values": "Energistics.Datatypes.DataValue" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| protocol | The ID of the protocol that this represents, as defined in the Energistics.Core.ProcotolID enumeration. | int | 1 | 1 |
| protocolVersion | The specific version of the protocol to be used. | Version | 1 | 1 |
| role | Most of the supported protocols involve two mutually exclusive roles. For example, when using the ChannelData protocols, a given actor will be either a "producer" or "consumer". The values expected for this string are defined by the sub-protocols. | string | 1 | 1 |
| protocolCapabilities | A name/value map of protocol-specific configuration or capability data. The key names, defaults, optionality, and expected data types will be defined as necessary by each protocol. Key names are case-insensitive in all cases. | DataValue | 0 | n |

#### ErrorCodes

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| ENOROLE | The agent does not support the requested role. | int | 1 | 1 |
| ENOSUPPORTEDPROTOCOLS | Server does not support any of the requested protocols. | int | 1 | 1 |
| EINVALID\_MESSAGETYPE | The message type ID is invalid for the given protocol. | int | 1 | 1 |
| EUNSUPPORTED\_PROTOCOL | Agent does not support the protocol identified in a message header. | int | 1 | 1 |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes",       "name": "ErrorCodes",       "symbols":       [           "ENOROLE",           "ENOSUPPORTEDPROTOCOLS",           "EINVALID\_MESSAGETYPE",           "EUNSUPPORTED\_PROTOCOL"       ]  } |

#### Protocols

This enumeration represents all of the known sub-protocols of the Energistics Transfer Protocol specification. The integer values for the enumeration members correspond directly to the value found in the [protocol](#BK_298527717) field of the [MessageHeader](#BK_870129603) record.

| Enumeration | Value | Description |
| --- | --- | --- |
| Core | 0 | Core protocol, supports connection management, re-connect buffer, exception management, and message acknowledgement. |
| Channel\_Data | 1 | The basic streaming protocol. This is used for simple devices (which can support only this protocol) and for publish/subscribe to send the channel data. |
| Channel\_Tabular | 2 | Protocol for retrieving a larger set of historical data from a channel provider, including channels that are no longer growing. |
| Discovery | 3 | Protocol for navigating a data provider to find the objects, channels, etc. for which it can provide data. |
| Object\_Store | 4 | Protocol for handling CRUD operations on a store. |
| Object\_Query | 5 | Protocol for querying specific data objects from a store. |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes",       "name": "Protocols",       "symbols":       [           "Core",           "Channel\_Data",           "Channel\_Tabular",           "Discovery",           "Object\_Store",           "Object\_Query"       ]  } |

#### union : DataValue

DataValue is the basic union that represents a single datum in a ChannelDataBlock. An array of these values makes up one 'row' in a data block.

##### Attributes

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| double |  | double | 1 | 1 |
| float |  | float | 1 | 1 |
| int |  | int | 1 | 1 |
| long |  | long | 1 | 1 |
| string |  | string | 1 | 1 |
| date |  | DateTime | 1 | 1 |
| null |  | null | 1 | 1 |
| vector |  | ArrayOfDouble | 1 | 1 |
| boolean |  | boolean | 1 | 1 |

#### ChannelData

**Stability:** : n/a

The ChannelData schema package is used to support both the Streaming (protocol 1) as well as Historical channel data (i.e. Log), protocol 2.



Figure : ChannelData - Common Types

##### Record : ChannelStreamingInfo

The ChannelStreamingStart information for a single channel. The ChannelStreamingStart message sends and array of these.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "ChannelStreamingInfo",       "fields":       [           { "name": "channelId", "type": "int" },           { "name": "startIndex", "type": "Energistics.Datatypes.ChannelData.StreamingStartIndex" },           { "name": "receiveChangeNotification", "type": "boolean" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelId | A URI for the subscription | int | 1 | 1 |
| startIndex | The requested starting index point for all channels in the subscription. | StreamingStartIndex | 1 | 1 |
| receiveChangeNotification | Indicates that the consumer wishes to receive notification of inserts, updates, and deletes on the channels of this subscription. Acceptance of a subscription with this flag indicates a guarantee that the producer will provide such notifications. | boolean | 1 | 1 |

##### Record : DataItem

A single data point on a channel is the data record used in the Streaming protocol.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "DataItem",       "fields":       [           { "name": "indexes", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.IndexValue" } },           { "name": "channelId", "type": "int" },           { "name": "value", "type": "Energistics.Datatypes.DataValue" },           { "name": "valueAttributes", "type": { "type": "array", "items": "Energistics.Datatypes.DataAttribute" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| indexes | The value of the indexes for this data point. The union members used MUST match the indexType of the index metadata for the associated channel.  The array MUST be of length 0, or the length of the corresponding index metadata array for the channelId. If the length is 0 then the value is determined from the previous item in the array of DataItem, which MUST have identical index metadata as the channelId of this record. | IndexValue | 0 | n |
| channelId | The identifier of the channel for this point, as received in a ChannelMetadata record. | int | 1 | 1 |
| value | The value of this data point. | DataValue | 1 | 1 |
| valueAttributes | Any qualifiers, such as quality, accuracy, etc. attached to this data point. Array of id/value pairs, where the ids and the values are described as part of this specification. Release 1 of ETP does specify any such value metadata, and this field is reserved for future releases. | DataAttribute | 0 | n |

##### Record : IndexMetadataRecord

Metadata for an index.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "IndexMetadataRecord",       "fields":       [           { "name": "indexType", "type": "Energistics.Datatypes.ChannelData.ChannelIndexTypes" },           { "name": "uom", "type": "string" },           { "name": "datum", "type": ["string", "null"] },           { "name": "direction", "type": "Energistics.Datatypes.ChannelData.IndexDirections" },           { "name": "mnemonic", "type": ["string", "null"] },           { "name": "description", "type": ["string", "null"] },           { "name": "uri", "type": "string" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| indexType |  | ChannelIndexTypes | 1 | 1 |
| uom | The units of measure for the index. In the case of time or relative time indexes this value is ignored, as the units are implicit from the index type (i.e. micro-seconds). For depth indexes, it must be a valid LengthUom from the Energistics UoM Specification. | string | 1 | 1 |
| datum | Describes the vertical datum for depth indexes. This field is reserved for future use, it's meaning is not yet formalized. | string | 0 | 1 |
| direction |  | IndexDirections | 1 | 1 |
| mnemonic | A mnemonic description of the index. This is an optional field; in the absence of a value, the string representation of the indexType enumeration should be considered the mnemonic. | string | 0 | 1 |
| description | Optional human readable description of the index. | string | 0 | 1 |
| uri | A URI for the index. This optional field allows for an index to reference a permanent object on the producer or server. | string | 1 | 1 |

##### Record : ChannelMetadataRecord

Describes metadata for one channel, corresponds roughly to the LogCurveInfo structure in WITSML. The ChannelMetadata message sends an array of these.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "ChannelMetadataRecord",       "fields":       [           { "name": "channelUri", "type": "string" },           { "name": "channelId", "type": "int" },           { "name": "indexes", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.IndexMetadataRecord" } },           { "name": "mnemonic", "type": "string" },           { "name": "dataType", "type": "string" },           { "name": "uom", "type": "string" },           { "name": "startIndex", "type": ["Energistics.Datatypes.ChannelData.IndexValue", "null"] },           { "name": "endIndex", "type": ["Energistics.Datatypes.ChannelData.IndexValue", "null"] },           { "name": "description", "type": "string" },           { "name": "status", "type": "Energistics.Datatypes.ChannelData.ChannelStatuses" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelUri | A uri for the channel. Must be unique within the context of a server. | string | 1 | 1 |
| channelId | Channel is a producer-defined integer identifier for the channel. Channel IDs are only unique or meaningful within a given session. The same channel URI may result in a different ID if you start a new session. | int | 1 | 1 |
| indexes | The description of the indexes associated with this channel. There MUST be at least one index description in the array. It is considered the primary index and MUST be the first index in the array. The values of the primary index MUST be unique within the channel. | IndexMetadataRecord | 1 | n |
| mnemonic | The mnemonic for the channel. In WITSML this corresponds directly to the mnemonic in LogCurveInfo. | string | 1 | 1 |
| dataType | The dataType is the nominal data type for the channel (i.e. double, float, int, etc.) | string | 1 | 1 |
| uom | The unit of measure for the channel. All DataItems and DataRows will send data using this UoM. The ChannelData protocol does not support conversion to a consumer-requested system of measurement. | string | 1 | 1 |
| startIndex | The first (lowest) recorded primary index value for the channel. | IndexValue | 0 | 1 |
| endIndex | The last (highest) recorded primary index value for the channel. For active channels, this value is only good at the time the meatadata is sent. | IndexValue | 0 | 1 |
| description | Human readable description of the channel. | string | 1 | 1 |
| status | Current status of this channel, any changes to this status during session will result in ChannelStatusChange notification. | ChannelStatuses | 1 | 1 |

##### Record : ChannelRangeInfo

Sent from Consumer to Producer to request data over a specific range for one or more Channels.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "ChannelRangeInfo",       "fields":       [           { "name": "channelId", "type": "int" },           { "name": "startIndex", "type": "Energistics.Datatypes.ChannelData.IndexValue" },           { "name": "endIndex", "type": "Energistics.Datatypes.ChannelData.IndexValue" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelId |  | int | 1 | 1 |
| startIndex |  | IndexValue | 1 | 1 |
| endIndex |  | IndexValue | 1 | 1 |

##### ErrorCodes

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| EINVALID\_URI | Sent from producer to consumer when a request is made to describe a URI which does not exist on the producer. | int | 1 | 1 |
| EINVALID\_CHANNELID | Send from producer to consumer when operations are requested on a channel that does not exist. | int | 1 | 1 |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "ErrorCodes",       "symbols":       [           "EINVALID\_URI",           "EINVALID\_CHANNELID"       ]  } |

##### ChannelStatuses

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| Active |  |  | 1 | 1 |
| Inactive |  |  | 1 | 1 |
| Closed |  |  | 1 | 1 |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "ChannelStatuses",       "symbols":       [           "Active",           "Inactive",           "Closed"       ]  } |

##### Roles

These are the classes of actors in all of the Streaming Channel Data protocols. They can be either **Consumers** of data or **Producers** of data. Specifically, this role is independent of the client/server roles associated with the web socket connection. Both clients and servers can be either producers or consumers.

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| Producer |  |  | 1 | 1 |
| Consumer |  |  | 1 | 1 |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "Roles",       "symbols":       [           "Producer",           "Consumer"       ]  } |

##### IndexDirections

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| Increasing |  |  | 1 | 1 |
| Decreasing |  |  | 1 | 1 |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "IndexDirections",       "symbols":       [           "Increasing",           "Decreasing"       ]  } |

##### ChannelIndexTypes

The kinds of indexes that are allowed for channels. These are currently limited to Time and Depth. Depth indexes must be related to a datum, which must be provided in the ChannelMetadata.

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| Time |  |  | 1 | 1 |
| Depth |  |  | 1 | 1 |
| ElapsedTime |  |  | 1 | 1 |

| Avro Schema |
| --- |
| {       "type": "enum",       "namespace": "Energistics.Datatypes.ChannelData",       "name": "ChannelIndexTypes",       "symbols":       [           "Time",           "Depth",           "ElapsedTime"       ]  } |

##### union : StreamingStartIndex

This union represents the various ways in which a subscriber can request the starting index for a subscription to a set of channels.

###### Attributes

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| LatestValue | Each channel in the subscription will be streamed starting with it's latest measured value. | null | 1 | 1 |
| IndexCount | Begin streaming the channel from 'IndexCount' values prior to latest value. | int | 1 | 1 |
| IndexValue | Begin streaming all channels from the specified index value. If the subscription does not support an index of the specified type (i.e. Depth or Time) the producer must signal an exception and the subscription will not be valid. | IndexValue | 1 | 1 |

##### union : IndexValue

Every data item must begin with this element, which cannot be null, and cannot be anything except for the acceptable types of Indexes. Thus, in Avro it is represented as a union of DateTime, double, and float.

###### Attributes

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| time | The absolute time index of the data point. | DateTime | 1 | 1 |
| double | The measured depth of the data point, in the units and relative to the datum specified in the channel's metadata. | double | 1 | 1 |
| elapsedTime | The elapsed time, in microseconds of the data point, relative to any starting time datum specified in the channel metadata. | long | 1 | 1 |

#### ChannelDataFrame

**Stability:** : n/a

##### Record : DataRow

Represents one row in a data block, as used by the historical protocol. Each row in the data array MUST have the same number of elements as the channel array.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.ChannelDataFrame",       "name": "DataRow",       "fields":       [           { "name": "index", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.IndexValue" } },           { "name": "data", "type": { "type": "array", "items": "Energistics.Datatypes.DataValue" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| index |  | IndexValue | 1 | n |
| data |  | DataValue | 1 | n |

#### Object

**Stability:** : n/a

The Object namespace contains datatypes for working with static objects.



Figure : Object - Common Types

Common Types

##### Record : DataObject

Record structure to carry a single data object.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.Object",       "name": "DataObject",       "fields":       [           { "name": "namespace", "type": "string" },           { "name": "version", "type": "string" },           { "name": "objectType", "type": "string" },           { "name": "contentType", "type": "string" },           { "name": "contentEncoding", "type": "string" },           { "name": "data", "type": "bytes" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| namespace | The namespace of the schema for the contained object. Currently a WITSML/PRODML/RESQML namspace. Other could be supported in the future. | string | 1 | 1 |
| version | The version of the schema for the contained object, exactly as written in the XML data. | string | 1 | 1 |
| objectType | The top level (singular) element name of the contained XML data. Normally will map one-to-one with an obj\_Xxx schema in one of the Energistics specification, however it will not contain the obj\_, but will be exactly the name of the element itself. So, for example, the WITSML well object would be identified by the string "well". | string | 1 | 1 |
| contentType | Currently must be the string "text/xml". | string | 1 | 1 |
| contentEncoding | Indicates if compression is used. Currently must be either an empty string or the string "gzip". | string | 1 | 1 |
| data | A byte array containing the encoded object, either as a utf-8 string or its gzipped form. | bytes | 1 | 1 |

##### Record : Component

Not a message, the data structure for the resource descriptions in the previous message. This is the one that requires the most thought. The Uri provides the identification of the child object., the name is a human-readable string for a GUI, and I added subscribable just because it was needed for my prototype. I’m not sure it belongs here, but maybe.

The one thing I would like to add is some sort of ‘node type’ information. This would be something closely tied to our energistics information model, so that node types of ‘well’, ‘wellbore’, ‘log’, etc. would have unambiguous meanings.

It might also be useful to have ‘hasChildren’ attribute that could be used in the UI to determine whether or not to make a node expandable. I’m not 100% in favor of this idea, because it could also be an expensive operation on the server to actually know whether there are children and it could really slow things down.

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Datatypes.Object",       "name": "Component",       "fields":       [           { "name": "uri", "type": "string" },           { "name": "name", "type": "string" },           { "name": "subscribable", "type": "boolean" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uri |  | string | 1 | 1 |
| name |  | string | 1 | 1 |
| subscribable |  | boolean | 1 | 1 |

### Protocol

**Stability:** : n/a

This package is the root namespace for all sub-protocols of the Energistics Transfer Protocol. Each child package represents a protocol and has a UML Tagged Value of the Protocol ID, and each in this section will have the <<Message>> stereotype. The attributes of the message can be either primitives or any of the structured Record classes from the Datatypes package. Each Message is decorated with the following UML Tagged Values

* MessageTypeID (the integer value of this message type)
* Correlation Id Usage. Specifies how the correlationId is used for this message. A value of "n/a" means that the correlationId is not used for this message and MUST be set to a value of 0 in the message header.
* MultiPart. Specifies if the Message can be multipart.

#### Core

**Stability:** : 3 - Stable



Figure : Core

##### Message : RequestSession

Sent by the Client to request a new session with a Server.

**Message Type ID**: 1

**Correlation Id Usage**: n/a

**Multi-part**: False

| Constraint | Details | Description |
| --- | --- | --- |
| Within a given request/response for protocol support, protocol id must be unique. | inv: self.requestedProtocols->isUnique(p | p.protocol) | inv: self.requestedProtocols->isUnique(p | p.protocol) |

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Core",       "name": "RequestSession",       "messageType": "1",       "protocol": "0",       "fields":       [           { "name": "requestedProtocols", "type": { "type": "array", "items": "Energistics.Datatypes.SupportedProtocol" } },           { "name": "applicationName", "type": "string" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| requestedProtocols | An array of protocol IDs which the client expects to communicate on for this session. If the server does not support all of the protocols, the client may or may not continue with the protocols which are supported. | SupportedProtocol | 1 | n |
| applicationName | The string by which the client identifies itself, normally a software product or systemname. This may or may not include a version, and the format is entirely application dependent. Vendors are encouraged to identify their company name somehow as part of this string. | string | 1 | 1 |

##### Message : OpenSession

A positive response from the Server to the RequestSession message, providing the Session ID and supported sub-protocols.

**Message Type ID**: 2

**Correlation Id Usage**:

MUST contain the message id of the RequestSession message that resulted in this session being created.

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Core",       "name": "OpenSession",       "messageType": "2",       "protocol": "0",       "fields":       [           { "name": "applicationName", "type": "string" },           { "name": "sessionId", "type": "string" },           { "name": "supportedProtocols", "type": { "type": "array", "items": "Energistics.Datatypes.SupportedProtocol" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| applicationName | The string by which the server identifies itself. This may or may not include a version, and is entirely application dependent. Vendors are encouraged to identify their company name somehow as part of this string. | string | 1 | 1 |
| sessionId | An Identifier for this session. This must be a uuid as specified by [RFC 4122](http://www.ietf.org/rfc/rfc4122.txt). The UUID must be formatted as a string, in the form commonly known as "windows registry format", without the enclosing curly braces. Example:    **65CCC68E-C0B4-454E-8757-3284F94AE861** | string | 1 | 1 |
| supportedProtocols | An array of identifiers of the sub-protocols supported by the server. | SupportedProtocol | 1 | n |

##### Message : CloseSession

Sent from either consumer or producer to close the current session. Receiver of this message should respond by doing a clean shutdown of the WebSocket connection.

**Message Type ID**: 5

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Core",       "name": "CloseSession",       "messageType": "5",       "protocol": "0",       "fields":       [           { "name": "reason", "type": ["string", "null"] }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| reason |  | string | 0 | 1 |

##### Message : ProtocolException

A message used to indicate an error condition in a protocol. This message MUST NOT be used to indicate general failures of low level protocols (such as web sockets, HTTP, or TCP/IP) on which the Energistics TransferProtocol depends. A given protocol may dictate that Error information is sent as a stand-alone message, or may also define situations where error information is sent embedded in another message.

**Message Type ID**: 1000

**Correlation Id Usage**:

MUST contain the message id of the message that caused the exception to be raised.

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Core",       "name": "ProtocolException",       "messageType": "1000",       "protocol": "0",       "fields":       [           { "name": "errorCode", "type": "int" },           { "name": "errorMessage", "type": "string" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| errorCode | Unique number, defined by the specification for an error. Error numbers are defined separately for each sub-protocol, thus to unambiguously identify an error message, you must combine the ProtocolID with errorCode. | int | 1 | 1 |
| errorMessage | A string representation of the error message. TODO - define language requirements for the description, describe how placeholders and substitution might work. | string | 1 | 1 |

##### Message : Acknowledge

This message is used in any Protocol where specific acknowledgement of receipt of a message is required. In general, the ETP does not follow a Request/Response pattern, so this is not often needed. There is no content to this message. The correlationId in the message header must be the id of the message that is being acknowledged.

**Message Type ID**: 1001

**Correlation Id Usage**:

MUST contain the message id of the message who's receipt is being acknowledged.

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Core",       "name": "Acknowledge",       "messageType": "1001",       "protocol": "0",       "fields":       [         ]  } |

#### ChannelStreaming

**Stability:** : 3 - Stable

The ChannelStreaming protocol supports the streaming of channel data.



Figure : ChannelStreaming

##### Message : Start

Start the protocol. For unsubscribed producers, this message indicates they should start streaming data. Includes parameters to throttle size and rate of messages.

**Message Type ID**: 0

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "Start",       "messageType": "0",       "protocol": "1",       "fields":       [           { "name": "maxMessageRate", "type": "int" },           { "name": "maxDataItems", "type": "int" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| maxMessageRate | Do not send two channel messages within a smaller window that this (in milliseconds) | int | 1 | 1 |
| maxDataItems |  | int | 1 | 1 |

##### Message : ChannelDescribe

Sent from Consumer to Producer to request metadata about one or more channels, specified by URI. The URI may refer to a single channel or a higher level object - such as a Well, Wellbore, or Log. Producer will respond with (TODO - one or more?) ChannelMetadata messages.

**Message Type ID**: 1

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelDescribe",       "messageType": "1",       "protocol": "1",       "fields":       [           { "name": "uris", "type": { "type": "array", "items": "string" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uris |  | string | 1 | n |

##### Message : ChannelMetadata

Send from Producer to Consumer to describe channels which may be streamed to the Consumer in future messages.

**Message Type ID**: 2

**Correlation Id Usage**:

MUST contain the id of the ChannelDescribe message which caused this metadata to be sent.

**Multi-part**: True

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelMetadata",       "messageType": "2",       "protocol": "1",       "fields":       [           { "name": "channels", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.ChannelMetadataRecord" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channels |  | ChannelMetadataRecord | 1 | n |

##### Message : ChannelData

Contains an array of <index,value> 2-tuples for one or more channels. Unlike the ChannelDataBlock, which is structured like the <data/> element of a WITSML Log, there is no requirement that any given channel appear in an individual ChannelData message, or that a given channel appear only once in ChannelData message (i.e. a range of several index values in one message).

**Message Type ID**: 3

**Correlation Id Usage**:

If this ChannelData is a response to a ChannelRangeRequest MUST be equal to the message id of the ChannelRangeRequest. Otherwise, MUST be 0.

**Multi-part**: True

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelData",       "messageType": "3",       "protocol": "1",       "fields":       [           { "name": "data", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.DataItem" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| data |  | DataItem | 1 | n |

##### Message : ChannelStreamingStart

Sent from Consumer to Producer to request that the Producer begin streaming one or more channels. The startIndex value is union possible points to begin the stream (latest, n-points back from now, etc.), see [ChannelStreamingInfo](#BK_1144288842) for more information on these options.

**Message Type ID**: 4

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelStreamingStart",       "messageType": "4",       "protocol": "1",       "fields":       [           { "name": "channels", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.ChannelStreamingInfo" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channels |  | ChannelStreamingInfo | 1 | n |

##### Message : ChannelStreamingStop

Sent from Consumer to Producer to request that streaming be discontinued on one or more channels.

**Message Type ID**: 5

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelStreamingStop",       "messageType": "5",       "protocol": "1",       "fields":       [           { "name": "channels", "type": { "type": "array", "items": "int" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channels | An array of the channels to stop streaming. | int | 1 | n |

##### Message : ChannelDataChange

Sent from Producer to Consumer to notify the consumer of changed date points on the Channel. Sent for all changes, whether or not this particular Consumer has previously been sent the original point. All data items must belong to the same channel, which MUST be specified in the channelId field of the main message. Processors MUST ignore the channelId field of the data items.

The message should be processed in the following way:

1. Delete the data between, and inclusive of, startIndex and endIndex.
2. Insert the points in the provider data element in its place.

This message is also used for data deletions, in which case the data element would simply have zero records.

**Message Type ID**: 6

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelDataChange",       "messageType": "6",       "protocol": "1",       "fields":       [           { "name": "channelId", "type": "int" },           { "name": "startIndex", "type": "Energistics.Datatypes.ChannelData.IndexValue" },           { "name": "endIndex", "type": "Energistics.Datatypes.ChannelData.IndexValue" },           { "name": "data", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.DataItem" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelId | The channel to which all changes to apply. The channelId in all data records MUST be the same as this value. | int | 1 | 1 |
| startIndex | The starting index of the data to be replaced by the supplied data. | IndexValue | 1 | 1 |
| endIndex | The ending index of the range to be replaced by the supplied data. | IndexValue | 1 | 1 |
| data | An array of data items to be replaced. An empty array should be sent if a complete deletion of points in the range is desired. | DataItem | 0 | n |

##### Message : ChannelDelete

Sent from Producer to Consumer to indicate that a Channel is no longer actively streaming data. Producer MUST NOT send additional ChannelData messages on this channel once it has been deleted. The optional deleteReason should be a human readable description of why the channel is being deleted. At this point the Channel's ID is retired for the balance of the session. Producer MAY reactivate a Channel with the same URI

**Message Type ID**: 8

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelDelete",       "messageType": "8",       "protocol": "1",       "fields":       [           { "name": "channelId", "type": "int" },           { "name": "deleteReason", "type": ["string", "null"] }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelId | The ID of the channel to be deleted. | int | 1 | 1 |
| deleteReason |  | string | 0 | 1 |

##### Message : ChannelRangeRequest

Sent from Consumer to Producer to request data over a specific range for one or more Channels.

**Message Type ID**: 9

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelRangeRequest",       "messageType": "9",       "protocol": "1",       "fields":       [           { "name": "channelRanges", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.ChannelRangeInfo" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelRanges |  | ChannelRangeInfo | 1 | n |

##### Message : ChannelStatusChange

**Message Type ID**: 10

**Correlation Id Usage**: n/a

**Multi-part**: False

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelStreaming",       "name": "ChannelStatusChange",       "messageType": "10",       "protocol": "1",       "fields":       [           { "name": "channelId", "type": "int" },           { "name": "status", "type": "Energistics.Datatypes.ChannelData.ChannelStatuses" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channelId |  | int | 1 | 1 |
| status |  | ChannelStatuses | 1 | 1 |

#### ChannelDataFrame

**Stability:** : 1 - Experimental

The Historical protocol is used for sending a data 'frame' for complete well logs.



Figure : Tabular

##### Message : RequestChannelData

**Message Type ID**: 1

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelDataFrame",       "name": "RequestChannelData",       "messageType": "1",       "protocol": "2",       "fields":       [           { "name": "uri", "type": "string" },           { "name": "fromIndex", "type": ["Energistics.Datatypes.ChannelData.IndexValue", "null"] },           { "name": "toIndex", "type": ["Energistics.Datatypes.ChannelData.IndexValue", "null"] }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uri |  | string | 1 | 1 |
| fromIndex |  | IndexValue | 0 | 1 |
| toIndex |  | IndexValue | 0 | 1 |

##### Message : ChannelMetadata

Send from Producer to Consumer to describe channels which may be streamed to the Consumer in future messages.

**Message Type ID**: 3

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelDataFrame",       "name": "ChannelMetadata",       "messageType": "3",       "protocol": "2",       "fields":       [           { "name": "indexes", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.IndexMetadataRecord" } },           { "name": "channels", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelData.ChannelMetadataRecord" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| indexes |  | IndexMetadataRecord | 0 | n |
| channels |  | ChannelMetadataRecord | 1 | n |

##### Message : ChannelDataFrame

Sent from Producer to Consumer, this is the main data transfer message for logs, or any channel report that is oriented in tabular fashion, with rows of aligned data on index values. The size of the channel array must match the size of every row in the data array. The data array may contain nulls, which use only 1 byte in the data row.

**Message Type ID**: 4

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.ChannelDataFrame",       "name": "ChannelDataFrame",       "messageType": "4",       "protocol": "2",       "fields":       [           { "name": "channels", "type": { "type": "array", "items": "int" } },           { "name": "data", "type": { "type": "array", "items": "Energistics.Datatypes.ChannelDataFrame.DataRow" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| channels | An array of channelIDs (integers) of the channels that are contained in each of the contained data rows. | int | 1 | n |
| data |  | DataRow | 1 | n |

#### Discovery

**Stability:** : 1 - Experimental

The discovery protocol includes messages for discovering the contents of a store or server.



Figure : Discovery

Discovery

##### Message : Resource

The async response message to the Get, containing the child URIs. In addition to the correlation ID in the message header, it also has the original URI that was requested, as a convenience.

**Message Type ID**: 2

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Discovery",       "name": "Resource",       "messageType": "2",       "protocol": "3",       "fields":       [           { "name": "uri", "type": "string" },           { "name": "components", "type": { "type": "array", "items": "Energistics.Datatypes.Object.Component" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uri | The original uri for which this message contains children. | string | 1 | 1 |
| components | A collection of child, or related URIs, which can in turn be enumerated. | Component | 1 | n |

##### Message : Get

A request to enumerate a uri. Every ETP server must support a URI of “/”. The server will either send the Resource message, below, or will send an exception message saying the URI doesn’t exist, isn’t available, etc.

Note that because this is the discovery protocol (not static object or query) the meaning of the ‘Get’ message is unambiguously ‘get child URIs’. So if I do a Get on /well(‘23423423432423’), I will get back it’s wellbores, not the static well object itself.

**Message Type ID**: 1

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Discovery",       "name": "Get",       "messageType": "1",       "protocol": "3",       "fields":       [           { "name": "uri", "type": "string" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uri |  | string | 1 | 1 |

#### Store

**Stability:** : 1 - Experimental



Figure : Store

##### Message : GetFromStore

Get a single object from the store, by its URI. For growing objects, the growing portion is NOT returned. If the object is actively growing, the records are retrieved using the Streaming protocol; for inactive, or not currently objects, it is retrieved using the Historical protocol.

**Message Type ID**: 0

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Store",       "name": "GetFromStore",       "messageType": "0",       "protocol": "4",       "fields":       [           { "name": "uri", "type": "string" }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uri |  | string | 1 | 1 |

##### Message : AddToStore

Adds an object to the store. Uses 'Upsert' semantics, where update and insert use the same message, and the object is created for the first time if it does not exist.

The initial support is for the static objects defined for WITSML 1.4.1

**Message Type ID**: 1

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Store",       "name": "AddToStore",       "messageType": "1",       "protocol": "4",       "fields":       [           { "name": "data", "type": { "type": "array", "items": "Energistics.Datatypes.Object.DataObject" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| data |  | DataObject | 1 | n |

##### Message : DeleteFromStore

Deletes one or more objects from a store.

**Message Type ID**: 2

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Store",       "name": "DeleteFromStore",       "messageType": "2",       "protocol": "4",       "fields":       [           { "name": "uri", "type": { "type": "array", "items": "string" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| uri | The URIs of objects to be deleted. | string | 1 | n |

##### Message : Object

**Message Type ID**: 3

**Correlation Id Usage**: n/a

**Multi-part**: n/a

| Avro Schema |
| --- |
| {       "type": "record",       "namespace": "Energistics.Protocol.Store",       "name": "Object",       "messageType": "3",       "protocol": "4",       "fields":       [           { "name": "dataObjects", "type": { "type": "array", "items": "Energistics.Datatypes.Object.DataObject" } }       ]  } |

| Attribute | Description | Data Type | Min | Max |
| --- | --- | --- | --- | --- |
| dataObjects |  | DataObject | 1 | n |

#### Query

**Stability:** : 1 - Experimental

End